IFIP WORLD CONFERENCE ON COMPUTER EDUCATION 1970
24—28 August 1970, in Amsterdam, The Netherlands

PART THREE

USE OF COMPUTERS IN EDUCATION

A GENERAL PURPOSE SYSTEM FOR COMPUTER-COMMUNICATION BASED INSTRUCTION

Joseph Bordogna, David K. Hsiao, Noah S. Prywes

1. INTRODUCTION

The traditionally employed instruction-by-lecture system cannot meet the increasing demands of education without an adverse effect on quality and effectiveness. There are more students at all levels; there are greater distances within and between educational complexes; and there has been a great increase in course content as well as curriculum. There has also been an increasingly difficult communication problem between teacher and student, as well as in the use of immense library collections or complex laboratory procedures. In short, the lecture system is hard pressed in its ability to meet effectively the modern requisites of education which ensure social and economic progress.

The new technologies available today are potentially capable of evolving a radically new instructional methodology. Such an instructional system must be spontaneous in its response to a class or to an individual student who is able to indicate desire for review, digression, accelerated or decelerated progress in pursuing subject matter. Another requirement, very important for economic and staffing reasons, is that material prepared (or referenced) by teachers should be completely and automatically processed, stored, and retrieved upon demand. The technological and intellectual difficulties related to these two requirements have been the subject of research in laboratories and universities for the past decade. This background indicates that ample technology and methodology is at hand to proceed with a synthesizing effort leading to the realization of a truly novel instructional system.

The traditional teaching of electrical engineering in undergraduate studies has undergone some recent, notable changes, Rubinoff (1968), Bordogna (1969). The Cosine (Computer Science in Electrical Engineering) Committee of the Commission on Engineering Education, U.S.A. National Academy of Engineering, Cosine (1967), has recognized the importance of the digital computer as an "educational software system" which can significantly aid and strengthen the curricula of computer sciences in particular and electrical engineering in general. At the same time, the Committee points out that there is no general software available for such a purpose.

There has also been increased motivation toward developing a unified thread of basic engineering concepts through the entire undergraduate curriculum. This idea has become so intriguing that it has been extended into the pre-college educational system of the U.S.A. through the Engineering Concepts Curriculum Project (ECCP) of the Commission on Engineering Education, National Academy of Engineering, ECCP (1969).

We believe that a general purpose undergraduate engineering educational software system can be realized and also that its general applicability can be based on fundamental engineering concepts. The Computer-Communication Based Instructional System, already in a developing state on an RCA Spectra 70/46 computer system at the Moore School, will provide these mechanisms. Specific objectives of the research underway and a description of the system are discussed below.

2. DESCRIPTION OF COMPUTER SOFTWARE AND REPOSITORY MATERIAL

The electrical engineering concepts material includes a collection of program modules and their associated data to be placed in the computer repository. Additional tutorial data is associated with the modules containing instructional information about the use, presentation, and understanding of these modules. Collectively, these modules, data and software means include

(a) commonly used mathematical routines:
E.g., for finding roots of polynomials and solving ordinary differential equations. Some of these routines are provided by the computer manufacturer; however, the use of these routines is hindered by the necessity of complicated and tedious program preparation, namely, setting up the job control cards. With the simple and versatile new language, routines could then be called in for execution directly by students and instructors.

(b) programs related to the analysis and simulation of basic electrical engineering problems:
They include the areas of transient analysis, frequency response, Fourier analysis, and network analysis. They also include special purpose simulation programs such as MIMIC, CSMP, ECAP, BLODI, and JOBSHOP.

(c) semantic and syntactic tools for automatic or semi-automatic processing of indexing, classification, storage and retrieval, as well as text editing of instructor's material and student work.

(d) utility programs:
E.g., for grading students' examinations. The on-line facility for student-computer and instructor-computer interactions will be made available. This requires considerable re-orientation of current teaching methods and approaches. This teaching and learning experience will be used as a testbed from which conclusions about usefulness and further explorations are derived.

(e) a relatively simple and versatile language for students and instructors to communicate with the software: It could be used to specify input and output material, and initiate computation and manipulation of data. Problems in undergraduate studies are not restricted to numerical calculations; they involve non-numerical manipulations, such as two dimensional graphic display and model simulation. Thus the commonly adopted drill in a numerical-calculation-oriented language such as FORTRAN may not be entirely justified. To teach students additional problem-oriented languages for simulation, string manipulation and graphic display would overburden the students as well as the instructors. A language with some basic and versatile features is therefore needed for students and instructors alike.

(f) a repository of instructional material of fundamental importance to the undergraduate engineering curriculum:
The material should include lecture notes, diagrams, charts, and pictures which play a major role in introducing basic concepts in engineering to the student. Without the aid of the software, a great deal of the instructor's effort is expended in preparation of the material, (and subsequent lengthy presentation to the student) since material taught must be updated continuously to keep up with the advances in research and development. With the availability of the repository, the situation would be changed. Basic material could now be shared among instructors. Duplicate effort in updating and preparation could be eliminated. Automated lectures and home preparation would now require team work and pooling of talents and expertise. Thus, both the quality of the material and the coherency of presentation are greatly enhanced. Furthermore, there would be more time available for personal interaction with students in guiding them through the material.

(g) an automatic facility whereby a user of the software does not have to be concerned with the management, storage and retrieval, execution and parameter-passing, and formation of complex program and data structures of his accumulated modules, data, and instructional material.

3. DESCRIPTION OF COMPUTER-COMMUNICATION BASED SYSTEM

As conceived in general form, a graphical portrayal of the computer-communication based instructional system is shown in Fig. 1. Although not in itself a new idea, it has a number of distinguishing features of which, probably, the most important is the intermediate repository between the input of educational material and its dissemination as output. Namely, the teacher in preparing lectures or other instructional material, addresses himself to the repository where the material is being received, updated, and filed away. The teacher may continuously refer to this material, modify it, and perfect it. The users, in turn, also communicate with the repository, receiving display of instructional material via fixed service, closed circuit, or broadcast television in response to demands placed via keyboard terminals and telephone communications.

The role of the teacher is considerably modified when the subject matter of a course is presented in this manner and not from the teacher himself as it has been done traditionally. There is a new dual role for the teacher. On the one side of Fig. 1 he is the originator and curator of instructional material – greatly aided by automation in the preparation and selection of material. As shown, he may specify the input of a variety of documents which serve as instructional material to be placed in the repository; he may prepare audio-visual material including sketches that serve as slides or animated movies to be shown during class instruction or study sessions; and finally, via conventional television programs, entire lectures may be placed in the repository. The other role for the teacher is at the output end of Fig. 1 where he is relieved of the traditional lecturing to enable him to plan better the presentation of educational material. He uses a keyboard to search and place demands for the display of chosen material. This type of system permits the teacher to give individual attention to students, guide them and encourage them – a function for which there is a great need and demand but inadequate time in the present educational structure. This alteration in the role of the teacher will require much experimentation, study, and adjustment to account for the various impacts.

The central computer in Fig. 1 serves the dual purpose of input processing and demand searches. The similarity of these functions to those of a computer serving a library is evident. However, from the start we should like to avoid the costly and staff-demanding functions in which even the most modern libraries are engaged. Therefore, we shall strive toward performing almost the entire input-processing by the computer. For example the computer could aid the instructor or the staff in adding or extracting alphanumeric information in conjunction with input of all audio-visual material. This information could be scanned automatically and translated into digital form to serve as a basis for indentification and future retrieval of the material. It is advocated here that the most advanced approaches to storage and retrieval be employed. For instance, the indexing process based on the digital portion of the information should be entirely performed by the computer, utilizing content analysis and automatic indexing techniques, Prywes (1970). It should be based on the posteriori indexing approach; the vocabulary used should not be controlled. Provisions could be made for authorized staff to freely add or delete index terms assigned to material in the repository. This vocabulary of index terms should also be automatically arranged in a dynamic classification to provide the structure for interactive man-machine refinement of search queries to pinpoint the area of interest of the demands. The methodology developed at the Moore School effectively avoids the manual effort involved in indexing, cataloging, and maintaining indexing vocabularies, Litofsky (1969), Prywes (1964). Finally, reliance should be placed on the interactive man-computer search which offers the best potential for effective retrieval. In time, such a storage and retrieval system will become truly useful as an extension of human memory and recall.

Documents referenced or used as texts by instructors in college courses constitute a significant portion of college library. The availability of this material at remote terminals is tantamount to "exporting" the library to users scattered over the areas which are being serviced. The duplication of major portions of libraries in microform and its computer-

controlled storage and retrieval appears to be both practical and economical in the foreseeable future. One outstanding problem is the appropriate identification and retrieval of the material. While the documents themselves may be handled entirely as microform images, certain key portions would have to be translated into digital form; this includes information which is typically on library catalog cards such as title, author, publisher, dates, etc., much of which may be derived from the front page. Thus, this information may be extracted from the document semi-automatically, incorporating manual keypunching and editing with the digitizing by scanning of entire pages and the employment of character recognition processes. (Significant advances in recent years in the recognition of printed characters of greatly varying fonts are encouraging). Content analysis techniques can then be applied to the digital information extracted from the incoming document to perform automatic indexing of subject matter based on titles, abstracts, or similar information.

The processed information would then be associated with the images and stored together while maintaining catalog entries, and thesauri of index words arranged according to automatically generated classification systems. The dictionaries would include additional information, about authors, instructors and the courses. Access to the information, for inspection, dissemination or modification would be limited only to those authorized, Hsiao (1968). Extensive work on these techniques has been conducted at the Moore School and experimental collections automatically-indexed and automatically-classified have been maintained in computer mass storage, McEowen (1969). Figure 2 diagrams the process of document input flow.

The search procedures used within such a system are included within directories and they aid the searcher in the interactive process of browsing through the collection to retrieve desired information. This approach is applicable, not only to published material, but also to research data which constitutes much of the information used in instruction. This applies equally to many areas of University instruction ranging from the natural sciences to engineering, medicine, and the social sciences. The latter type of information is frequently locally generated and may be captured in digital form at the source at the time of initial transcription. In this form it may then be subject to extensive computer analysis, formatting, and editing during the preparation of responses to demand searches.

The repository of instructional information is located in automated mass storage devices. Three types of storage media are involved: digital, video signals, and microform inmages. The digital information consists primarily of directories and catalogs. The video and image information consists primarily of documents places totally or partially in storage.

The manner of preparing course material within the proposed computer-communications system is very similar to that presently employed by an instructor. The difference lies only in the means that he would employ. Instead of pencil, paper and typewriter, he would employ a light pen, the screen of a display unit, and a keyboard. He could similarly enter linguistic script of drawings, a topic presently under separate investigation at the Moore School. In operation, the instructor would first determine the ideas which he wants to convey. Then he would proceed with creating a broad outline for a class or study session. Finally, he would move to a more detailed level, specifying the diagrams, slides, reference material, and script that would be required. If appropriate, the instructor might then perfect further the material through the addition of audio and animated movie portions appropriately inserted.

The animated movie technique is particularly valuable where movement of parts of a figure further illustrate a principle, and where the animation helps to emphasize dynamic changes. Examples for use of animated movies are three programs under development at the Moore School to illustrate electromagnetic wave phenomena, projective geometry, and graph theory, Wolfberg (1969). This mode of preparing instructional material is illustrated in Fig. 3 showing an instructor utilizing a display-keyboard-light-pen-microphone console and interacting with the computer system. The process of input described above is very similar to computer programming and a similar technique, Simmons (1968), will be employed to capture and automatically document the instructor's planning of teaching material. Since the information flowing into the computer is in digital form, except for the audio portions, it can be readily processed by the computer which may also aid the instructor through a suggested protocol for various functions. For instance, Fig. 3 indicates instructor-computer collaboration in the indexing, drawing, animation, and sound coordination functions. When the material is considered complete, even in a temporary sense, it will be filed in the repository with appropriate entries being made in the catalog and the directories to account for the new material that has been added. This material will continue thereafter to be available to the instructor for additional modification and perfection.

Since the dissemination of the information to students requires considerably more communications channel capacity for movie and sound than for display of slides and sketches, it would be valuable if both modes could be retained, with the sketches and slides constituting a subset of the animated and sound portions. Then the students and instructors can make an appropriate selection when the material is being taught.

Video tapes are another important and widely used source of instructional programs. With present technology, this medium presents problems of higher costs because of need for the relatively larger channel capacity for transmission, and because of the video tape represents low storage efficiency. On the other hand, the preparation of video tape can be carried out independently of the computer system and its input does not have to be on-line with the computer during the preparation, but can be postponed until a convenient time. The process of involving video in a computer-communication based system is diagrammed in Fig. 4.

4. PROGRESS TO DATE AND PLANS

The arrangement of procedures here is intended to provide a basis for evaluation of the computer-communication instructional system applied to a specific undergraduate course and thereby to form a foundation upon which the technique can be expanded to the entire curriculum and also adapted for use at other schools.

The initial steps of the development of the computer-communication hardware complex (RCA Spectra 70/46 time-sharing system) and collection of descriptive material on basic pervasive concepts have been implemented. The collection of concepts material has been accomplished over

the past four years, resulting in the establishment of a new electrical engineering concepts course for freshman students taught by two members of the project team. This is a one-semester course which meets three hours per week for 15 weeks and was offered for the first time in the fall term 1969. The course serves as an introduction to electrical engineering concepts including decision-making, optimization, modeling, prediction, energy coupling, amplification, feedback stability, and the implementation of logical thought. Orientation of the course material is toward discovering that these technical concepts are fundamental to many situations arising in our daily lives. For example, the students learn that the feedback concept can explain interactions among individuals as well as between outputs and inputs of electrical systems.

This fall's (1970) electrical engineering concepts course will be split randomly into two or three sections of about 15 students each. One of these sections will constitute the test group for the proposed computer-communications educational system; the other section(s) will be taught in the usual lecture-recitation manner.

In preparation for teaching the test group, the project team will spend the three summer months preceding the fall semester reorganizing the concepts material for input of the computer. Since it is anticipated that with a lead time of only three months only a few of the concepts can be processed in this manner, development of the material will continue throughout the following year in preparation for a second presentation to a new test group in fall 1971. The lessons learned from the moderate presentation to the first test group will be used to modify the continuing development of the system. In the fall term 1971 another test group out of the entering freshman class will then be subjected to an electrical engineering concepts course completely within the proposed computer-communication system.

REFERENCES

1. BORDOGNA, J. and A.L. ARLAS, 1970, An Undergraduate Computer Engineering Program, Engineering Education, vol. 60, pp. 393-394.
2. COSINE Committee Report, U.S.A. National Academy of Engineering, Commission on Engineering Education
   a. Princeton COSINE Conference Report dated 28-29 March 1967.
   b. Interim COSINE Report dated September 1967.
   c. 'Some Specifications for a Computer-Oriented First Course in Electrical Engineering', September 1968.
   d. 'An Undergraduate Electrical Engineering Course on Computer Organization', October 1968.
   e. 'Proceedings of Meeting on Computer Science in Electrical Engineering Education', Stanford University, 24-25 October 1968.
   f. 'Some Specifications for an Undergraduate Course in Digital Subsystems', November 1968.
3. ECCP, 1969, The Man-Made World (McGraw-Hill Book Co., New York).
4. HSIAO, D.K., November 1968, Access Control in an On-Line File System Working Papers of FILE 68 – International Seminar on File Organization by IFIP (IAG) Administrative Data Processing Group, Copenhagen, Denmark.
5. LITOFSKY, B., 1969, Utility of Automatic Classification Systems for Information Storage and Retrieval, Ph.D. Dissertation; The Moore School of Electrical Engineering, University of Pennsylvania.
6. McEOWEN, J., 1969, PAL: A Design for a Personal Automated Library, Ph.D. Dissertation, The Moore School of Electrical Engineering, University of Pennsylvania.
7. PRYWES, N.S., June 1964, Browsing in an Automated Library through Remote Consoles, Symposium on Computer Augmentation of Human Reasoning, Washington, D.C.
8. PRYWES, N.S., 1970, Information Storage and Retrieval: Extending Human Memory and Recall. In: Rose, J., ed., Survey of Cybernetics, Iliffe Books of Great Britain.
9. RUBINOFF, M., 1968, Education in Computer Engineering, Journal of Engineering Education, vol. 58, pp. 931-932.
10. SIMMONS, D.B., 1968, Automatic Flowcharting of Computer Programs, Ph.D. Dissertation, The Moore School of Electrical Engineering, University of Pennsylvania.
11. WOLFBERG, M.S., 1969, An Interactive Graph Theory System, Ph.D. Dissertation, The Moore School of Electrical Engineering, University of Pennsylvania.

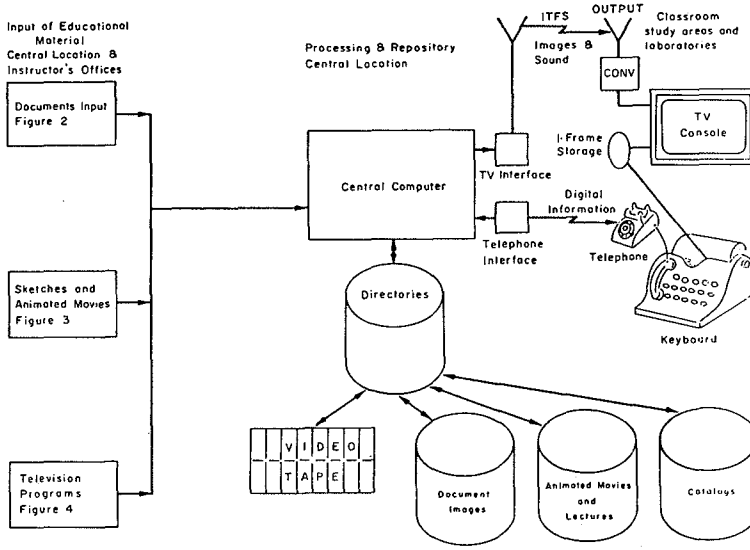


Figure 1 Overall Information Flow Diagram

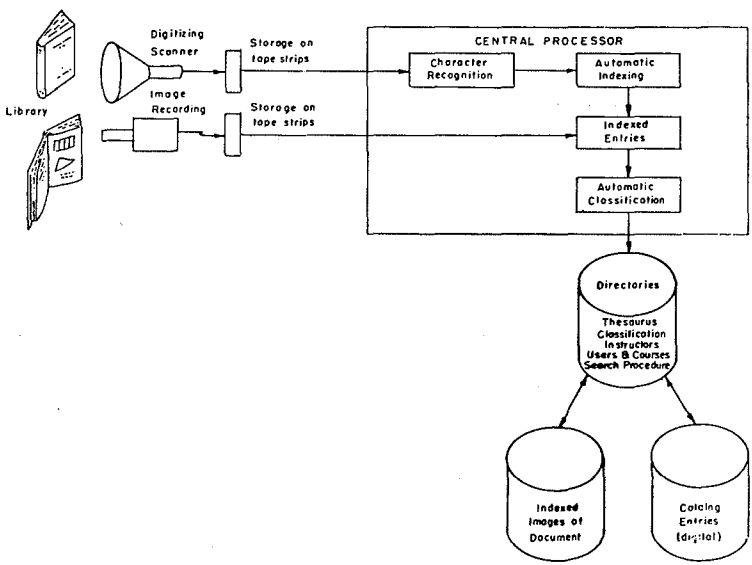


Figure 2 Document Input Flow Diagram

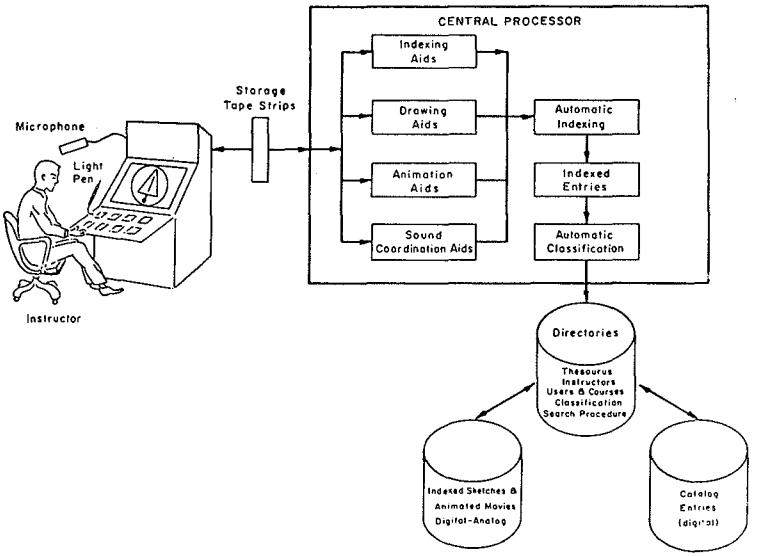


Figure 3 Sketches and Animated Movies Input Flow Diagram

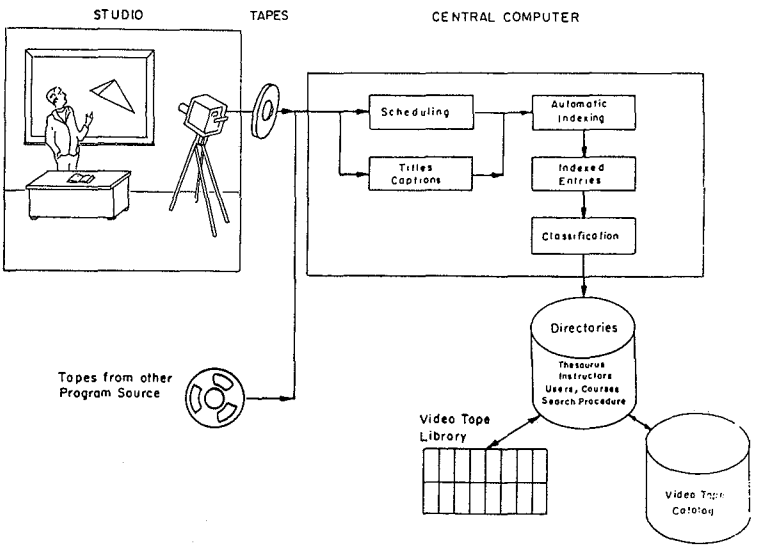


Figure 4 Video Programming Flow

ESTABLISHMENT OF TECHNICAL ENSEMBLES FOR SEQUENCE DESIGN AND PRODUCTION

John Francis Hart, Alan Gray Slemon and Ian Peter Suttie

1. INTRODUCTION

The purpose of this paper is to present some principles that the CAI group at the University of Western Ontario has been considering as guides for its future CAI curriculum projects. Some of the work in CAI that has been done at Western and some illustrative studies from the literature are cited in order to point out the directions that curriculum developments appear to be taking. Although a great deal of effort in the CAI field is being invested in developing software such as author languages, the focus of this paper is on CAI courses.

The involvement of the University of Western Ontario with CAI dates from 1967 when discussions were held with Dr. Suppes' group at Stanford. Every effort has been made to maintain an interdisciplinary operation, and there have been participants from Althouse College of Education, the Department of Computer Science, the London Teacher's College and the Department of Psychology. A PDP-10 computer was the central unit of a time-share system suitable for CAI. The Computer Science Department modified the Ontario Institute for Studies in Education author language, CAN, to make it suitable for our computer, to imprve its performance, and to add some special features. Starting in the fall of 1969, research and development in CAI was begun, but, due to budget limitations, the amount of work has not been great. Most of the work has been undertaken in the Department of Computer Science, however, beginning in the fall of 1970, Althouse College of Education intends to set aside some terminal time for CAI projects.

2. EVALUATION OF SELECTED CAI PROJECTS

2.1 *Tutorial programs*

Most CAI programs, as far as can be judged from descriptions in the literature, are of a fairly standard, branching variety. They can be illustrated by a brief trigonometry program developed at the University of Western Ontario (Suttle, 1969). It consists of 8 one-hour lessons. Lessons are made up of a series of frames, each one of which poses a question to the student. If the student makes an error he is branched to a remedial routine. At the end of each lesson homework assignments are given based upon the student's performance. This type of teaching is sometimes referred to as a „tutorial" system. While it is true that there is only one student working at a terminal, the response repertoire of the „tutor" is very restricted since the programmer has to predict the range of correct and incorrect student responses that might occur and prepare remedial routines for errors. Anticipating the total range of possible student response is impossible, but in spite of the limitations, the prospects for providing immediate, individual positive reinforcement in the case of correct responses, and some degree of corrective feedback in the case of incorrect responses, is outstanding. Certainly a greater degree of individual response can be provided than could be reasonably expected from a busy classroom teacher with more than 30 students. Drill and practice routines, e.g. Suppes' (Suppes and Morningstar, 1969) arithmetic drills are an excellent example of this limited, but extremely valuable form of immediate reaction to student response.

Although in one sense this type of CAI curriculum is individualized, in a more fundamental sense it can be seen that the degree of individualization or „student-centredness" is very limited. This limitation is apparent when one examines how a quality CAI curriculum is prepared. The programmer does a careful task analysis, defining desired terminal behaviour objectively. He carefully defines the entering behaviour of the population for whom the program is being prepared, and he then writes a program working carefully from lower-order concepts and subprinciples through to the principles that are the terminal goals. The student progresses through the program and his comprehension of the various components of the ultimate skills or principles is continually tested. This procedure is sometimes referred to as individualized and as adjusting itself to students, but it should be noted that this is only true to a limited degree. All students are exposed to the same concepts and principles in the same sequence. They have no influence on how the material is ordered or on how the frames are stated. The same limitations are imposed on the classroom teacher who selects the CAI package. Essentially the basic design of the course is in the hands of the program writer.

2.2 *Multiple-level programs*

Multiple-level programs have somewhat more power to adapt to student ability. An example is a program designed to teach introductory FORTRAN (Arnett, 1969). Three versions of each CAI frame were prepared, varying from a simple, fairly concrete version to a more difficult, abstract version. Indices of difficulty level were derived for each of the three versions based upon the successes of students taking that version of the frame. In addition, a value was calculated for each student which continually changed as a function of the student's success during the course of the program.

Prior to each frame a branching decision was made about which level to present to the student using a decision rule recommended by Smallwood (1962) that incorporated the

index of student success up to that point, and the indices of level difficulty associated with the 3 versions of the upcoming frame. This approach resembles the multiple level Drill and Practice arithmetic exercises developed by Suppes.

Adams et al. (1968) report a language program to teach German where the classroom teacher can determine whether students are given more material within a particular unit of a program or even, in some cases, skip entire units by adjusting certain criteria of student performance. In this latter procedure, the classroom teacher is given some control over the course that the program takes, however the control is still minimal. Basically the general framework of concepts, principles, practice exercises, and their order of presentation are determined by the program writer.

2.3 *Increasing student control*

A number of attempts have been made to increase the degree to which students can make decisions about whether they need to do certain parts of CAI programs. An example is a course intended to teach a simplified machine language for programming (Newkirk, 1969). In this course, the student is given a passage of text, and in order to go on to the next passage, he must demonstrate comprehension by answering questions about the content of the passage. However, within this broad framework he has considerable leeway in requesting or rejecting extra explanations and extra practice problems. Hansen (1969) reports that college students, when they are given the opportunity, are quite capable of making decisions about whether they require additional instructional materials that would remedy weaknesses of knowledge and performance. Hansen's observation is in keeping with the results of Newkirk's work.

Perhaps the most ambitious attempt to write a CAI program that will allow student control is that of Grubb (1968). He reports a project to develop a CAI program which gives the student an overview of a course and then lets him enter it at whatever point the student selects. At all times the student is free to move ahead, backtrack, or continue working at the same point. Like Newkirk's approach, the programmer decides what concepts, principles and practice exercises will be included in the program, and how they will be phrased, however the student has considerable choice about the sequence in which he will take the program. Grubb's work is very interesting, however the program required is very long and complex.

An article by Mager (1961) has had a very great influence on our views. He demonstrated that students, when they are given the opportunity to do so, choose to begin a course of study with very different topics than curriculum experts. Mager's work raises the possibility of involving students in curriculum development, even in the initial stages of planning topic sequences. This possibility will be developed in greater detail later in the paper.

2.4 *Research and implementation*

Turning from the organization of CAI curricula, it is worthwhile discussing some of the research in this area. If CAI follows a similar path to PI (Programmed Instruction), we can expect that much of the research will focus on rather narrow, theoretical problems and use brief programs that require, at most, a few hours of terminal time. It is important to study problems of how best to present stimulus materials and how to handle informational feedback to the student. It is also important to examine how a relatively unknown medium such as CAI will motivate students. However, in the final analysis, these sorts of questions can only be answered in actual classroom settings where CAI is integrated as part of the total teaching situation. The novelty effect of a short-term exposure to CAI might give quite different results than would be found with an extended use of CAI.

It can be predicted that there will be numbers of studies comparing the outcomes of CAI with other instructional modes. Undoubtedly, lectures, texts, linear programs and other media will be used as comparisons. While this type of study has some limited value in demonstrating that a new method is no less effective than present methods, it has little other value. In point of fact, when CAI comes into common use, it will probably be used to achieve objectives for which it is particularly well suited. If this assumption is correct, much of the heat generated by questions such as, „Does CAI do a better job than the classroom teacher? " is dissipated, and it becomes clear that whether CAI is more or less effective than some other media is a function of the task and of the students. Adams (1966, p.4) essentially makes this point when he argues that three fundamental questions about instructional method must be answered whenever a decision is made to implement CAI;

What parts of a subject can be efficiently taught?

What instructional techniques would be effective and economic?

What administrative restructuring of traditional course work is necessary?

Ideally, then, development of a course would involve a teacher working out how he could best combine various media in order to accomplish his objectives. In addition, problems of cost and convenience would have to be considered.

Criteria of effectiveness of PI and of CAI program frames have tended to emphasize such measures of achievement as time spent learning the program, amount of immediate and delayed retention, error rate while working on the program, and amount of transfer. In some cases there is an investigation of the student's attitude toward the CAI program. In PI, frames were revised if the error rate becomes too great. Here, again, the programmer makes the decision about whether changes should be made, and the student has only an indirect influence based on his error rate. It might be useful if students could have a more direct influence in frame modification by indicating their opinion of each unit of the program and recommending changes.

3. DISCUSSION AND IMPLICATIONS FOR FURTHER DEVELOPMENT

The preceding observations concern the degree to which much of the development of CAI curricula is technology or expert oriented. Clearly, limitations will be placed upon any CAI programs that are a function of limitations of computer hardware and software systems. However, even when allowances are made for the technical medium, most CAI curricula involve an organization of subject matter that seems logical from the viewpoint of the curriculum expert. Similarly, recommendations for organizing PI or CAI curricula emphasize definition of behavioral objectives and sequencing of topics starting from basic concepts and progressing toward ultimate principles. Most recommen-

dations emphasize starting at the level of the student's entering behaviour, they do not emphasize starting with topics of most interest to the student or sequencing topics in a manner that is meaningful to him. In most disciplines, there is some leeway about where one might begin a course, and starting at a point of maximal interest to students seems like a reasonable consideration.

An outcome of this focus on technology and expert design is the implicit tendency to view students and teachers as passive objects in the CAI program. They are recognized to be the ultimate consumers of the CAI product, however there appears to be a tendency to approach them in somewhat the same way as a physical scientist would study inert or living matter in isolation. They are variables with certain idiosyncrasies to which the program must adapt. The CAI group at Western asked what changes in procedure might be called for if the focus of attention was shifted to the classroom teacher and to the student. If classroom teachers and students become active participants in the development of CAI programs, we may have to turn to the work of innovators such as Mager in order to find procedures for determining the most effective sequences of subject matter from the viewpoint of the student.

3.1 *Variety of student reaction*

One additional problem related to student attitude that has not been documented might be mentioned. There is a great variety of reactions to CAI, and it is quite possible that attempts to develop CAI curricula in different subject matter areas might have to be handled in quite different ways, not just because of differences in the subject itself but because of personality differences in learners in the different subject areas. Any of you who have tried to explain the fascinations of computers to a colleague in humanities can appreciate that an approach that seems natural and reasonable to a graduate student in physics might be quite offensive to an English student who intends to teach in an Elementary school. If this observation is correct, it indicates that the reactions of different types of students, who are the ultimate users, should be very carefully considered.

3.2 *Student participation in materials preparation*

The CAI group at Western felt that we should broaden our approach in order to try and use the computer to facilitate learning of expressive skills. As indicated in the preceding section, models of CAI have tended to conceive the student's role as sitting before a terminal working on a pre-arranged program. We would like to suggest that this use of the computer only partially exploits the great potential of the system. Students should also get direct experience preparing material for presentation by computers. We feel that this additional use of the computer would have a variety of positive consequences. Some of the feeling of helpless resentment towards machines is overcome when one is in the position of being a programmer. Attempting to communicate knowledge forces one to learn concepts and principles in precise detail. Of course, such desirable outcomes would require a favourable environment and careful planning.

Mager (1961) found in his study that if students had genuine control over the learning situation, they would structure the course quite differently than would a curriculum designer. In the subject area used by Mager, all students independently began by asking questions about the same topic in the course. Students appeared to be highly motivated. At the beginning of each session a number of students recapitulated previously learned material to make sure it was correct, but they did not want a review by the instructor. This type of study indicates that the ultimate user of the program, the student, should be much more directly and actively involved in design of learning materials. It seems that the design of curricula has been more controlled by the logical organization of the subject matter than by a psychological organization. If students were more deeply involved in design of material, initial entry into programs and program organization might be influenced very much more by what is of interest to the student.

If planned carefully, a number of educational objectives might be achieved. Traditionally, students have been required to write essays in order to develop skills in communication. Similarly, science reports are intended to develop communication skills. However, the feedback from these assignments is limited.
Usually the teacher is the only one who reads the papers, and he gives a general assessment. If a student prepared materials for computer presentation to other students, he would have the advantage of learning how a variety of other people react to his productions. In addition, with CAI procedures it is possible to obtain detailed feedback about reactions to the component parts of a project as well as an overall, general assessment. This sort of detailed feedback from a variety of other people can be immensely helpful in learning about one's strengths and weaknesses in communication skills as authorities in programmed instruction attest.

3.3 *Students as teachers*

It has frequently been observed that we learn a subject much better when we teach it. Not only do we master details of concepts and principles better, but also we are forced to work out the way they are organized into hierarchies from subordinate through higher order levels. If the experience is well-designed, the students should come to understand such basic principles of learning and communication as stating behavioral objectives, using optimal balances of abstractions and examples, and selecting content that appeals to the audience. One benefit of examining factors that facilitate the process of learning might be that students would apply these factors more effectively in their own learning. Another benefit is that in this situation the teacher becomes a consultant helping students with their projects. Teachers could become active participants in projects rather than directors. Improved communication between teachers and students and greater insight into the teacher's role are desirable educational objectives that could be achieved in this learning situation.

One other advantage of placing students in the position of communicating material to other students might be mentioned. Lindsley (1970), who has been deeply involved with training children to chart certain clearly-defined responses that they make, points out that, even at the elementary school level, bright children often are much more effective at explaining charting procedures than adults. For one thing, they spontaneously use vocabulary and sentence structures that are maximally meaninful to an audience of children. Adults are often inhibited by the necessity of translating their technical and semi-technical vocabulary. If students are directly involved in preparation of material there may be a great gain in simplicity and

naturalness of expression.

When students are given considerable freedom in selecting projects of interest to them and consultation and facilities for completing these projects, a remarkable variety of materials is produced. The quality of these projects varies from excellent through mediocre to very poor. Even poor productions may serve a valuable purpose if students can profit from the experience of preparing and testing them. Some projects will be of sufficient quality that they can be used as instructional resources in their own right, if they are developed with adequate professional-technical support. The major objectives would be the development of greater knowledge and expressive skills in students, however, a very important bonus could be the production of valuable instructional materials that could be used by other students. Production of products that are valued and useful to other people is a very rewarding experience.

3.4 *Organization and resources*

In order to accomplish the above objectives an adquate organization would have to be developed involving teachers, curriculum consultants, large numbers of students, consultants in learning and instructional theory, technical personnel and appropriate software and hardware. Probably the best arrangement would be on a regional basis so that an adequate number of teachers who were anxious to participate could be included. Teachers would have to be trained to work in the situation, probably by participating in the same sort of project-oriented program that they would later use with their students. The ultimate objective would be to expand this educational approach so that a majority of students in the region would be involved. The students, in consultation with their teacher, would decide upon a project that interested them in the subject they were studying.

Availability of consultants and technical assistance to students and to teachers would be particularly important. A factor that would have to be under constant evaluation and development is a mode of communication among the various consultants, the technical specialists and the students and teachers working on specific projects. If this approach is to be succesful, effective and rapid consultation is particularly important. Students would order and structure material and write sequences, and so they would act as initiators and constructors of materials. However, consultation about the exact meaning of concepts and the application of principles might require a subject specialist. Similary, advice on techniques of setting objectives, writing, evaluation, and presentation would be provided by consultants in learning theory, measurement and media. Technical support people such as systems analysts, media operators and secretaries would have to be available to the students, teachers and consultants.

A particularly important part of project planning would be to determine how the computer would interrelate with other media such as seminars, tutorials, books, documents, graphics and audio recordings in order to accomplish the course objectives. Very careful consideration would be given to working out an integration of media so that their strong features would be maximized and their weaknesses would be minimized. During the developmental phase the student and teacher would decide what supporting materials would be required, would begin collecting them and, with the help of consultants and technical specialists, would prepare any other necessary graphics, models and audio materials. It is very important to plan curricula so that there is a selection of media that is optimal with respect to course objectives and to the interests of the students.

Basically, we conceive this type of CAI system working as a research and development laboratory closely tied with a library and recource centre and with consultants and technical help on call. Lines would go to a time-share computer and a range of peripheral devices would be available to students.

3.5 *Student role*

After deciding on a project, a student would research the problem and begin working on teaching material. The project would be developed and tested in the laboratory. In all stages of development the writer would be constantly evaluating and testing his material within his group. During the course, the student would be involved in several roles, and he would be exposed to a variety of content areas and approaches to those areas. His roles would include research in a project area, active developer of material, critic and adviser about others' material, and user of quality material that had been produced by past groups. Initially, students might be encouraged to work on small topic areas. As they improved in skill, imaginative teacher and student groups might expand their objectives. Possibly the group would work on a larger project with individuals or subgroups working on special phases or sections. The problems involved in organizing lower-order concepts and principles into larger patterns of knowledge would have to be dealt with. This form of cooperative organization would add a new dimension as students acted as subjects to determine whether this hierarchical organization of material had been communicated effectively.

4. RESEARCH

Although the present suggestions have tended to emphasize curriculum development there should be a great deal of possibility for research. It seems to our group that advances in knowledge and in practice will be most facilitated if students, teachers and experimenters are encouraged to participate together in research and development projects in an active and creative manner. The use of a computer allows detailed recording and analysis of a variety of responses of individual students while they are in the process of learning. One particularly important set of observations might involve the way in which the research and development team functions. Such data might help indentify some of the critical variables in interdiciplinary cooperation. Experts in technology, on the one hand, and teachers and students on the other, will have to develop methods of working together in a productive manner. When we consider that 150 hours or more of preparation are required for 1 hour of CAI terminal time, it is imperative that ways be found to draw on the immense potential resources of students and classroom teachers for curriculum and project development. Furthermore, research on human learning of meaningful subject matter is going to require the sort of long-time study with a type of control and response measurement that is only possible with modern computer systems.

REFERENCES

ADAMS, E.N., 1966, Computers and Automation, 15, 2-4.

ADAMS, E.N. H.W. MORRISON, and J.M. REDDY, 1968, The Modern Language Journal, 52, 3-16.

ARNETT, B., 1969, A decision structure for teaching FORTRAN IV by Computer Assisted Instruction methods. M.Sc. Thesis, University of Western Ontario.

GRUBB, R.E., 1968, Programmed Learning and Educational Technology, 4, 38-42.

HANSEN, D.N. 1968, Development of a CAI mathematics curriculum. In: HANSEN, D.N., W. DICK, and H.T. LIPPERT, eds., Annual Learning, Florida State University, Jan. 1, 1968 through Dec. 31, 1968, 119-138.

LINDSLEY' O.R., 1970, Personal Communication.

MAGER, R.F., 1961, Psychological Reports, 9 405-413.

NEWKIRK, R., 1969, Evaluation of a Learner Control Strategy in a CAI sequence dealing with the CLIP computer. M.Sc. Thesis, University of Western Ontario.

SMALLWOOD, R., 1962, A Decision Structure for Teaching Machines, (M.I.T. Press), 34-41.

SUPPES, P. and M. MORNINGSTAR, 1969, Science, 166, 343-350.

SUTTIE, I.P., 1969, Design philosophy for CAI sequences. M.Sc. Thesis, University of Western Ontario.

SIMULATED APPRENTICESHIP

Arthur B. Kahn

1. PURPOSE

At the first IFIP World Conference on Computer Education, there is already evidence of a bifurcation of computer education into two distinct areas.

One is education about the computer which emphasizes detailed hardware and software techniques rather than functional utility. The other is computerized education wherein the computer serves merely as a medium of instruction, albeit a powerful medium.

While there is little need to testify to the importance of both these areas, the purpose of this paper is to call attention to a third aspect. of computer education. This third area involves a fusion of the computer as the medium of instruction with the computer as the subject of instruction. The intent is not only to use the computer as a vehicle to advance the student's knowledge of a subject area, but to develop at the same time skills and understandings for effective computer utilization.

This paper considers the computer from two viewpoints: first, as a force with great impact upon our environment; and second, as a valuable tool which can help our educational system prepare students for the rapid changes in the environment. In Section 2, these viewpoints are amplified to show the need for simultaneous use of the computer as both a means and an end of education. We consider methodologies to meet this need in Section 3. A comparison of the older form of apprenticeship education with current modes of formal education leads to the conclusion that the two approaches should be used to complement each other.

Simulated Apprenticeship is proposed as a means to combine the advantages of on-the-job learning with formal or semi-formal education and as a means to integrate various disciplines. This is accomplished by extracting a set of progressively sophisticated apprenticeship experiences from a productive setting for simulation in a classroom.

Section 4 presents the general methods used to construct Simulated Apprenticeship courses. A specific example is presented in Section 5 which also discusses experiences with various types of classes which included medical doctors, managers, practicing engineers, city aldermen, and teachers as well as liberal arts and engineering undergraduates.

2. THE IMPACT OF THE COMPUTER

2.1 *Computer Appreciation*

The developments reported are extensions of 'An Appreciation of Computer Appreciation' Kahn (1967) which proposed that computer scientists must develop an appreciation of just what Computer Appreciation might be. That paper implicitly defined Computer Appreciation by reporting the development and implementation of a specific pedagogical approach for engineers and managers of a large industrial organization.

Since this is only the first IFIP World Conference on Computer Education, a definitive definition is premature. Following is a tentative definition for purposes of discussion.

> Computer Appreciation: That understanding of the computer's implications which is necessary for an individual to intelligently contribute to policy making on computer utilization.

This definition is broad enough to encompass the wide range of viewpoints expressed in a panel discussion at the 1967 ACM Conference and reported by Organick (1968).

Before presenting the extensions to the previous paper, we must review some of its salient points.

2.2 *The Age of Specialization*

A primary purpose of education is to prepare the student to cope with his environment. To remain relevant, the educational system must track the ever changing environment. 'An Appreciation of Computer Appreciation' noted that an important change in the environment strongly associated with the advent of the computer is the growth of interdisciplinary projects.

Before this change, we relied almost entirely on handbooks for the transmitting of information. While the handbooks were efficient, since it was cheaper to retrieve than to re-analyze, the effect was to fracture our technology into isolated islands of specialization. The specialization which hastened the progress of the Industrial Revolution also created segmentation into fragments.

This was vividly brought home in World War II when we recognized that a complicated system, such as an airplane, could not be designed by specialists who did not communicate with one another. Here is where the handbooks became an obstacle. The feasibility of handbook construction depends upon use of special cases and simplifying assumptions. However, simplifications adequate for the discipline creating the handbook frequently prove totally inadequate for integration with another discipline.

The need for interdisciplinary integration stems from the fact that every solution to a problem creates some new problem. When Henry Ford invented the automobile he could not know that he had also invented a major air pollution problem. One hundred, or even fifty years ago, it was acceptable to ignore the new problem for the sake of accelerated progress, but with the increased pace of population and technology we can no longer affort the luxury of ignoring the new problems created by partial solutions.

2.3 *Role of the Computer*

Fortunately, the modern computer permits us to recompute from fundamentals rather than retrieve simplified solutions. In the process we obviate the restrictions of the handbooks and gain the generality and flexibility required for interdisciplinary interfaces. The increasing urgency of such problems as environmental pollution, urban strife, and the world-wide population explosion make it clear that man's survival may hinge upon effective use of the computer for interdisciplinary problem solving.

It would be ironic if man's survival depends on our ability to computerize bigger and better solutions. Already we have large masses of militants in the United States who feel that a non-militant voice cannot be heard above the clatter of the ponderously mechanical government. The computer poses both a promise and a threat to our democratic institutions. Since the computer makes it possible to consider the needs and aspirations of each and every citizen, we can envision the most effective democratic society the world has ever known. At the same time the computer threatens investigation of the private life of each citizen in much more comprehensive detail than was available to any tyrant in history.

2.5 *The policymaker and the Computer*

In a democratic society the authority to direct our course between such extremes rests with duly authorized officials of government and private industry, the policy-makers. Most of our current policy-makers completed their education 25 years B.C. (before computers). They are generally unfamiliar with the computer and afraid to become involved. The result is a serious danger that the policy-making prerogatives will be defaulted to technologists.

It is urgent that we provide our policy-makers with sufficient computer appreciation to permit them to exercise their prerogatives intelligently. We must provide short courses, seminars, and even private tutoring to our current policy-makers.

Just as important are the future policy-makers presently enrolled in our universities and colleges. We must start incorporating Computer Appreciation into the curricula immediately if we are to have properly prepared policy-makers by 1995. We cannot limit ourselves to future heads of government and industry, even if we could predict who they will be, since policy-making does not stop on this lofty level. The ultimate policy-maker is each and every citizen when he casts his ballot or votes with his consumer dollar. Computer Appreciation will be a prerequisite for every intelligent layman who wishes to maintain some level of control over his life.

3. EDUCATIONAL CHALLENGE

3.1 *Policy-Maker's Needs*

The current and future policy makers urgently need an understanding of the role of the computer in the interdisciplinary problem-solving process. At present they are in the position of the conductor who is desperately trying to coordinate the sections of an orchestra without any real understanding of the conductor's purpose and methods. A two-day quickie course in FORTRAN is little more than teaching him how to wave a baton. He requires more than that. He must understand that a computer application, just like symphony, has a progressive evolution with a start, a build-up and a climax. Even more than this, our policy-maker is not really conducting a performance, but rather he is providing direction for an effort in composition and arrangement.

3.2 *Apprenticeship vs. Formal Education*

There are serious questions as to the adequacy of our formal educational system in teaching the methods of progressive interdisciplinary problem-solving. These questions are of relatively recent origins. In earlier periods professions were learned largely through apprenticeship. In such a system one automatically gains a view of the evolutionary development of a project drawing its support from a wide variety of disciplines into an integral whole.

As recently as 1836, Abraham Lincoln learned his profession by 'reading law' in the office of John T. Stuart. Until the turn of the century, the same was true of medicine, and the medical school graduate was the exception rather than the rule.

In time a body of knowledge developed which could be more readily taught in a more formal setting. Thus, for example, reading and writing and arithmetic became prerequisites for many apprenticeships. With the onset of the Industrial Revolution, it became more efficient to encourage specialization. As the base of factual knowledge exploded, even specialists could no longer be efficiently taught through an apprenticeship system.

Gradually, the formal education system, which is so much more efficient in presenting facts, became more prolonged. Thus, we arrive at a point where high school and college are primarily organized for presenting bits and pieces of knowledge. The student sits in the class, a bell rings, he crosses the hall to another class and another subject, and rarely, if ever, do the twain meet. Dry bones of knowledge are tossed at him and by some miraculous process, with little or no assistance, he is expected to assemble a viable skeleton. As the bones proliferate, it becomes a formidable task even for a modern day Ezekiel.

Our educational system is geared to prepare the student for an environment which for 100 years or more has been tending toward greater specialization. It presents nicely catalogued, polished conclusions. Even when we present derivations, all rough edges of the derivation process are polished away. We impress the student with the antiseptic elegance of Euclid's theorems, when it might be more purposeful to show him the blood which Euclid spilt along the path of his problem-solving.

These deficiencies of our educational system are not new discoveries. John Dewey (1931) pointed them out long ago. If this was true then, it is ever so more true today. There have been numerous attempts to develop methodologies for integrating the curriculum. Our purpose here is to show that the computer provides an ideal vehicle for such an approach. We shall demonstrate this with a methodogy which is based somewhat upon the apprenticeship method of education.

3.3 *The Dilemma*

Apprenticeship education has advantages over formal education in developing problem solving skills. But we cannot return to the system of 100 years ago. Another alternative is to select policy-makers largely from the ranks of experienced programmer analysts. But this would only foster technocracy with computer scientists as high priests.

Even if it were feasible to return to our former system,

apprenticeship is not a pedagogical panacea. It is embedded in a functional environment which produces constant conflict between pedagogical and productive goals. The specific topics covered are at the random whim of the work load. In order to meet schedule pressures, the supervisor must frequently impose a methodology upon the apprentice and inhibit the educational activity of developing a 'new' method. Frequently, the repetitive operational requirements may limit the learning experience to the first year repeated three times with increasing over-polish, rather than providing three years of progressive growth. Finally, the superisor may have little or no pedagogical skill.

In the formal system we have a better opportunity to train the teachers and also to organize efficient presentation of factual knowledge. But there are sharp discontinuities when the student crosses from one classroom to another. In real life the hot pursuit of a solution to a problem does not stop short because the project has crossed the county line into the jurisdiction of another discipline.

We cannot resolve the difficulty by selecting between formal education and apprenticeship. We must recognize that formal education did not replace apprenticeship – it merely delayed it. Even today, many professions, such as accounting and engineering, require several years of experience in a junior position as a prerequisite for professional licensing. Elsewhere, we have the internship of medical doctors, and research assistantships in graduate school.

We must recognize that much of our formal education is preparatory for an apprenticeship which most professionals serve in the first few year of their career, and that it is in the apprenticeship where much of the relevance to real life is developed. This apprenticeship is delayed by the over lengthening of the formal education. It may be that already this delay is too long to provide much needed motivation for growing numbers of students.

3.4 *Resolution*

There is a paradox in the impact of the computer. It increases the urgency of the apprenticeship, and at the same time it adds to the body of factual knowledge which delays the apprenticeship.

If the computer contributes to this paradox, it can also contribute to the resolution. There is no reason why it cannot provide interdisciplinary glue in the educational system as it does in real life.

The computer provides an ideal vehicle for this integration in two respects. First, as a medium of instruction we can construct much more elaborate and realistic case histories and simulations of real life than would otherwise be possible.

Second, as the subject of instruction we can make the student aware of the use of the computer as a tool in interdisciplinary problem-solving. It is the author's contention that if we use the computer in this way, then Computer Appreciation is developed almost as a by-product.

This conference testifies to the growing concern over computer education. The question is will we realize the full potential of the computer in education, or, will computer subjects become another fractured segment patched onto an already overpatched curriculum? And will the computer medium serve merely to accelerate the outpouring of bits and pieces?

4. SIMULATED APPRENTICESHIP

4.1 *Curriculum Development*

Simulated Apprenticeship is an attempt to realize the full educational potential of the computer as a vehicle to blend the motivations of a productive situation, and the integrating effect of apprenticeship with the pedagogical control and efficiency of formal education.

In Section 5 we shall present a typical syllabus and student reactions, but first we shall discuss how the curriculum is developed in three distinct steps: Selection; Scenario Construction; and Extracting the Syllabus.

4.2 *Selection*

Selection involves establishing the scope of the syllabus as well as the choice of a case history scenario. As in any curriculum development project, we start asking, who is the audience, what are the goals, and what resources do we have in time, equipment and materials? From this we construct a list of topics and skills to be developed.

Now we depart from the traditional approach. We 'THINK REAL'. We consider the student as an apprentice in a productive shop. The student is the boss's son-in-law and we have been directed to assign him to that project which will provide the best progressive experience. He will follow this project over the next two or three years in order to gain an insight into its complete evolutionary cycle from concept proposal through: trial design, finished design, implementation, testing, evaluation, redesign, operational use, obsolescence and retirement. But we have better control than the productive supervisor. We can select the project and alter the scenario to minimize the conflict between productive and pedagogical purposes.

The selected project must provide a representative and plausible thread on which to string the various topics and skills to be developed. If all of our topics do not fit the string, we must look for another string. If we cannot find one, we must reexamine the validity of the topic set. In such an apprenticeship, topics are taught because they are required by the productive environment and not because they are in an arbitrary syllabus.

This insistence upon a single string may seem to be a stringent requirement. But Dewey points out that many so called project case histories 'are too trivial to be educative.' He asks for the development of '...problems and projects that come within the scope and capacities of the experience of the learner and which have a sufficiently long span so that they raise new questions, introduce new and related undertakings, and create a demand for fresh knowledge.'

4.3 *Scenario Development*

One we select a project which meets these requirements we develop a scenario by holding a dialogue with a hypothetical apprentice. We imagine what should happen, what might happen, and what we want to happen. This step might be comparable to the development of Programmed Instruction material. However, our concern is macroconcepts rather then microconcepts. For example, we present just enough machine language to provoke the student into inventing symbolic language.

The scenario starts with a statement of the problem and a straight forward solution which is plausible to the novice. The purpose of the solution is not to present the elegance of a finished product, but rather to develop an insight to the construction process. (A by-product of this approach is

that there is little embarrassment if unitended bugs crop up during presentation to the class).

Next, we plan an assignment which will use this formulation as a tool in making a management decision. This assignment is carefully planned under a principle of 'Controlled Frustration'. Hidden deficiencies have been built in to produce frustration on a level low enough to ensure success, but high enough to provoke ideas for improvements. We anticipate suggestions from the student and prepare material to present the new techniques required. We also implement the improved solution with still further built-in deficiencies to be discovered.

The entire case history is developed as a continuous series of such cycles: motivation, techniques, active implementation, and discovery which motivates the next episode. Instead of telling the students about the historical evolution of either Computer Science or our case history, we have him experience it in an accelerated fashion. To create the necessary episodes we draw freely upon the environment of the project and the interactions between various disciplines such as engineering, accounting, marketing, and management.

We build greater sophistication with each episode until we reach a climax with a program which automatically optimizes. This leads to questions of whether the computer can replace the policy-maker. This is turn raises basic questions of goals and values which can be related to the age old quest of religion.

We must take care to remain in the THINK REAL mode as we implement the scenario with a sequence of computer programs. In the process we save any bad runs and blind alleys, and if enough are not encountered we create a few.

4.4 *Extracting the Syllabus*

The completed scenario serves as the basic source material from which we extract a syllabus for actual use with a class. We evaluate each episode in the case history on the basis of our view of the student, his background, and his goals, as well as the class schedule and available resources. Some episodes are eliminated and elsewhere new ones created to fill gaps. We also worry about such details as the clarity of the notation, and the presence of any overly sophisticated programming.

A mode of treatment is then established for each selected episode: brief discussion, detailed discussion and demonstration, or a student exerices. In a short course of 12 to 15 hours it is only possible to have the students actually program one or two episodes. However, three or four major episodes and perhaps 10 or 20 minor incidents can easily be demonstrated and all set in perspective by lecture.

In a full semester course there may be 7 or 8 episodes actually programmed by the students, but there is also a need for many incidents to teach basic skills such as flowcharting, tracing and debugging.

5. EXPERIENCE

5.1 *Scope*

To data the methods described have been used to construct a number of scenarios. The first, (reported in the author's previous paper) developed the classical Bouncing Ball problem into a complete scenario for engineers. The next scenario to be implemented demonstrated that the methodology could be adapted to as different an audience as medical doctors. It involved the use of a computer to monitor a critically ill patient.

Currently, work is starting on a scenario for Real Estate Appraisers which will be based upon access to a time sharing system. Work is also commencing upon a full scale scenario for business managers. However, for the present the needs of this audience are being met by the GRADBLOCKS scenario which has been stripped out of the engineering syllabus.

We shall describe the GRADBLOCKS scenario in some detail. It has been used on several occasions in short courses sponsored by the American Management Association and also for the Madison City Council. Portions of it have been adapted for an undergraduate survey course.

5.2 *GRADBLOCKS, Inc.*

GRADBLOCKS, Inc. is a company which manufactures sets of graduated cubes. Each block has a hole which permits a machinist to stack them on a peg to set tools. The blocks are sold in sets of five ranging from one to two inches in quarter inch increments. Initially, our problem is to determine the costs of the set given the cost rates of: drilling per inch, surfacing per square inch, and material per cubic inch. This problem is formulated with a simple flow chart and an elementary FORTRAN program.

The initial presentation of this material was for the entire first day of a five day short course sponsored by the American Management Association for corporative executives of middle-sized companies. In this instance, part of the challenge was to explore the feasibility of simulating machine interaction in a situation where no machine access was available. In order to accomplish this, we worked in advance with a hypothetical student who produced a sequence of 30 progressive versions of this program.

A major unit of the actual presentation consists of a guided tour through the listings of these programs and their output. The first four runs show an actual sequence of debugging runs with various syntactical, logical, and arithmetric errors. Once the program is debugged the purchasing department reports that we can vary the material cost continuously by changing the alloy. The marketing department recommends that we use the best quality alloy which can be obtained within a price limit of five dollars per set. The next several runs involve a search for the cost rate of material which will yield this target price.

When the class suggests that the limitation to a single trial per run is tedious, we introduce the concept of incremental loops. After two or three runs to debug a runaway loop, marketing requests pricing for different finishes. This leads to a doublenested loop which spews out large volumes of results including details on each block in the set. The students react with a request for condensation, and suggest that since we desire only the total cost we should delete the WRITE statement from the innermost loop. The next run implements this and produces overabbreviated results because the omitted detail includes the identifying information. Another run corrects for this.

Additional runs in the series introduce the concept of arrays and their use for printing out tables, as well as the concept of subroutines. At present this series of thirty runs culminates with a dichotomizing search which automatically determines a material cost rate that would yield the desired price.

In actual class presentation, a certain momentum develops as the students suggest an improvement and the response is to turn the page. When used without machine access this entire series requires two to three hours.

In a six hour day, we spend an introductory hour covering the functional subsystems of computers, machine language programming, and the hierarchy of languages. There is also ample time to conclude with the punch line dealing with the role of the computer and the policy-maker.

For example. we describe how we can use steepest gradient ascent to find the optimum design. We then use this design to produce $100,000 worth of blocks only to discover that we cannot sell them. The motivational research consultant, who is called in, determines that most of the machinists who use the blocks have one of two types of handicaps. One group has poor sight and requires a particular surface finish; the other group has poor coordination and requires a particular sized hole. The computer found us an optimum solution which serves the needs of neither group!

Since it is prohibitive cost-wise to satisfy both needs with either a single design or two separate designs, we are forced to choose between the blind and the lame! In this way we drive home the point that the computer cannot take over the policy-maker's role of setting values, but it can help him do a better job by freeing him from much of the tedious detail.

This material was also used for a short course for the members of the Madison City Council who wanted to be in a better position to evaluate budget requests for computers. In this instance, machine access was available so two of the episodes were used as student exercises with active machine interaction. One member of this group, who claimed to have never had high school algebra, recognized that the total cost was linear with regard to the drilling cost. This incident has led to the development of another episode in the scenario.

It is planned to expand this sequence to about 100 runs. Possible extensions include: collection and analysis of a data base on costs: inventory considerations, capital investment considerations, graphic output, and actual implementation of optimization.

5.2 *Medical Doctors*

Exceptional success in developing participation was achieved in a short course on computers and medicine jointly sponsored by the American College of Physicians and the University of Wisconsin Medical School in October of 1968. This success was largely due to the contribution of Dr. J. L. Simms in selection and development of a problem appropriate to the audience.

A fairly elaborate program was developed representing the logic of a computer monitoring a patient in an intensive care unit. After describing the medical regimen, a flowchart and a FORTRAN program were developed. Then the participants were asked to medically evaluate a simulated performance with a patient. The doctors were asked to identify instances where the computer followed poor medical practice and to recommend modifications to the program. They were also given machine access to actually implement some of these improvements.

5.3 *Undergraduate Students*

In the fall of 1968 the author accepted a visiting faculty position at the University of Wisconsin in order to adapt the methods of Simulated Apprentiseship to undergraduate teaching.

The working drafts of a textbook for undergraduate engineers using the Bouncing Ball Scenario were used for classes in the fall of 1969 and in the spring of 1970. Some of the chapters were not available in time for the first class. Nevertheless, the student reaction was generally favorable and criticism was largely of a constructive nature. The reactions of the second class, now in progress, indicate that the major rough spots have been smoothed and that considerable student involvement and enthusiasm is being generated.

Elements of the Gradblocks Scenario were used in a survey course intended for a broad spectrum of undergraduates, but in this case, the student reaction ranged from unfavorable to indifferent. It appears that in such a course, students are looking for a broad survey and find it tedious to follow a single problem in depth. The approach used by Davidson and Koenig (1967) is perhaps more suitable for such students.

5.4 *High School Teachers*

It is appropriate to conclude by discussing experiences with a graduate seminar for high school teachers. As a term project each teacher was required to develop material which would integrate computers into their existing course which covered mathematics, physical sciences or business. It took considerable effort to get these teachers to discard their preconceived notions of inserting an isolated unit on computers in favor of integrating the computer by applying it to units already in their syllabi. Although success was ultimately achieved, they had difficulty in seeing the curriculum as an integral whole.

An interesting anomaly in the group was the only elementary school teacher. This individual was extremely weak in mathematical and programming skills, but her term project may well be the most significant contribution to come from this group. She developed a set of flowcharts to present forest ecology to sixth graders, and is currently working with the author to develop a companion computer model. Although this is an isolated case, it is perhaps indicative that over the years, since Dewey suggested the need, there has been significant development of integrating methodologies on the primary school level.

There have been also numerous efforts to introduce such methodologies into secondary and higher education. The author is interested in receiving communications which would help to relate Simulated Apprenticeship to other existing methods. But even more important there is a need to examine the entire catalog of integrating methodologies to establish the role of the computer, not only as a medium of instruction, and not only as another subject of instruction, but as a vehicle to facilitate integration.

We must realize this full educational potential of the computer if our policy-makers of 1995 are to have the computer appreciation they require.

6. SUMMARY

A major impact of the computer is to provide a powerful tool for solving the interdisciplinary problems we must solve in order to survive. Our policy-makers must have an appreciation of the computer in this role if they are to make effective use of this resource. Our secondary and higher education systems, although they have many advantages, present knowledge in too highly fractured

fashion to meet this need. The development of skills in synthesizing formal knowledge for problem-solving is more generally acquired in a professional apprenticeship.

To meet our current needs, we cannot return to the old apprenticeship system, but we can insert elements of apprenticeship into the formal curriculum to better integrate the two stages of education. Simulated Apprenticeship provides a pedagogical mechanism directed at this goal.

The computer enhances the approach in two ways. It provides subject content on a powerful real life problem-solving tool and it permits more sophisticated simulation than would otherwise be possible. When we use the potential of the computer to implement integrating methodologies such as Simulated Apprenticeship, we develop a good measure of the Computer Appreciation which is essential for our policy-makers.

REFERENCES

1. DAVIDSON. C.H. and F.C. KOFNIG, 1967, Introduction to Computers and Applied Computing Concepts, (John Wiley and Sons, Inc., New York)
2. DEWEY, J., 1931, The Way Out of Educational Confusion (Harvard University Press)
3. KAHN, A.B. 1967, An Appreciation of Computer Appreciation. In: Proceedings of 22nd National Conference ACM. (Thompson Book Company, Washington, D.C.)
4. ORGANICK, E.I., 1967, Communications of the ACM. Vol. 11. pp. 263-268.

Acknowledgements

The author wishes to acknowledge the contribution made to this project by numerous organizations and staff which have contributed support. Particular mention should be made of the Defense and Space Center of Westinghouse Electric Corporation in Baltimore, Maryland, and the Computer Sciences Department of the University of Wisconsin, Madison, Wisconsin. This work would not have been possible without the material support, cooperative staff and favorable environment provided by both these organizations.

A NEW EDUCATIONAL STRUCTURE FOR THE COMPUTER AGE

B.M. Murphy

1. SCIENTIFIC AND TECHNOLOGICAL PROGRESS

Scientific and technological progress generates a considerable pressure upon national educational systems. Not only quantitively as more people with more knowledge and skill are needed but also qualitatively as the level of intellectual content in the knowledge and skills required is raised. New disciplines have to be introduced, old established boundaries melt away and a continuously open mind has to be maintained. New technologies arise, develop, are exploited and die away in the working lifetime of one man. No longer can education be equated with the absorption and retrieval of a set quantity of knowledge. A continuing capacity to learn must be acquired.

2. THE COMPUTER AGE

Rapid technological progress is exemplified in the development and application of the computer over the past thirty years. In 1939, Couffignal, Zuse and Turing had each developed general theories on computing but their work was little known outside a very small circle of enthusiasts. Between 1941 an 1951 these theories were applied in the first electronic computers, the Z3, the ENIAC and UNIVAC 1. Between 1951 and 1961 came the trasistorised computer, the first compiler, FORTRAN, ALGOL and COBOL and multi-programming. The pace of change is such that it is still too early to list the significant developments of the third decade, 1961-71. However, we do know that in 1959 there were about 550 systems operational in Western Europe, by 1961 the number had increased to 1,600 and at the end of 1969 the number was in the region of 20,000 computer systems. A phenomenal rate of growth against a background of new ideas and a technology of growing complexity.

3. NEED FOR TRAINED PEOPLE

The 20,000 systems in Western Europe probably had some 300,000 trained staff attached directly to them and possibly twice that number were so directly affected by their activities that they needed more than a general knowledge of computers to carry out their work. In addition there were the training staff and the teachers, of science, mathematics and physics in particular, who had produced the computer personnel and training staff. Around a million people approximately had tackled and absorbed a new science, a new subject in some twenty-five years. An impressive figure, but it had strained the European educational system to the limit. Already many of those concerned with the future of computers realise that the time is shortly coming when the growth of the industry will be limited not by shortages of technological or financial resources but rather by a grave shortage of adequately trained personnel. The overall growth of the number of computer installations in Western Europe is likely to be over 20% per annum for the next five years. The educational system as it is currently structured just cannot meet the demands for trained personnel that the growth figure predicates.

3.1 Rising demand for qualified scientists and technologists is not, of course, restricted to the United Kingdom alone, it is a world wide phenomenon. In the United States, for example, the U.S. Bureau of Labour Statistics forecasts a shortfall of some quarter of a million qualified scientists and technologists between 1960-1970, whilst the U.S. Engineering Manpower Commission expects to have to obtain some 400,000 engineering graduates from outside the U.S.A. if demand is to be met between 1965-1975. A considerable proportion of the U.S.A.'s shortfall is likely to be met from Europe, particularly from Britain, whose qualified scientists and engineers will be attracted by similarity of language, higher salaries and better conditions. The serious situation in Europe will therefore be exacerbated.

3.2 Within Europe governments, industry, commerce and academics have all agreed that the key to increased national productivity can be found in the effective, widespread use of computers. Within Britain, France and Germany some action has already been taken to rationalise the native computer industries so that they can meet the demands of the future. Industry and commerce have been stimulated by growing competition to apply computers to an ever increasing degree, the universities are gradually being supplied with the computers they need and the Aigrain Committee plans a European computer network. Seen from the position as it was five to six years ago an enormous amount has been achieved.

3.3 All these gains can be lost over the next decade, however, unless immediate action is taken. However great the investment in computing systems, however considerable the support of governments, Europe cannot benefit if its computer installations are bereft of trained men and women, guided by properly trained managers, a large proportion of whom must come from a computer orientated background.

4. THE SECONDARY EDUCATIONAL SYSTEM

All Western European countries have, over the past 20 years, been faced with having to provide mass education at

the secondary level. Taking the 1969 figures against those for 1955 there is an increase of 40% in enrolments overall. If the figures are broken down between general secondary education and that oriented towards technical/vocational education, however, a considerable and disturbing uneveness is apparent. For example;

Total Enrolments 1965/66 against 1955/56

	%Growth General Secondary	%Growth Technical Secondary
Austria	17	0
Belgium	81	114
Germany	11	-19
Italy	41	141
United Kingdom	34	1
Netherlands	63	44

Table 1.1

4.1 It should be remembered that the distribution of intelligence throughout the community can be shown by a normal distribution curve and that we must be rapidly approaching the point where society has 'worked-out' the vein of intelligence in the upperclass student group.

4.2 There appears to be a loose connection between I.Q., currently our only generally accepted way of quantifying intelligence, and social class.

Job Category	I.Q.	
High Professional - administrative	153	)
Lower Professional - technical and executive	132	) computer personnel
Highly skilled manual and clerical workers	117	)
Skilled manual workers	109	
Semi-skilled	98	
Unskilled	87	
Casual labour	82	

This is, of course, a very broad categorisation but it does point out once again the fact that society has narrowed down the area from which we can recruit. It is up to us to take the necessary measures needed to open it up again.

4.3 The possession of an I.Q. of a certain grade does not guarantee the success or failure of a person in their battle with existence. To be the only son or daughter of a rich, well-educated family is likely to weight the battle in your favour even if you are of only average intelligence. A person born with the same innate intelligence to a large, impoverished family with little interest in anything but the daily fight for survival will be unlikely to be anything other than as unsuccessful as his or her parents. In Western Europe we need to follow and improve a pattern of education similar to that developed in the U.S.S.R. That is one that concentrates on getting a sound higher secondary education to the greatest number of people. In other words, we need to start our educational recruitment for such an education from around, at least, the I.Q. level of 105.

4.4 One step that can be taken towards achieving this aim, and I believe it to be a significant step, is the introduction of computer education into secondary schools. By computer education in this context I mean general computer appreciation, computing as a subject and the computer used as a tool by the pupil to aid him in his other studies.

4.5 This idea, although limited enough, meets with a considerable degree of opposition from many of the older members of the teaching profession. They argue in defence of their prejudice that money is short, that computing is but an extension of mathematics and that the subject will only benefit the cleverer pupils. On the question of money they may well be right but we in Europe must learn to invest now in order to ensure a worthwhile future. Secondly, it is up to those of us with experience of the wide applications of computing to educate the educators

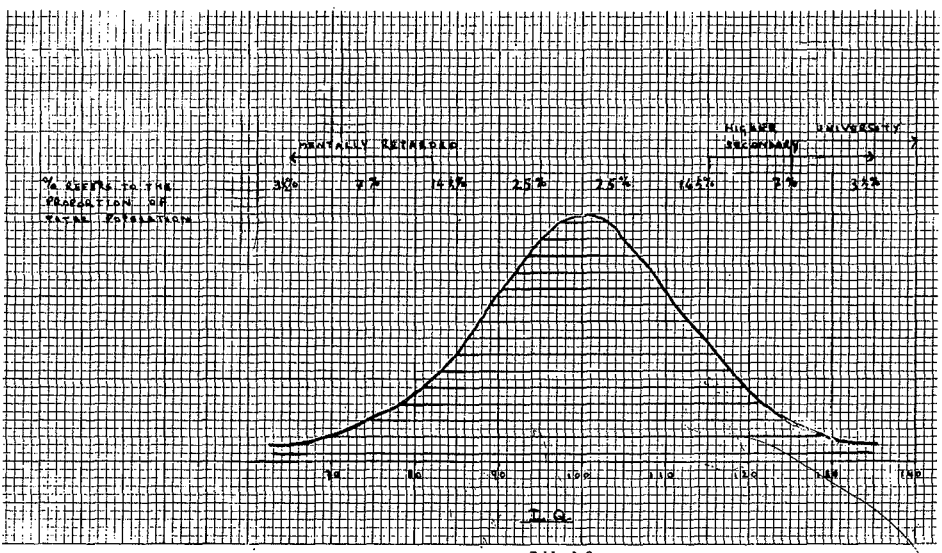


Table 1.2

into a realisation of the truth that those being educated now may well be working until the year 2020. By then certainly, and I believe long before, virtually every task, both intellectual and commercial, will involve the use of computers. Thirdly, experiments in England have shown that children from deprived and disturbed backgrounds are quite capable of learning, albeit slowly, the basic principles of logical thought and the simple use of computers. What is more they not only benefited educationally, their standard of behaviour and social responsibility also improved noticeably.

4.6 We in Europe must see that our children have an understanding of computers, their applications, limitations and their impact on society. We need to replace the fears and prejudices that surround computers with knowledge and understanding based on practical experience.

4.7 A Europe-wide campaign needs to be mounted by the manufacturers, software houses, bureaux and large computer users to ensure that as many schools as possible rapidly introduce into their curriculum, general computer appreciation courses. This is a necessary short term action and would;

- give the maximum number of children, regardless of intellect or background, the minimum knowledge required.
- give teachers an understanding of the principles of computing and allow them to utilise computers in their own fields.

This would allow the subject of computing to develop naturally through practical experience and would stress its wide area of application. Additionally, it would greatly extend the reserve of educated personnel from which the computer industry and computer users could recruit.

4.8 Throughout Europe, various national bodies are attempting to formulate some form of programme for computer education. Generally they are meeting with little success as they lack official support and are made up of part-time volunteers. It is absolutely vital that a full-time organisation be established with official backing. In default of unified pressure from any other area, it may well be that once again computer manufacturers, large users, software houses and bureaux will have to take the initiative. The functions of the required organisation must be

- to carry out a depth study to establish the scale of the problem.
- to recommend detailed solutions.
- to press governments and all appropriate bodies to ensure that the recommendations are carried through.
- to monitor the situation over the years and to act as a clearing-house for information on all aspects of computer education.
- to promote research and development in this area.

Without some central body to direct effort in the field of computer education, Europe continually lags behind the U.S.A. and probably also behind the U.S.S.R. and Japan eventually. The consequences in human and economic terms would be disastrous.

5. HIGHER EDUCATION

The rapid growth of Europe over the past 25 years and its general economic improvement has led to a period of academic affluence. Enrolments to universities and other higher educational establishments have greatly increased but a study of the situation reveals disturbing social factors. There is a need to democratise higher education. We run our economy using an elite chosen from a small minority of the total population. In the U.S.A., more than half the college students are the children of manual workers; in Europe this is true of less than a quarter. Furthermore in the U.S.S.R. 35 per cent of new engineering graduates are women, in Europe only 0.1 per cent. We are obviously ignoring a high percentage of our total intellectual resources. In absolute terms, so a recent O.E.C.D. study tells us, the number of students of working-class background increased threefold in Germany and fourfold in France and Italy between the mid-1950s and the 1960s. But in relative terms the picture looks different, for the number of students from an upper-class background increased by 6:1,000 in Germany, 47:1,000 in France and 20:1,000 in Italy. The figures for working-class students are 1:1,000; 3:1,000 and 1:1,000 respectively.

6. EFFECTIVE MANAGEMENT

If it is to meet international competition, Europe needs a greater output of scientists and technicians and educated management who can apply discoveries in development and production. To achieve this there must be a broader degree of higher education. In the U.S.A. two out of three boys can expect some kind of higher education, in Europe only two out of nine. In Europe many U.S. firms earn higher profits than their European counterparts; a study of this phenomenon indicates that this is, in part, due to the better qualifications of their executives.

Per Cent of Executives with University Degrees	Per Cent Profits/Total Assets
13.5	0 - 24
25 - 49	17.9
50 - 74	19.5
75 and over	23.8

Table 1.3

6.1 In particular, more executives with scientific and engineering qualifications must enter general management.

7. GREATER MOBILITY

Greater mobility is needed between government, industry and the centres of higher education, especially at senior level. One result of planned mobility of this kind could be the emergence in Europe of a centre of excellence such as has grown up on Route 128 outside Boston where some 250 advanced companies have been created by the movement of scientists and engineers out of the Massachusetts Institute of Technology. Work with industry at M.I.T. confers academic status, and the companies thrive on both governmental and commercial markets drawing bright people and ideas from and to M.I.T. According to the U.S. Department of Commerce study on innovation, the total net personal worth of a dozen of these new academic/industrial entrepreneurs is around $500 million.

7.1 It is important, too, that the educational system should be positively formed so as to reflect the new

structure of society. There is no point in laying down an educational system for the future based on present experience which will be overtaken rapidly. We have to see that technology becomes a desirable element of our national cultures and that those concerned with the political, social and economic sciences and the humanities receive an education that is supplemented by an introduction to technological problems and an education in basic science.

7.2 In the same way this process must be reversed and we must broaden the education given to our engineers and scientists. They must see the consequences of their actions upon the whole economic, socio/political structure of society.

8. FUTURE PATTERN

One serious weakness of our present secondary system lies in its uniformity. From our own experience we know that different children and adults respond best to different educational procedures. This being so, equality of educational opportunity requires ideally a system that allows the individual to be taught by the right educational process for that individual. Such a system would obviously be prohibitively expensive but we can, through computer-aided instruction systems, come much closer to it than even before; so today the computer is the key to the new learning. If we are to have a future in the fields of science and technology we must first set up an appropriate educational system that will give everyone a broad appreciation of the computer's potential and a sound idea of how their own lives will be affected by them.

8.1 In order to broaden the outlook of the universities we must make it as academically respectable to work in applied science as it is respectable at present to work in basic science. We must get people out of the universities and into industry. Industry itself must do a great deal to ensure that the people who are working inside it are continuously re-educated in the technological field. To meet the challenge of life today all of us need long re-training courses every two or three years throughout our working lives. If we do not do this we will soon find ourselves out of touch.

8.2 Obviously, if we are going to do this, we have to call upon the teaching profession to provide the training programmes and training. I hope that this will be the first step towards the integration between the two separate classes of society at the moment; the academic and the industrialists we must get them working together if we are to produce managers capable of profiting from the new technologies.

8.3 All these ideas can be carried out purely on a national basis, but economies of scale demand that we think in at least a European context whilst really, international action requires to be taken. After leaving college or university, students should, as far as possible, go and work in an industry or organisation outside their own country, and the fact that they have done this for a year or two should count strongly towards their promotion when they return to their own country.

8.4 I agree with those who say that a European institute of science and technology should be established. The overall objective of such an institute should be the training of leaders for the management of technological research and development and innovation with reference to the total socio/economic European environment. This implies educational and research programmes.

a) Stimulating an interdisciplinary approach to complex problems taking into account the scientific, technical, economic and social aspects of possible solutions.
b) Promoting methods of facilitating the transfer from scientific discovery to technical application.
c) Imparting the ability to forecast future situations and possibilities, and hence to conclude where to intensify and where to abandon scientific and technological projects.
d) Stimulating studies of large-scale enterprises concerned with problems of international importance such as communication, transportation, pollution and so on.

9. CONCLUSION

I have spoken today of some of the problems facing my fellow Europeans in constructing an educational system to meet the demands of the computer age. These are not Europe's problems alone, however; they are repeated to a greater or lesser degree troughout the world. Unless they are seriously considered and solutions can be rapidly found to them, the world will not benefit from the computer age which, instead of fulfulling its great promise, will prove to be one of the most disheartening in world history. It is our responsibility to see that this does not happen.

COMPUTER ASSISTED INSTRUCTION

A. Romano

1. INTRODUCTION

The aim of this report is to present the objectives of C.S.A.T.A. CAI laboratory's research and development programs. The laboratory structure and the results obtained in the first period of activities (May 1969 – May 1970), are described here.

C.S.A.T.A. (Center of Studies and Applications in Advanced Technologies) operates in the Bari University environment and is supported by Italian Agencies for Southern area development as Cassa del Mezzogiorno and FORMEZ, and other agencies and industries interested in promoting research and development activities in the South.

The development of instructional and research programs in the Computer Sciences field is one of the areas in which C.S.A.T.A. operates.

The CAI laboratory is conceived as a function of two aspects closely related to computer use: Computers for Education, and Education about the Computer. In this sense the CAI's activity is interpreted in a broader context as the information system in education. The laboratory makes use of the C.S.A.T.A. equipment including the IBM 360/65, IBM 1130 and IBM 1800 systems (see figure I).

The IBM Scientific Center of Bari collaborates with five researchers in the laboratory activity program.

2. DEVELOPMENT AND DIRECTIONS

The CAI laboratory's research and development activity is closely related to the C.S.A.T.A. activity in the computer science field.

In order to do this it takes, and will take, advantage of the methodologies of all other research groups operating in C.S.A.T.A.: the group developing information storage and retrieval techniques, the group working on numerical and statistical analysis automation, and the group investigating pattern recognition techniques and methods.

The approaches taken by these CAI laboratories take into consideration the three more important aspects of the computer role in education:

a) business management of educational institutions;
b) logical and computational tools used by technical professional people;
c) direct factors in an instructional system.

This justifies the broader context in which we put CAI, because all functions common to other systems, i.e., storing, retrieving, processing, disseminating and creating information, are taken into consideration.

As is well known, point (c) is until now in an experimental phase and its development is connected to some technical progress which must be reached:

- more advanced terminals;
- improved processor in this kind of terminal-oriented application;
- more powerful logical techniques of response processing;
- a better understanding of what are good instructional strategies and programming methods.

Taking into consideration these general aspects and the realization and orientation of several CAI projects in U.S. Centers, we intend to direct the CAI laboratory development in the following operational directions:

i) Computer science and instruction:
   a) training for computer users and computer technical-professional people;
   b) undergraduate degree programs and graduate research programs.
ii) Instruction and use of computers:
   c) convenience computing for students, to develop suitable facilities and prototypes for use in courses:
      - for problem-solving using APL, FORTRAN, PLI;
      - in simulation laboratories using APL;
   d) general student and faculty use of advanced application packages in all schools of the University; by developing a center of expertise for promoting this use and providing consultation services.
iii) Instructional use of the computer as a medium:
   e) development of system programs;
   f) development of a demonstration center of expertise in instructional techniques and course strategies;
   g) development of expertise in computer-managed instruction by means of a prototype course development project;
   h) development of a center for instructional evaluation.

3. PRESENT ACTIVITIES

3.1 *Computer science and instruction*

This activity has been developed for many years through yearly courses for undergraduates and two year courses for graduates concerning the basic logic of data processing systems, programming languages (PLI, FORTRAN, Assembler) and some of the basic methodologies of the Information Science (numerical and statistical analysis, simulation, mathematical programming, methods of information managements).

This educational activity also allows for the development of a training program for people associated with the research program of C.S.A.T.A.. Every year 50 undergraduates and 20 graduates are instructed.

3.2 *Instructional use of computers*

This activity is closely connected to the CAI group which develops conversational systems and applied mathematically oriented languages.

At present the group is studying:

a) a conversational system for experimental data processing;
b) a language for formal computation;
c) a language for mathematical programming;
d) the definition and the carrying out of a new conversational system for automatic numerical analysis and of procedures for formal computation.

We consider the results of this activity as a basic point for using the CAI system at problem-solving and simulation level for University curricula.

3.3 *Instructional use of the computer as a medium*

We focus our attention on the purely technological aspects concerning three principal functions:

a) defining suitable languages for the student-machine interaction and their use in some experimental applications;
b) implementation of more flexible methods of response analysis. This is a complex field to which the laboratory gives the meaning of basic research activity;
c) methodologies for the course management aspect (CMI).

These points have been developed, carrying out some basic applications.

3.3.1 Conversational language for CAI

The student-computer interaction is realized through a language by means of its text-processing, diagnostic and computational functions. The different kinds of languages actually used belong to two main classes in which are prevalent:

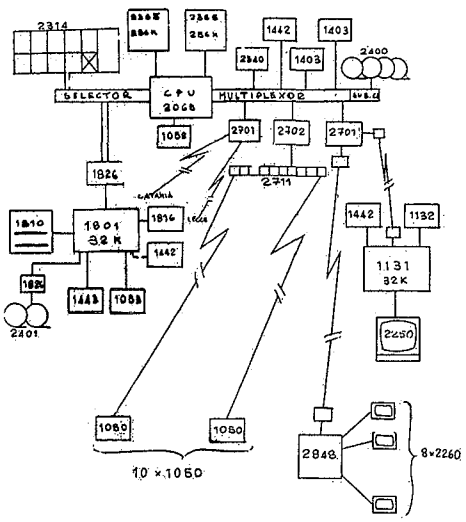


FIGURE 1 - Hardware facilities

- computational performances;
- text processing performances.

At the C.S.A.T.A. laboratory, general objectives not being limited to a particular project, we studied two languages belonging to the two previously described classes (Coursewriter III and APL). These languages were analysed for their internal structure point of view.

In a CAI program in a large shared system we can distinguish between communication locations (CPU and fast memory controlled by an operating system, slow memory, user stations, partitions of the core memory with partition supervisor) and communicators (human student and author, course program, interpreter or compilor, operating system, i/o control programs). In the process of constructing, computing and executing the CAI programs, communication becomes necessary between a number of different pairings among the abovementioned communicators. Each such communication pairing or interface has its characteristic language for communication.

To develop and use the system efficiently it is necessary to understand the requirements for each of these interfaces and how they are achieved in the particular system one is working with.

An effective language reflects, in its choice of characteristics, the nature of each of the two communicators and the kind of messages they have to exchange. We want briefly to discuss the two languages we use for CAI in large shared computer system: the Coursewriter III language running on the IBM 360 system, and the APL language.

Conversational programming systems like APL and Coursewriter involve a system of several programs and languages:

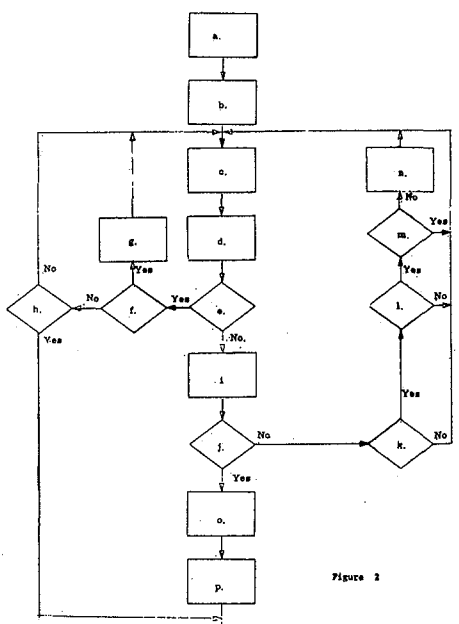


Figure 2

- a partition supervisor with its task control;
- a programming language;
- an author text editing language;
- a command language for controlling program execution.

We can associate the conversational programming system, lik APL or Coursewriter III, with a conceptual machine to wich it is addressed. The APL conceptual machine has:

a) a CPU which can execute the APL programming instructions;
b) a working memory;
c) a slow memory in which reference copies of working memory are kept.

The Coursewriter III conceptual machine has:

a) a CPU consisting of the Coursewriter supervisor and interpreter, record log program, etc.
b) a working memory normally stored on disk.
c) a program memory always stored on disk.

When a user makes a request for service, the working memory is rolled in and the appropriate page of the program memory is copied into core, unless already there.

Control is given to the program page, which executes until the request is serviced or it ends before service is complete. After service is complete, the memory is rolled out to disk.

It is noteworthy that the relation of the APL user to the APL program is very different from that of the Coursewriter III program. The APL user is a source of a program and this program is executed as soon as it is received. The Coursewriter III student is a source of data; the program executed is that of the author. Therefore the Coursewriter III program is most analogous to a program of process control with the student as the process.

At present, we are making applications of these languages in different fields and with different instructional techniques (drill and practice, tutorial, simulation, problem solving). It is evident that APL is vastly more powerful for what is concerned with computational capability. However Coursewriter III has several useful features which are not present in APL:

- the capability to „page" a very long course through a small „key hole" of core.
- a course may control any number of students.
- student record data and performance storing.
- the function feature, which permits the addition of special user's functions to Coursewriter III.

Taking into account the last capability, we have begun to add to the Coursewriter III some functions implied by our experimental experience with the language in different fields such as mathematics, physics, Italian grammar. In particular, our attention is now devoted to adding some computational functions which can be used in various teaching courses in case the student is only required to set the problem.

With the aim to make more dynamic and flexible the dialogue between the student and the computer we are furthermore developing several answer processing functions.

On the basis of the experience with the existing conversational languages, we developed a subroutine package for advanced CAI applications using the IBM 2250 video unit; this unit will be connected to the IBM 360 system via an IBM 1130 computer

3.3.2 Experimental applications

The first experimental applications of the CAI laboratory had the aim to define more suitable instruction strategies depending on the course subjects and levels, and to achieve a deep knowledge of the actually employed languages (Coursewriter III and APL). For this we developed several course segments concerning:

- Arithmetic for the secondary level schools (general properties of arithmetical operations);
- Physics for the first course of the University level (Coursewriter laws);
- Grammatical analysis for the second level schools.

The arithmetical and physics courses make use of the tutorial technique. A grammar course example, using the drill and practice technique, in which the student has to compose Italian words, is reported in appendix A.

According to the C.S.A.T.A. activities in the computer users training, we are now developing courses for teaching programming techniques and languages (Assembler, FORTRAN, PLI, etc.). A CAI course for the FORTRAN language, using both the tutorial and drill and practice modes of operation depending on the student's knowledge, is in an advanced phase of realization.

We are further carrying out some applications at the University level with the simulation and gaming technique, by using the APL language and/or the subroutines package developed by the CSATA laboratory for the interactive use of video terminals.

In order to experiment with the capabilities of the inquiry mode technique in a learning system, we have begun a research activity in the response analysis field.

3.3.3 An approach to the automatic answer analysis

By use of the computer as an aid to instruction we put our attention on the capability of the computer to analyze messages written in natural language in order to recognize the meaning of what is said in a given context and then to take the more appropriate decisions peculiar to a multiple branching technique. It seems to us that an adequate development of this capability will give the computer an exceptionally broad potentiality in the education field.

The usefulness of an educational method in which the computer supervises the teaching functions is tightly connected with the computer's ability to react in an appropriate way to the student behavior, that is to say, it resides in the diagnostic value of the decisions consequent upon the student-answers analysis and in the peculiarity of the corrective action undertaken. In many cases such decisions can be taken in an adequate and proper way basing oneself on mechanical rules, provided the meaning of the answer has been exactly discovered.

Unfortunately, while it is desirable to allow the student to reply in a free enough form, the difficulty of an automatic analysis of the student-message greatly increases when this message is not closely constrained.

The method shown here is limited to the answer processing case rather than to more general problems of message processing. In the former case, in fact, the question provides a certain type of constraint, since the answer must at least contain crucial elements that can be foreseen by the author. Even question-constrained answers will vary greatly when they involve generation of relatively complex sentences; this fact forces us to substitute an analysis on a pure right-wrong basis with a more sophisticated one which points out the degree of correctness of the student answer.

3.3.4 The algorithm

The aim of the study is the definition of an algorithm which, though providing the possibility of reducing, to a very limited number the basic reference forms, allows us at the same time, to process answers whose forms can vary in a fairly wide range.

The research field has been limited to algebra and geometry because the terms of their vocabulary, further because of their being limited in number, they can, in most cases, be considered as having a meaning which does not strictly depend on the context.

Furthermore, in their propositions, some logical structures are detectable and make possible their reduction to symbolic forms. Anyway, generally speaking, the reasonably possible propositions of any theorem, present a structure complexity, a form variety and, for each of them, a possible choice of words such as to make us believe that it is impossible to efficiently follow a method of analysis based only on the check of some key-words presence.

Let's think, for example, of the variety of forms, equivalent to each other, by which it is possible to express the „concept" of „SQUARE".

Anyway, it is possible in those cases in which the number of concepts expressed in a proposition is limited, to change in a natural way from „Key-words" to a „key concepts" analysis. This is the case of the geometric and algebraic propositions.

The proposed algorithm is rather complicated in nature and cannot be described here in detail; however we shall give some outlines, using examples which illustrate how it runs.

The logical operations of the algorithm of interest for our purpose are:

- the determination of the standard reference forms,
- check of the presence of some „key-concepts" and absence of others,
- rearrangement of the student's answer in a symbolic form,
- message to the student to confirm, correct, complete, in whole or in part, his answer.

In this kind of analysis, a main role is played by the „key-concepts" research algorithm.

It makes use of a vocabulary structured by successive inclusion levels and such that each of its elements, which fully describes a concept, defines a section which constitutes the title.

Each section is defined as an union of two classes:

- the class of synonyms defined as those words such as by themselves express the concept associated to the section they belong to;
- the class of definitions, such as when every-one of them is constituted of more elements, each of which expresses a complete concept distinct from the others, contained in its own section and chosen in such a way that their number is minimum.

One of the possible definitions, for example, belonging to the section „SQUARE", has as elements: 'QUADRILATERAL, EQUILATERAL, EQUIANGULAR".

This structure can be repeated for every element of each definition, giving rise, in successive inclusions, to more and more lower levels. That is to say, corresponding to more and more elementary concepts. The process is terminated when it reaches sections for which the class of definitions is empty.

When, for every section, we use the class of definition as empty, this algorithm reduces to a generalized keywords analysis, in the sense that each of them can be substituted from one of its synonyms.

The algorithm, anyway, gives the author the facility to substitute, where he wishes, a research of the mentioned type, a proper key-word research.

Let's observe that the same section can be included in more sections of an upper level. This fact allows us to utilise, in the formulation of other concepts, previously structured and defined sections, so avoiding the listing, at more and more elementary levels, of all of their expression forms.

In the section „RECTANGULAR", for example, we use the section „QUADRILATERAL", previously defined as „SQUARE".

Let's anyway observe that:

1) the possibility of alternatives in the definition of some concept and
2) the possible non-empty intersection, of the section corresponding to two or more elements belonging to the same definition; involve an ambiguity in the assignment of the elements intersection (if they are present in the answer), to one or other section. This fact can cause the missing of recognition of the present and/or completeness of concept research form, notwithstanding the necessary elements for its definition all being present in the answer.

The algorithm provides an optimum criterion for the resolution of such an ambiguity. Furthermore, it verifies the gradual completion of every key-concept and the presence of them in the answer and provides a localized diagnosis of the error (and therefore a specific remedial action). In addition it gives the author the possibility of scoring the student's answer not on pure „right-wrong" but on a „degree of correctness" basis.

3.3.5 An advanced technique for learning system management

The student-computer interaction control and recording represents one of the basic elements for a computer-based learning system. This is one of the CAI aspects connected with instruction management (CMI).

From a technical point of view, these functions have to be realized by taking into due consideration some general solutions:

a) the data must be organized on direct-access supports, easy for updating;
b) their format has to be free; in fact, only in this case it is possible to generalize the procedures to the manifold demands of the different cases;
c) facilities such as abstracting lists, statistics, subject, copies, etc., must be automatically accessible. Further, the updating operations must be executable by means of simple languages;
d) in a complex educational system case, a certain degree of data format independence from a specific computer has to be kept.

In order to have a quick and efficient way of creating and updating archives, the main procedures of the GISS (General Information Store System) carried out by the C.S.A.T.A. research group in the Information Storage and Retrieval field, are just being inserted in the Coursewriter III system.

Under the direct management of the Coursewriter, the GISS system allows other applications in the education field, such as the planning and administration of

educational services (study curricula for each course, programs and course status, school status, etc.).

The system is composed of a set of subroutines, procedures, macro-instructions, access methods automatically available to the users by means of a language which needs no particular knowledge of the computer functions.

This language allows automatic data updating of archives and the reorganization of the stored information on direct access support. The main characteristic of the system is the complete freedom in the data format choice: it is possible to generate archives holding any length of alphabetic information, different size integer number and a logical vector for any set of binary information.

The continue-string data organization is independent of the computer type. In the case of scholastic archives this fact allows one to utilize the simple archives for the usual management and to collect data at a centralized level in order to obtain statistical evaluation in a standardized way. Some routines supply statistical information, conditional lists, archive reorganization and partial copying.

The system allows the optimal use of three access methods:

a) indexed sequential for homogeneous sets;
b) direct access for fast access requests;
c) an original direct-sequential method to maintain grouping of similar sets having at the same time direct access facilities.

We foresee an automatic system to choose the more suitable organizational method with regard to store optimization.

The research key can be defined in a free format with the automatic selection of the randomizing method.

The archives can be entered in:

a) batch-mode for listing, statistics, large updating;
b) teleprocessing-mode to obtain information with fast response time.

The system actually runs on the IBM 360. We are developing an IBM 1130 and 1800 version. As we said previously the peculiar data organization neverthless makes the system computer independent.

We used the PLI and Assembler languages for the management subroutines and FORTRAN and PLI for service subroutines.

APPENDIX A

– Grammar course

In fig. 2 is reported the logic diagram of the first part of a lesson of the grammar course which the student practices in building some composed words starting from two lists of words which contain the first and second parts of the ten words to compose. Among the words to compose are hidden the names invented by Dante for two devils. Upon the fourth mistake the author advises the student. Upon the fifth mistake, a list of uncomposed words is shown and the meaning of them explained with common examples.

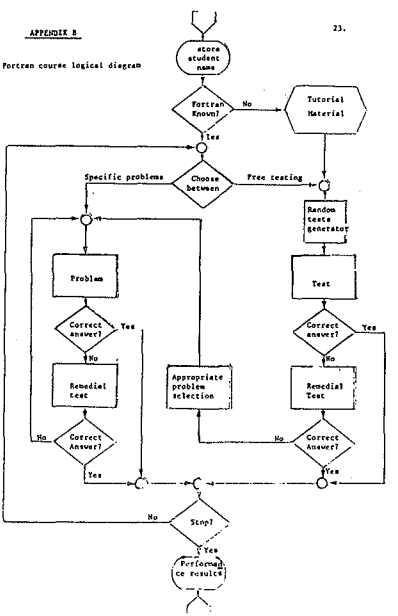
The second part of the lesson consists of having the student split up some composed words, writing every part as in the Italian dictionary (names in the singular, adjectives in the masculine singular, verbs in the infinitive).

The student has to analyze each word in order to perform the grammatical analysis and complete some phrase with one or more chosen words among those just composed.

The first practical experiments with the grammar course, performed on a limited group of students, have awakened a lively interest among them.

The meaning of the different blocks of fig. 2 is reported here:

a. Exercise aims and examples.
b. Lists of words to compose.
c. Request for answer.
d. Analysis of the answer.
e. Is the answer in the list of the correct ones?
f. Was this answer already given?
g. Advisory message to the student.
h. Have all the words been correctly composed?
i. Error message to the student.
j. Is this the fifth error?
k. Is this the fourth error?
l. Did the student give six correct answers?
m. Did the student compose the devils' names?
n. Advisory message to the student.
o. Lists of uncomposed words.
p. Meaning of uncomposed words.



PLANNING FOR THE EFFECTIVE UTILIZATION OF TECHNOLOGY IN EDUCATION SYSTEMS

Morris Rubinoff

1. TECHNOLOGY AND EDUCATION

There is an abiding faith, both among educators and technologists, that technology will somehow revolutionize education. In the 1920's, radio was going to revolutionize education. In the 1930's, films were going to revolutionize education. In the 1960's, it was programmed learning, and now it's computers.

The fact is, we have yet to see a successful marriage between education and technology. This doesn't mean there hasn't been any promise. Technology has been the perpetual bridesmaid. As early as 1841, Josiah Bumstead stated in a book on blackboards:

> "The inventor or introducer of the blackboard deserves to be ranked among the best contributors to learning and science, if not among the greatest benefactors of mankind."[1]

Similar claims have been made for other technologies:

For Films (1940)

> "The motion picture is the most revolutionary instrument introduced into education since the printing press."[2]

For television (1957)

> "It now seems clear, however, that television offers the greatest opportunity for the advancement of education since the introduction of printing by moveable type ."[3]

For programmed learning (1962)

> "...it (programmed learning) is the first major technological innovation in education since the invention of printing."[4]

For computers (1967)

> "The impact of computers on society and, hence on the curriculum has been compared to that of moveable type and the printing press since Gutenberg."[5]

Technology has surely been the tempest in the educational teapot. As we pat ourselves on our backs at this world conference on computer education, let us remember that we have yet to fully realize the potential of the **Blackboard**, and let us reflect a moment on the widening gap between basic thinking and applied action in educational technology.

Suppose there had been a world conference on blackboard education in 1841, and there might well have been. Suppose it had been organized around the recurring theme of this conference:

1. The blackboard as a means of better conducting education.
2. The blackboard as an instrument in education.
3. The blackboard as a subject for study.

The real question is: Can we discuss education and new technology, even blackboards, by focusing on three relatively peripheral uses of technology, without raising and examining the whole philosophical problem of education and technology? The salient problems in education simply do not lie at the technological level of devices and arrangements. The problems are basically philosophical, and the questions are basically questions of purpose and direction. If technologists are to have an impact in education, they must reorient themselves from individual gadgets and material systems to functional goal-oriented systems, with man as the central point of reference.

On the other hand, this does not fully explain why technology has somehow flown around the world of education. Traditionally, the educational institutions have been charged with transmission of our culture. Consequently, most educational practices, politics, and values have been developed on the assumption that stability was more characteristic of the world than change; that change was but a temporary perturbation of stability or a transition to a new and presumed better stable state. Technology undermines this fundamental metaphysical assumption. Consequently, educational practice is not prepared to cope with it.

In summary, then, these are two prerequisites for effective utilization of technology in education:

1. Expansion of technology's vista.
2. Systematic expectation of change.

Both prerequisites must be incorporated into an education planning system.

2. IMPACT ON EDUCATIONAL PLANNING

The implication of the above two concepts on educational planning are profound. It calls for a complete change in the way educational planning is currently being carried out.

The concept of planning has been commonly understood as a method for defining certain objectives and then recommending general and specific policies and actions to guide efforts toward the stated objectives. In the past, many individual efforts have been initiated to deal successfully with the problems confronting education. However, these planning efforts made no provisions for the

systematic expectation of change. Furthermore, planning efforts tended to be highly compartmentalized into fiscal planning, facility planning, programs planning, management planning, staff planning, etc. Technology, if it was at all considered, was relegated to a minor role as part of a materials and equipment planning process.

What is clearly called for is a comprehensive approach to educational planning, rather than a fragmented, piecemeal approach. Comprehensive planning may best be understood in terms of the three sets of responsibilities common to the management of education agencies:

1. **Responsibilities for the Conduct of the Affairs of the Education System**

These responsibilities involve the exercise of guidance, influence, and leadership to steer the system and its component parts toward desirable goals and objectives. Management pursues diverse activities to provide the necessary guidance, influence and leadership. This set of management responsibilities must be planned.

2. **Responsibilities for the Conduct of the Affairs of the Education Agency**

The agency engages in activities directed at specified goals and objectives, identifying work to be accomplished to realize those ends, specifying ways of accomplishing the work, and mustering resources to get the work done. This set of management responsibilities must be planned.

3. **Responsibility for the Conduct of Managements Own Affairs**

Managers perform work presumably directed to explicitly recognized goals and objectives. It is necessary to identify work to be accomplished, specify ways of accomplishing the work and most needed resources (time, money, personnel) to get the work done. This set of management responsibilities must be planned.

Consistent with these sets of responsibilities, Anthony[6] identifies three sets of administrative processes:

1. **Strategic Planning**

Strategic planning is the process of deciding on objectives of the education system, on changes in these objectives, on resources used to attain these objectives, and on the policies that are to govern the acquisition, use, and disposition of these resources. Strategic planning is associated with the first set of management responsibilities.

2. **Management Control**

Management control is the process by which managers assure that resources are obtained and used effectively and efficiently in accomplishment of the agency's objectives as a part of the education system.

3. **Operational Control**

Operational control is the process of assuring that specific tasks are carried off effectively and efficiently.

At this point, it is important to distinguish between "systems" and "processes". In brief, a system facilitates a process; it is the means by which processes occur. A comprehensive planning system then facilitates all three of the processes described above. In such a system, every activity of an organization is compared to every other activity in terms of its relative merits in achieving organizational objectives. Thus, a comprehensive planning system is not concerned so much with planning per se, but with tying together the individual planning efforts which occur throughout an organization and at all levels of an organization in such a manner as to give a common thrust to the work of the many individual units which comprise an education system.

3. DEVELOPING A COMPREHENSIVE EDUCATIONAL PLANNING SYSTEM

While it is possible to talk about comprehensive planning as a means for planning for the effective utilization of technology, the question remains can it be put into practice. As of yet, no organization has been able to develop such a system.

First, comprehensive planning differs markedly from the way planning is traditionally carried out. Planning, as any school boy will tell you, involves three types of calculations:

1. defining a problem;
2. evaluating alternatives; and
3. selecting a solution.

We planners are remarkably bullheaded about this. On the other hand, comprehensive planning, if it is to be comprehensive, involves three different types of calculations:

1. Justification of need for a program without reference to legislative mandates or past practices.
2. Justification of program expenditure levels based on need.
3. Justification of program "costs" from group up.

We planners have yet to recognize the legitimacy of such a process. Furthermore, comprehensive planning goes beyond the traditional concern of planners in at least two ways:

1. It is concerned both with specification of objectives and the selection among programs.
2. It presupposes decisions on these matters and can be aided by systematic analysis, using criteria which are primarily not political.

On each count, it significantly affects that web of relations known as the "politics" of the budgetary process, an area planners have always shied away from.

In closing, we find ourselves in the embarrassing position of the freshman engineer who knows all about why a strapless evening gown is held up, but does not yet quite know how. We know why a comprehensive planning system is a superior approach for planning for the effective utilization of technology in education, but don't quite yet know how to implement such a system. Maybe comprehensive planning represents nothing more then wish fulfillment on the part of educators and technologists. Maybe the incrementalists are telling it the way it is. At least we should find out.

FOOTNOTES

1. Charnel Anderson, "Technology in American Education," 1650-1900, U.S. Government Printing Office, Washington D.C., 1952, p. 18
2. Charles F. Hoban, Jr., "Focus on Learning," American Council on Education, Washington, D.C., 1942, p. 27
3. Alexander J. Stoddard, "Schools for Tomorrow" (an educators blue print), Fund for Advancement of Education, Ford Foundation, New York, 1957, p. 27
4. Dael Woefle, "Prophecy Fulfilled," Science CXXV, No. 3503, 16 February 1962, p. 503
5. John Caffrey and Charles J. Mosmann, "Computers on Campus" (a report to the President on their use and management), American Council on Education, Washington, D. C., 1967, p. 12
6. Robert Anthony, 'Planning and Control Systems" (a framework for analysis), Harvard Press, 1965, p. 13

MODELING INSTRUCTION USING COMPUTER GENERATED DIALOGUE*

Roulette Wm. Smith

1. INTRODUCTION

Much, if not most, of the literature on teaching appears to be aimed at the social and emotional aspects of how teachers behave and how students respond and develop. Few detailed and well designed studies have examined the cognitive effects of teaching. Notable among the exceptions are the efforts by Smith & Meux (1962, 1964), Bellack, et al (1963) and Taba (1963). These studies are descriptive. Their focus tends to be analyses of verbal protocols or units of 'thought'. Gage (1965) notes that...

'Any study of reaching that does not...concern itself.. .with the changes that teaching brings about in students, runs the risk of dealing with irrelevancies.'

Certainly the logical aspects of classroom discourse described by Smith & Meux (1962, 1964) lead to new dimensions of teaching and instruction if their importance were exhibited. The same can be said concerning the categories of meaning in classroom language discussed by Bellack (1963). Moreover, the less cognitively oriented research by Ryans (1960) and Flanders (1965) examined classroom interaction by an observational approach.

A more rewarding result would be the characterization of the variables and the processes of teaching in sufficient detail to postulate strategies of teaching. It is also desirable to establish upper bounds on the quality of teaching that would result from the use of those strategies.

Perhaps the main criticism of previous research efforts is that of the absence of rigorously controlled experimental situations. These are necessary for the models and inferences that are sought. Rigorously controlled experiments are not sufficient. The research paradigm must accomodate a representative class and number of known variables. These variables must be allowed to operate simultaneously. Consider the following example. To research the effects of teacher discourse on student performance the curriculum, classroom environment, teacher discourse and behavioral patterns should be precisely controlled. This author knows of no studies in which all these considerations were simultaneously controlled.

Some of these criticisms may be avoided. One method is to use computer-assisted-instruction(CAI) models in the research on teaching. It will not be sufficient to develop CAI systems in their present 'idealized' forms however. By 'idealized', I mean the presumption of artificially intelligent CAI systems. Rather, the system must be very close approximations or simulations of the actual classroom teaching process. The models must also interact with students as a classroom teacher does. It is important to understand that the simulation of a particular teacher's approach to instruction are not being discussed. Rather, the first task is to determine which variables are fundamental, but not necessarily exclusive, in the process of teaching.

Two advantages offered by this approach are presicion of control over the instructional environment and the ability to replicate an instructional paradigm. In a particular example concerning the relationship between teacher discourse and student performance there is an added advantage. The CAI model presumably generates a consistent discourse, possibly dependent upon the subject's protocol. Another advantage is that latent variables, which may be present in the classroom instructional process, might be discovered and explored. Still another advantage may be the ability to systematically 'fine-tune' variables. The underlying assumptions of the instructional process in the 'idealized' CAI systems may thus be verified.

There are several goals of this research. Although this author may not complete them, they are cited to demonstrate the general utility of our approach. The first goal, cited above, is the simulation of the process of instruction. The second goal is to model the protocol of a particular teacher, This is similar to the approaches taken by Newell (1966, 1967). The simulation of a particular teacher should not precede the simulation of the teaching process though; at least to the extent that this is possible. One reason is to assure that the model of a teacher incorporate aspects the teaching process in their proper perspective. It will be an important result to characterize what features are unique in the model of the teacher too. These features might be called, albeit without precise definition, 'teacher' variables. One can easily infer how these models (of teachers) can be used in making precise the claims and conjectures such as the recent Rosenthal & Jacobson (1968) results.

A third goal of this research is the use of the models (of the teaching process) in evaluating other CAI systems. This is accomplished by examining the target CAI model or system to determine the relative weighting of known variables (of the teaching process). It will then be desirable to characterize and verify the realtionships between those

variables and student performance. It is also a function of the cost of the model.

A final goal of this research is to encourage further research on CAI systems, programming languages, and simulation languages. One need not go far before he realizes that present systems and approaches leave a great deal to be desired. This is particularly true of commercially developed systems.

In summary, the principal interest of this work is to help bring about the development of CAI models that imitate, in a serious way, some of the features of 'good' teaching. This is not to conclude, however, that a model of the classroom teaching process is nessarily good.

3. THE 'LOGIC TEACHER'

The 'Logic Teacher' (LT) is a simple model of the teaching process. It is used in teaching sentential logic. The curriculum that it uses is developed by the IMSS (1967,1968) and has been used successfully throughout the USA. Some of the LT's features, which we describe below, are simply to permit an understanding of how these approximations affect the performance of the model. The model is being used to teach several classes of sixth graders.

The LT model includes aspects of the following salient cognitive traits in teaching behavior:

1. learning (by the instructional system) of new information about the student's behavior or subject matter;
2. alteration of the curricula to permit problem generation, deletion, replacement or reassignment as a function of individual and group differences;
3. forgetting of information concerning both the subject matter and the student's behavior;
4. deductive inferences about problems, their solutions, and the cognitive facts relating to these situations;
5. limited recognition of patterns and hidden concepts in order to emphasize the experiential details (and learning to incorporate them in the curricula (see item 1));
6. answer processing capabilities which include proof checking, stored answer analyses, and answer calculation;
7. 'optimization' of natural language discourse to minimize boredom, avoid redundancy, and accomodate individual differences;
8. answering questions posed in a restricted grammar of natural language such that succeeding dialogues may accomodate the questions that were asked;
9. ability to interrupt the instructional process to provide help or guidance, or to terminate a lesson.

The instructional discourse, and, hence, the student-teacher dialogue, thereby incorporates information concerning student and teacher cognition, problem and subject structure, and student performance. Descriptions of how some of these features have been implemented will be discussed momentarily. It is perhaps noteworthy that the features listed above easily relate to the characterization of the educational process as stated by Bruner (1962).

Before proceeding there should be some mention of features of the teaching process which are not a part of the LT. Foremost of these is what this author calls 'social and enviromental' learning. This would occur when two or more students interact with the teacher as well as among themselves. Few persons would deny this as an important consideration when describing the process of teaching. Another of these features might be called teacher incompetency or teaching capacity. This is in recognition of teaching variables being, in fact, a part of the teaching process. These and other features are deliberately ignored in this first pass at modeling the teaching process. It is felt that the nine items selected represent a sufficient and manageable sample of variables needed to demonstrate the technique of modeling classroom instruction. Nonetheless, future research is contemplated wherein these concerns will be accomodated.

3.1 *The 'LT' instructional strategies*

Because the LT is a CAI program several adaptations were required. Students using this system are remotely stationed at Model 33 TELETYPE consoles. The programs are run from a direct-coupled DEC PDP-1D/PDP-10 computer system operated by the Institute for Mathematical Studies in the Social Sciences (IMSSS), Stanford University. The computer programs are re-entrant, and coded in assembly language.

There are four basic modes of operation. Students are presented multiple choice, derivational, counter-example, and truth-analysis problems. This paper is only concerned with the derivational problems. An example of a problem of this type is found in Figure 1.

This problem is taken from the introductory logic textbook by Suppes (1957).
Problems of this type are presented linearly. The student is presently not permitted to purpue another problem until he successfully completes the problem in progress.

The student is expected to derive the desired conclusion by using any and as many of the given premises as necessary. This is accomplished by using parametrized mnemonics for rules of inference. These mnemonics and their corresponding rules of inference are taught by the LT.

Figure 2 demonstrates two solutions to the problem in Figure 1.

The student input is underlined (by this author) for clarity. The computer output is not underlined. At the time that a derivation is complete the LT types 'CORRECT' and proceeds to the next problem. Students therefore cannot specify a correct or wrong answer to these problems.

One observation, which should be obvious, is that students can pursue any proof. Creativity is therefore not stifled. The computer merely serves as a proof checker.

Dialogue may be necessary in the pursuit of either of these proofs. Imagine, for example, the student not being able to proceed past line 6 in the first proof. One reason may be his inability to recall how the 'DD' rule (deny the disjunct) is used. Another reason may be his lack of insight in how to proceed towards the conclusion. Either of these reasons may be inferred because of a prolonged latency to response.

Another need for dialogue may concern the working premise (WP) 'A & D'. This arises since the usual instruction concerning working premises emphasizes the introduction of the antecedent of a conditional expression as a working premise.

Figures 3 and 4 provide hypothetical examples of a dialogue between the student and the computer. Figure 3 would result because the LT has learned no proofs and it is necessary to generate a proof. The dialogue in Figure 4 depends on the student's previous history. The student's familiarity with the 'DD' rule is the prime determinant of the discourse generated by the LT.

"THE HORSE RACE PROBLEM"

Suppose that if A wins, then either B or C will place. If B places, then A will not win. If D places then C will not. Show that if A wins, D will not place.

Therefore ...
Derive : $a \rightarrow \neg d$

P (1) $a \rightarrow b \cup c$
P (2) $b \rightarrow \neg a$
P (3) $d \rightarrow \neg c$

Figure 1.

Solution No.1

Wp	(4)	a
1,4aa	(5)	b U c
2,4dc	(6)	$\neg$b
5,6dd	(7)	c
3,7dc	(8)	$\neg$d
4,8cp	(9)	$a \rightarrow \neg d$

correct

Solution No.2

wp	(4)	a & d
4ic	(5)	a
1,5aa	(6)	b U c
4rc	(7)	d
3,7aa	(8)	$\neg$c
6,8dd	(9)	b
2,9aa	(10)	$\neg$a
5,10fc	(11)	a & $\neg$a
4,5,10ip	(12)	$\neg$(a & d)
12dm	(13)	$\neg$a U $\neg$d
wp	(14)	a
13,14dd	(15)	$\neg$d
14,15cp	(16)	$a \rightarrow \neg d$

correct

Figure 2.

HYPOTHETICAL DIALOGUE USING THE BCGT PROCEDURE

wp	(4)	a
1,4aa	(5)	b U c
2,4dc	(6)	$\neg$b

Type H and a Space if you want help.

h — Do you want to work on an intermediate conclusion ? Y or N
y — Which one ?
c — Is the conclusion atomic or molecular ? A or M
a — Does the conclusion or its denial appear in any line?
y — What line?
5 — What is the dominant connective of line 5 ?
U — So it is a disjunction. If you have the denial of one disjunct, what rule lets you derive the other?
dd — Deny a disjunct. Do you see how to derive the denial of the other disjunct ?
y — OK. Do it.

5,6dd	(7)	c
3,7dc	(8)	$\neg$d
4,8cp	(9)	$a \rightarrow \neg d$

Correct

(Note : The above dialogue did not actually begin with the intermediate conclusion "c". It is assumed that there is a "scratch-pad" printout in which there is a list of intermediate conclusions that the student must derive. The above form should emphasize the wordiness of the BCGT procedure.)

Figure 3.

HYPOTHETICAL DIALOGUE USING THE LOGIC TEACHER PROCEDURE (ASSUMING STUDENT RECORD INDICATES FAMILIARITY WITH THE DD RULE OF INFERENCE)

wp	(4)	a
1,4aa	(5)	b U c
2,4dc	(6)	$\neg$b

Type H and a Space if you want a hint.

h — Now try to use the "dd" rule.

5,6dd	(7)	c

Very good, now finish the proof.

3,7dc	(8)	$\neg$d
4,8cp	(9)	$a \rightarrow \neg d$

Correct

Figure 4.

HYPOTHETICAL DIALOGUE USING THE LOGIC TEACHER PROCEDURE (ASSUMING THAT THE STUDENT RECORD INDICATES AN UNFAMILIARITY WITH THE DD RULE OF INFERENCE)

wp	(4)	a
1,4aa	(5)	b U c
2,4dc	(6)	$\neg$b

Type H and a Space if you want a hint.

h — Use the "dd" rule.
dd? — The form of the "dd" rule is ...

(x) A U B
...
(y) $\neg$A
...
x,yDD (z) B

(where either "$\neg$A" or "$\neg$B" can appear on line x and the other will be on line z).

5,6dd	(7)	c
3,7dc	(8)	$\neg$d
4,8cp	(9)	$a \rightarrow \neg d$

Correct

Figure 5.

In Figure 6 the student is being encouraged to pursue an alternate proof. He is expected to try to complete that proof in order to gain real 'hands-on' experience.' This procedure may be used in conjunction with either of the cases cited in Figures 4 and 5. It might follow the acknowledgement of the correct derivation.

The uage of alternate proofs may be useful in researching their effects on the student's proof-generating performance. They are also expected to enrich the experiences of problem solving. They should accomodate any individual differences that may be present too.

Further examples of types of comments that can be generated by the LT are found in Appendix A. See Appendix B for lists of some of the questions that are answered by the LT.

3.2 *Simulating teacher learning*

Teachers have at least two, if not more, reasons for learning. The first is to maintain the subject matter competency and their professional competency. This may involve the acquisition of new knowledge as a result of technological advances in their subject area. It may also involve the learning of better methods for teaching. A second type of learning in the teaching process in the acquisition of facts about student performance and behavior. This type primarily involves concept formation. The concepts are gleaned from observable performance data.

Learning in the LT involves the storage of different proofs, dialogue, and rule usage characteristics. Each new, distinct proof is stored in a linked list file (of proofs). The file of these proofs is accumulated for each problem. A maximum of eight proofs are stored. Future systems should attempt to weight problems so that proofs would be stored only if they are interesting ones. (One very simple way of doing this is by periodically purging files having small number of proofs. This would correspond to a forgetting strategy. It is, in some sense, analogous to the STS of the Atkinson & Shiffrin (1968) buffer model. One can also easily envision using rehearsal strategies to save small files which are used frequently. The initial LT model is a demonstrational model however. The intention is not that it be a complete model of the teaching process. A more elaborate scheme, like the one just described, would be needed in a comprehensive model of the teaching process.)

The 'LT' uses no strategy for censoring proofs except that derived lines, which are redundant or irrelevant, will be stricken from the proof. The reason for this decision is that, in the first version of the LT, it was found (administratively) desirable to not have the LT generate discourse related to those redundancies and irrelevancies. It is known, however, that the teaching process does involve learning of redundant and irrelevant information.

The proof procedure is stored along with the proof. That is,the results of applying axioms, rules, and theorems are saved along with the rule mnemonics and their parameters. This permits discourse concerning the rules and the concerning the rules and the consequences of having applied them. It also increases the speed of determining equivalent proofs.

The reasons for storing proofs as linked list are important. One reason is to facilitate the access and storage of proofs. It is considerably faster to manipulate pointers than continually adjust the text withing the file. But more important, this storage technique lends itself to efficient editing and it embodies aspects of the type of learning envisioned in the normal classroom situation. Another important reason is that since proofs are variable in lengths, disk storage may be preserved by locating them wherever space is available.

A second type of learning consists of accumulating individual and group characteristics of rule usage. Recall that the student has available to him certain rules of inference for deriving lines. His percentage of applications of each rule will be accumulated in a student record. Another record, indicating exposure to rules, is maintained for each student. Moreover, for each student there is a record of percentage of applications of rules for all other students exposed to the same rules. Thus individual as well as group performance may be compared. These data are used in the decision-making process when generating instructional discourse and other reinforcements.

One consequence of the above organization is that the brighter (faster) students and the earliest students to be taught by the LT receive discourse which is less rich and less varied. It is, therefore of interest to determine relationships between teacher learning and student performance.

It should be observed that the present discussion depends very little on the inherent properties of sentential logic. In fact, the LT can be used in teaching and subject whose format is analogous to curriculum described in IMSSS (1967,1968). The programs are being used in teaching logic, algebra, word problem solving, boolean algebra, and probability theory. The only difference in performance is that if no proofs are stored and if there is not a resolution procedure available for generating or directing the student to a proof, then the LT remarks that it... 'cannot help you.' In the logic curriculum a resolution procedure is available for generating proofs. It is the Binford, et. al., (1968) procedure, and has been described in Smith (1969).

3.3 *Dynamic alternation of the curriculum*

The LT employes simplified, dynamic item analyses to (sometimes) prescribe alterations to the curricula. It is assumed that the curricula are linear and homogeneous in 'difficulty' within lessons. That is, within lessons problems must be successiverly and successfully solved. They are also assumed to be ordered according to non-decreasing difficulty. Problems are reassigned only if they violate the monotonicity in difficulty assumption and if they have not been flagged indicating that are intentionally out or order – or specifically ordered by some other criteria. The curriculum author thus has the option of specifying that certain problems are exempt from the reassignment procedure.

It must be emphasized that the 'soundness' of the statistical tests are not a consideration here. The focus is a simulated description to teacher behavior.

The procedure for performing these analyses is as follows. Problems are stored on a disk file as linked lists. Problem reassignment becomes a task of manipulating the linkage pointers. One word of the problem record contains a count of the total number of errors (as defined below) and a count of the number of students who have attempted that problem. Problem reassignment may occur after the count of the number of students exceeds 30 (for the three reassignable problems) immediately before and after the current problem. When the number of errors for one problem exceeds the number of errors for a succeeding

problem the reassignment takes place. The actual process of reassignment is automatically effected when there are no students operating with the LT. This is to minimize systems problems since the LT is coded in re-entrant form.

Adding new problems to the curriculum causes no difficulties. The error count for a new problem is set to the average of the errors for the preceding and succeeding problem. The student count is set to the corresponding value of the preceding problem.

The notion of difficulty expressed here deserves an explanation. Because the derivational problems have no incorrect answer, difficulty is not expressed as the probability that the problem is correct. Rather, difficulty is defined in terms of the amount of discourse (overal students) required to complete the proof. The measure of difficulty will be the total number of errors, diagnostic, and hint messages generated for a problem. There is no attempt to control for characteristics of individual students. In other words, there are no controls for students who consistently require assistance by the LT.

3.4 *Recall of previous dialogue*

Problems requiring ten or more messages are recorded in a ten record file associated with the student's name. These problems, their proofs, and the codes identifying the message are saved. This information is purged each sixth session. (Six is an arbitrary choise. Its significance is that each sixth session should correspond to the beginning of a week. It is assumed that forgetting may occur in that period of time.)

Each time a diagnostic message is needed this file is scanned to see if the message was required in any of the proofs (and dialogue) saved. The proof is compared with the one that the student is using. Any correspondences found may be discussed depending on the values of certain internal weighting functions.

3.5 *Deductive inferences and heuristics for question asking and answering*

The concern is, among other things, that the 'LT' not require large data bases. Another concern is to use the computer to assemble well known facts (data) to produce less than obvious inferences. Simply stated, to expect that the lesson author will adequately perceive the many variations of correct answers runs counter to programming experience.

The 'LT' uses information which it has 'learned' in generating instructional discourse. This is particularly true of the stored proofs. It (the LT) scans those proofs for a derived line which is common to the proof that the student is pursuing and those on file. The scanning proceeds by examining the last lines of the student's proof and the stored proof and searching backwards through the stored proof until a match is found. If no match is found the algorithm considers the line before the last of the student's proof and proceeds similarly. If no lines of the student's proof occur in any of the stored proofs the 'LT' discusses the shortest stored proof. If no proofs are stored (learned) it uses the Binford, et al., scheme for generating a proof, which, in turn, is saved. Scanning of the stored proof always begins with the shortest proof.

There are many other strategies for scanning the file of stored proof for matches. The one cited above has the virtue of simplicity. One can easily imagine examining all proofs and choosing the one for which there are more lines in common. The scheme used by the 'LT' does have the advantage that if the student changes his proof, in mid-course, then any dialogue will probably be concerned with the most however. It can direct the student through a desired proof in order to however. It can direct the student through a disired proof in order to expedite matters.

4. CONCLUSIONS

Our research has shown that a useful technique for educational researchers may be the usage of computer models of the teaching process. The concept of modeling need not be restricted to those models of teachers or the teaching process. One can easily infer their usefulness in a variety of other areas. The simulation of school systems, conflict resolution, teacher (and tenure) evaluation, and administrative decision-making are all obvious candidates for these methods. This is particularly true in view of the size sophistication offered by many computing systems.

The 'LT' model was chosen in order to research the relationships between teacher discourse and student performance. Although these results will be reported elsewhere, we shall mention how the 'LT' model is relevant to this question.

The instructional discourse generated by the 'LT' relates to either lines of proofs or to previous dialogues. Hence one can systematically change which line of a proof is discussed. He would examine the student performance data for systematic changes in performance. For example, one group of appropriately chosen students might receive one type of message while another group would receive another type of message. Considering these as trearments an analysis of variance might be used to determine differences in performance. A protocol analysis may also be used.

Another method of examining this question is to consider proofs to the derivational problems. If systematic changes in the weightings of the variables lead to gross changes in the types of proofs generated a similar conclusion could be reported. (That is, that there is a relationship between teacher discourse and student performance).

Finally, it is hoped that the present research will challenge educators, psychologists and computers scientists to develop adequate CAI systems. These systems must be evaluated according to practical as well as empirical considerations.

Figure 6. EXAMPLE SHOWING STUDENT BEING LED THROUGH ALTERNATE PROOF

```
...
Let's solve the same problem using another proof.
I'll supply one or more of the steps.

Derive:  a     d
p        (1)       a → b U c
p        (2)       b → ¬a
p        (3)       d → ¬c
wp       (4)          a & d
         Now you try to complete the proof.
                  Type H and a Space if you want a hint.
h        Try using the "lc" rule.
4 lc     (5)       a
4rc      (6)       d
1,5aa    (7)       b U c
3:6aa    (8)       ¬c
         Remember how to use the "dd" rule.
6.8dd    (9)       b
2.9aa    (10)      ¬a
5.10     (11)         a & ¬a
                  Type H and a Space if you want a hint.
h        See if you can derive ..."¬(a & d)".
h        Try the "ip" rule, then.
4.5.10ip (12) ¬(a & d)
                  Type H and a Space if you want a hint.
12dm     (13) ¬a U ¬d
         Good, I was going to suggest that.
wp       (14)      a
13.14dd  (15)      ¬d
14.15cp  (16) a → ¬d     Correct,
                         I've just noticed that you didn't use line 11.
                         Your proof could have been reduced by 1 step.
```

REFERENCES

Atkinson, R. and Shiffrin, R., (1968), Human Memory: A Proposed System and Its Control Processes. In: Spence, J and Spence, eds., The Psychology of Learning and Motivation, Vol. 2. (Academic Press, New York) pp. 89-195.

Bellack, A. & Davits, J., (1963), The Language of the classroom: Meanings Communicated in High School Teaching, Institute of Psychological Research, Teachers College, Columbia University, New York.

Binford, F., Cox, M., Galda, K., & Titiev, R., (1968), Logic Dialog Proposal. Institute for the Mathematical Studies in the Social Sciences. Stanford University, Stanford, California.

Bruner, J., (1962), The Process of Education, (Harvard University Press, Cambridge, Mass.).

Flanders, N., (1965), Teacher Influence, Pupil Attitudes, and Achievements. Cooperative Research Monograph No. 12, U.S. Office of Education, (U.S. Government Printing Office, Washington, D.C.)

Gage, N., (1965), Research of Cognitive Aspects of Teaching. A Seminar on Teaching. Sponsered by the Association for Supervision and Curriculum Development and The NEA Center for the Study of Instruction.

Getzels, J., & Jackson, P., (1963), The Teachers' Personality and Characteristics. In: Gage, N., ed., The Handbook of Research on Teaching, (Rand McNally and Company, Chicago).

IMSSS (1967, 1968), First Year Logic Curriculum, Institute for the Mathematical Studies in the Social Sciences, Stanford University, Stanford, California.

Newell, A., (1966), On the Analyses of Human Problem Solving Protocols. In: Proceedings of the International Symposium on Mathematical and Computational Methods in the Social Sciences.

Newell, A., (1967), Studies in Problem Solving: Subject 3 ont the Crypt Arithmetic Task Donald+Gerald=Robert. Technical Report, the Carniegie-Mellon University, Department of Computer Science, Pittsburgh, Penn.

Rosenthal, R. & Jacobson, L., (1968), Pygmalion in the Classroom, (Holt, Reinhartmand Winston, New York).

Ryans, D., (160), Characteristics of Teachers, (American Council of Education, Washington, D. C.).

Smith, R. Wm., (1969), Modeling Instruction Using Computer Generated Dialogue, Unpublished doctoral dissertation proposal, School of Education, Stanford University, Stanford, California.

Smith, B., & Meux, M., (1962), A Study of the Logic of Teaching. Bureau of Educational Research, College of Education, University of Illinois, Urbana, Illinois.

Smith, B., Meux, M., et al., (1964), A Tentative Report on the Strategies of Teaching. Bureau of Educational Research, College of Education, University of Illinois, Urbana, Illinois.

Taba, H., (1963), Teaching Strategies and Thought Processes, In: Teachers College Record, pp. 524-534.

Suppes, P., (1957), Introduction to Logic, (D. Van Nostrand, New York), p 30.

*The research reported herein has been supported by National Science Grants NSFG-18709 and NSF GJ-433x. The author gratefully ackowledges this support and the personal support of the principal investigator, Professor Patrick Suppes.

Appendix A.

RESTRICTED SUBSET OF NATURAL LANGUAGE USED TO QUERY THE LOGIC TEACHER DEALOGUE PROCEDURE

Note: In the following the syntactic variables cannot contain both "and" and "or".

I. Is x y? Are x and y?
x :: <expression> | line <number> | li <number>
x :: x or x
x :: x and x
y :: needed | required | ne | re

II. Is x y in z?
x :: <symbol> | <connective>
y :: dominant| a connective| connective| do| co| a co
z :: <expression> | line <number> | li <number>
z :: z or z
z :: z and z

III. x?
x :: <rule mnemonic>

IV. Why?

V. Can x y?
x :: <expression> | line <number> | li <number>
x :: x or x
x :: x and x
y :: be eliminated | be eliminated from the proof | be deleted | el | de

VI. Is there a shorter proof?

VII. x another proof?
x :: cite | can you cite | is there | list| li | ci| is

VIII. x y proof.
x :: show | show me | display | sh | di
y :: another | shorter| longer |<number> step | <number line> |an | sh | lo | <number>

IX. Can x be used? [Can I use x?] [Is x used?]
x :: <any rule mnemonic> rule <any rule mnemonic> <the any rule mnemonic> rule | <any rule mnemonic> rule

X. What x between y?
x :: difference is there | is the difference
y :: this proof and the last | these two proofs | these proofs

Appendix B.

PARTIAL LIST OF COMMENTS GENERATED BY THE LOGIC TEACHER

1. No, x is not y. [x and y are as in I of Appendix B.]
2. Yes, x is y. [x and y are as in I of Appendix B.]
3. x may be y. This depends on which proof you pursue.
4. Look at your proof again. Notice that x is not referenced after line <number>. It therefore must not be y.
5. See if you can decide the answer to your question. Try recalling my comment in problem <problem number>.
6. Look-<student name>... there is only one possible answer to that question.
7. Yes, x is y in z. [x, y, and z as in II of Appendix B.]
8. No, x' is y in z. [y and z as in II of Appendix B. x' is the correct answer.]
9. Yes, x can y. [x and y as in V of Appendix B.]
10. Yes, x can y. x' can y too. [x and y as in V of Appendix B.]
11. Yes, I can think of <quantity> shorter proofs than yours.
12. I am not able to think of a shorter proof.
13. Surely there may be longer proofs.
14. If I correctly understand what you're trying to do then the <rule mnemonic> rule should be used.
15. You should be able to complete the proof without using the rule mnemonic rule.
16. Can you cite which rule you wish to use?
17. Can you tell me more. For instance, which working premise would you like to use? <Subject may answer "I don't know" to this or the next question.>
18. Well, which ones would you not like to use?
19. Try using the <rule mnemonic> rule.
20. Now, try to get the expression expression .
21. I've just noticed that you didn't use x. Your proof could therefore be reduced by y. [x :: line <number> .x :: x and x. y :: 1 step [< number gtr 1> steps.]
22. I just answered that in the last problem.
23. I now think that I see what you're trying to do. But first try to show that "<expression>".
24. I think that I've said enough for now.
25. Enough said...let's see you finish things.
26. Your proof seems to be an interesting one. It uses the x where other students have not. [x :: rule <rule mnemonic> | expression <expression>].
27. That is commendable.
28. Notice the differences in line(s) <line number(s)>.
29. Observe that x. [x :: different rules are used | line(s) <line number(s)> are different | rule <rule mnemonic> was used on line <line number>.]

CURRICULUM MANAGEMENT BY COMPUTER: AN ASPECT OF COMPUTER MANAGED INSTRUCTION

John F. Vinsonhaler, Clarence L. Winder, Clyde Morris, Daniel Millin

1. CURRICULUM MANAGEMENT SYSTEMS

Curriculum design should be an ideal field for operations research Churchman [1957]. Yet, over the past few years the present authors have observed the failure of several competent O.R. teams in curriculum design. We are convinced that the major source of failure is in the emphasis upon completely specifying educational objectives and rather restrictive selection from among the complete set. In our judgement, only rarely can an educational community clearly define a set of basic, static objectives. At best, a universe of objectives can be identified and a temporary priority assigned each objective. We emphasize that while role differentiation among students, faculty, and administrators is maintained, the system proposed operates to increase the impact of students and the current faculty on the curriculum. This increase in influence could be dramatic in its proportions.

One of the fundamental problems in college management is the lack of a formal model for the process of curriculum revision. We shall attempt to avoid this difficulty by immediately establishing a communications model for curriculum management. In general, the present model is designed to function in accordance with the principles of modern communication theory Garner [1962] and the three theoretical axioms summarized below.

First, the academic community is assumed to be a restricted communication system, as defined by Cherry [1966]. Thus, a college is a social unit established through the use of a common language or set of problems. Second, the main elements of communication are descriptions of educational objectives. Each member of the community (faculty, administrator, and student) enters the system with a unique set of educational objectives. Daily instructional activities are determined by the curriculum, i.e., some encompassing set of instructional activities. Third, the major objective of the curriculum management is to maximize the educational satisfaction of community members by helping them achieve their educational objectives. For proper functioning, the system must provide reliable communication of educational objectives among faculty, administrators, and students.

Figure 1 provides a formal model of the communication process in the traditional curriculum management system. As illustrated, most traditional curriculum management systems are based upon a linear communications network. In such networks, communication mainly occurs between personnel at equivalent levels. Even here, communication is handicapped by the lack of a standard language for use in stating agreements and conclusions with a reasonable degree of precision. Communication between levels (e.g., between the Dean and any given student) is subject to much distortion and loss. The curriculum management system summarized in Figure 1 illustrates two basic communication problems: message reliability and message validity.

Consider the problem of message reliability. Suppose you are a student hoping to transfer an opinion about educational objectives to your instructor, your department chairman, and your dean. Assuming that messages have a fifty percent chance of exchange, the probability of your message reaching the chairman is .50 x .50 or .25, since you must communicate indirectly through your instructor. Your chance of reaching the dean drops to .125. The problem of message reliability can be solved by providing multiple/direct channels of communication.

The problem of message validity concerns the effect of a message on the receiver. The message is valid to the extent that the receiver understands the educational objective of the sender. The method usually employed to make messages more valid is to standardize the language used to state messages. Thus, the problem of message validity can be partly solved by standardizing the descriptions of educational objectives.

In our judgement, traditional curriculum management systems have at least two serious faults: they lack sufficient lines of communication and they lack a standard language for stating objectives. Given the availability of a computer-based information management system, these main faults can be corrected through a common information file which could be independently accessed by all members of the academic community.

2. UDEANS: THE UNIVERSITY DEANS' INFORMATION SYSTEM

The UDEANS curriculum management system is an operational system now functioning on the CDC 3600 computer at Michigan State University. However, it should be emphasized that UDEANS is only a prototype developed for research and demonstration within the College of Social Science. The UDEANS system was developed with the Basic Indexing and Retrieval System (BIRS). The BIRS programs are used to store, manipulate, and retrieve data from the UDEANS information file. Essentially, BIRS is a collection of portable Fortran programs designed for information management in education and social science. Copies of BIRS are available at cost from the Information System Laboratory at Michigan State University, Vinsonhaler and Hafterson [1969]. The UDEANS system was developed in cooperation with the School of Social Work in the College of Social Science at Michigan State University. The school is an academic department with about 500 students, 20 faculty members, and 20 undergraduate courses. It grants both bachelors and masters degrees.

The first step in establishing the UDEANS system was the development of a standard method of stating education objectives by creating a formal statement of the major concepts included in the courses taught. After much consideration, a questionnaire, or profile, was decided upon. The profile consisted of a list of educational objectives as stated by faculty, students, and administrators in the School of Social Work.* An individual records his educational interests for UDEANS by assigning a number between one and five that is proportionate to his interest in each objective.

Figure 2 shows a partial list of objectives or concepts used to characterize each individual faculty member, student, and course. Faculty and student profiles are developed by having each individual score himself on the basis of a 5 point „interested-not interested" scale for the list of objectives. The UDEANS profile provides the fundamental data for the information file used in curriculum management.

Since UDEANS has yet to be regularly used for curriculum management, we can only suggest some patterns of application. Let us begin by considering the overall functioning of the system as summarized in Figure 3. As shown in the figure, the UDEANS operation may be analyzed into three main components: input, processing, and output.

The input for UDEANS consists of data records for faculty, students, and courses. At minimum, each record must contain identification and a UDEANS profile of educational objectives. The system can accommodate any other type of information required for curriculum management, e.g., faculty salaries and publication lists, student course records and programs; course costs and projected enrollments; etc.

All processing of the UDEANS data records is performed by the BIRS programs. The information input for faculty, students, and courses is stored on magnetic tapes as illustrated in Figure 3. As shown, two files are maintained: the data file containing data records, and the index file which is used to search for data records to be used for report generation.

The output of UDEANS may be classified into several basic types: student reports, instructors' reports, research reports, and planning reports. Figure 3 shows that each of these reports is designed to serve one or more of the types of individuals involved in curriculum development: students, faculty, administrators, and curriculum committees.

3. UDEANS REPORTS FOR STUDENTS

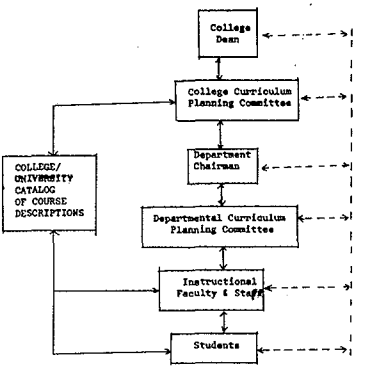
The system may be used to generate two types of reports for students:personalized catalogs of courses and directories of faculty members. These catalogs consist of course or faculty member résumés listed in order of similarity to each student's personal interest profile. Figures 4 and 5 illustrate the input and output associated with students' reports generated by UDEANS. As shown, the reports are requested by specifying the type required (faculty or course) and giving the student's interest profile. The output consists of faculty sketches or course numbers ordered by similarity to the student's interest profile.

Both courses and faculty members are listed in order of similarity. Thus, in the course catalog SW 810-811 has the

5

Figure 1. Curriculum Management by Traditional Methods: The Linear Communications System Model.

The model assumes that the curriculum (universe of educational activities) is based upon the utilities assigned to a specific set of educational objectives. Communication is restricted to the exchange of utilities and educational objectives among faculty, students, and administrators.



8

Figure 2. UDEANS Profile of Educational Objectives

Concepts Encountered in Dealing With the School of Social Work

Sources: 1. Michigan State University Catalogue
2. Replies to question "What concepts are you interested in?" Asked of seniors in the Department of Social Work during survey taken March, 1969.

Rating Scale: 1 2 3 4 5
(Low Interest) (High Interest)

Practical Work Settings

__Juvenile Delinquency
__Family Services
__Adoption
__Teaching Emotionally Disturbed Children
__Mental Health
__Child Clinic
__Work with Blind, Handicapped
__Social Planning
__Probate Court
__Counseling
__Administration

Professional Preparation

__Orientation to Agency Policies
__Social and Emotional Development
__Legislation in Social Work
__Dealing with Emotions of Clients

Study Approaches

__Family Interactions
__Welfare Rights
__Behavioral Approach
__Rogerian Human Relationships
__Public Agency Studies
__Federal-State Local Agencies and their Relationships

Attitudes to be Developed in Social Worker

__Self-Awareness
__Confidentiality
__Self-determination
__Individualization (Treating Clients as Individuals, not as Book Stereotype)

highest degree of similarity to the student's interest profile, while SW 813 has the lowest degree of similarity. The relevance value indicates the degree of similarity for each faculty member or course. The purpose of such individualized listings is to ensure that students can locate faculty members and courses concerned with their individual interests.

4. UDEANS REPORTS FOR INSTRUCTORS

The system is capable of generating a variety of useful reports for instructors. For example, each instructor may obtain a personalized directory of students listed by similarity of student and educational interests to those of the faculty member. The instructor can obtain a special summary of educational interests for any given group of students, e.g., the students currently enrolled in his course.

The retrieval methods are similar to those used for student reports. The instructor simply indicates the type of report desired and specifies his profile of educational interests. The system automatically generates lists of courses or students ordered by similarity to the instructor's interest profile. The output format is analogous to the illustrations above, for students.

5. UDEANS REPORTS FOR ADMINSTRATORS: THE CURRICULUM REVIEW ANALYSIS

The UDEANS system is capable of generating an impressive array of reports which would be valuable in curriculum planning. We can demonstrate the tremendous potential of computer-based curriculum planning by describing one easily obtained report: the curriculum review analysis. Essentially, the curriculum review reports are statistical summaries of the educational interest profiles of students, faculty, and courses. These reports provide curriculum planners with periodic overviews of the shifting pattern of educational objectives in the faculty and the student body. They also provide a periodic review of the major emphasis in the courses offered within a department or college. Figure 6 illustrates a curriculum review analysis of student profiles. This report is generated by submitting a complex search request specifying all possible interest patterns to UDEANS. The resulting output is then combined into a table showing the number of students, faculty, or courses with various levels of interest in each of the educational objectives included in the UDEANS profile.

One can imagine many uses for the type of analysis illustrated in the above figure. First, the faculty and administrators could compare the reports of faculty, students, and courses to identify divergent patterns of interest which might present serious problems – e.g., the faculty is interested in research goals while the students are interested in occupational goals. Second, the faculty and adminstrators could use student reports to keep aware of the interest patterns of the student body. In the example, the problem of adoption is clearly a popular topic for students in social work, while the problem of working with handicapped is much less interesting to most students. If this is a topic of real importance, special attention to the method of instruction may be necessary to enhance student interest as much as possible. Finally, the curriculum review analysis can provide a rational basis for curriculum changes. The UDEANS report gives reasonably firm numerical data on what the consumers and producers of higher education

Figure 3. *Curriculum Management with UDEANS: The University Deans Information System*

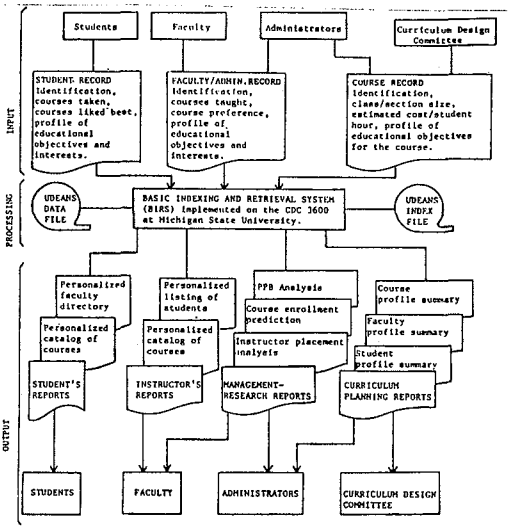
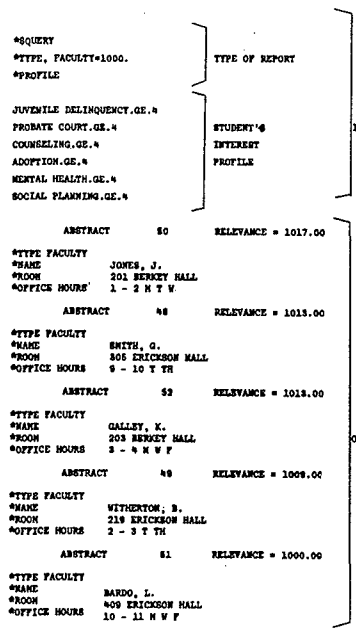


Figure 4. *Faculty Directory*



say they want from the university. These data may be directly applied to restructuring the curriculum to meet revised educational objectives.

6. UDEANS REPORTS FOR ADMINISTRATORS: PROGRAMMED PLANNING AND BUDGETING (PPB)

Among the most important recent developments in managerial technology is the concept of PPB – Programmed Planning and Budgeting. Essentially PPB is a class of procedures which directly allocates resources to well-defined objectives – rather than allocating them to administrative organizations. The UDEANS system can be used to help administrators review their allocation of funds by educational objective rather than by department, school, or college.

For example, the administrator can ask for a summary of all faculty or course records in which the objective of teaching „ADMINISTRATION" is assigned an interest weight „greater than or equal to" level three (GE to 3). The resulting output would show the allocation of faculty and courses, i.e., the number of faculty members in the system who show a high interest in „ADMINISTRATION" and the number of courses in which „ADMINISTRATION" objectives are emphasized. In the case of the School of Social Work under study, the UDEANS report showed that the major allocation for training in administration consists of the salaries for two interested faculty members and the costs required for four related courses. Obviously, the UDEANS report could reduce the allocation to dollar expenditures, provided such data were available in faculty and course records.

7. UDEANS REPORTS FOR ADMINISTRATORS: COURSE ENROLLMENT PREDICTION

To complete our review of the types of analysis which may be performed by computer-based curriculum management systems, consider predicting the next year's enrollment. At minimum, the accurate prediction of enrollment for any given college course requires three types of data: the total number of students in the program; the proportion of students who have taken the course in previous years; and the number of students whose program specify the course. If the UDEANS student data records include a list of courses taken and courses planned, the system can generate analyses which are valuable for predicting future course enrollment.

One type of enrollment prediction is made by establishing the upper and lower bounds for enrollment in a given undergraduate course. The lower limit for enrollment is determined by counting the number of those who must take the course during the next academic year. The upper limit is estimated by counting the total number of students who intend to take the course.

8. CONCLUSIONS AND IMPLICATIONS

The computer-based curriculum management system provides a means of continually adapting the curriculum to the needs of faculty and students. Such a management system can open lines of communication which are presently non-existent. It can provide for regular, periodic review and improvement of the college curriculum to meet

Figure 5. Course Catalog

```
*SQUERY
*TYPE, COURSE=1000          TYPE OF REPORT
*PROFILE

JUVENILE DELINQUENCY.GE.4                                   INPUT
PROBATE COURT.GE.4
COUNSELLING.GE.4
ADMINISTRATION.GE.4         STUDENT PROFILE
ADOPTION.GE.4
MENTAL HEALTH.GE.4
SOCIAL PLANNING.GE.4

        ABSTRACT        58          RELEVANCE = 1025.00

*TYPE   COURSE
*NAME   810, 811
*TITLE  HUMAN GROWTH AND BEHAVIOR:  NORMAL DEVELOPMENT

        ABSTRACT        55          RELEVANCE = 1021.00

*TYPE   COURSE
*NAME   830
*TITLE  APPLICATIONS OF SOCIAL SCIENCE TO SOCIAL WORK PRACTICE

        ABSTRACT        54          RELEVANCE = 1019.00

*TYPE   COURSE
*NAME   820, 821
*TITLE  SOCIAL WELFARE ORGANIZATION I AND II

        ABSTRACT        57          RELEVANCE = 1018.00

*TYPE   COURSE
*NAME   860
*TITLE  COMMUNITY ORGANIZATION I

        ABSTRACT        58          RELEVANCE = 1006.00

*TYPE   COURSE
*NAME   813
*TITLE  HUMAN GROWTH AND BEHAVIOR:  PSYCHOPATHOLOGY
```

Figure 6. Curriculum Review Analysis

STATISTICAL REPORT:

STUDENT EDUCATIONAL OBJECTIVE	INTEREST PROFILES LEVEL OF INTEREST (NUMBER OF STUDENTS) LOW 1	2	3	4	HIGH 5	NO ANSWER	TOTAL
FAMILY SERVICES	2	6	17	17	5	--	47
ADOPTION	3	5	7	18	14	--	47
TEACHING EMOTIONALLY DISTURBED CHILDREN	5	7	15	13	5	2	47
MENTAL HEALTH	3	6	13	4	20	1	47
CHILD CLINIC	1	3	17	13	11	2	47
WORK WITH BLIND & HANDICAPPED	8	17	17	1	1	3	47

changing student educational needs and changing faculty interests and competencies.

It would appear to the present authors that computer-based curriculum management systems could provide the basis of complete Computer Managed Instruction systems for college and university education. However, it is very likely that such automated systems will be faced with powerful opposition within the academic community. Two stereotypic examples are already familiar to the present authors: The Traditional Academic and Student Radical.

The Traditional Academic rejects computer managed curriculum systems on the basis that the reports generated are irrelevant because he believes curriculum changes should be designed intuitively in terms of abstractly defined goals rather than in terms of student and faculty interests. The Traditional Academic argues that revisions in the curriculum should be prescribed by faculty with highest academic rank and impeccable traditional scholastic credentials.

The Student Radical rejects the concept of computer-based curriculum management on an emotional basis believing that computers can not possibly provide the means for achieving humanistic goals. The Student Radical argues that adequate curriculum revisions should be prescribed by student ultimatums – supported by the threat of disruptive demonstrations.

Both the Traditional Academic and the Student Radical can invalidate any curriculum management system like UDEANS by encouraging faculty and students to behave irrationally, e.g., by deliberately falsifying interest profiles. Such systems can not be imposed by higher authority; they must be developed jointly by administrators, faculty, and students.

In summary, computer-based curriculum management systems have great potential for the reformation of higher education by improving communication and providing empirical data for discussions and decisions. However, such systems also have a potentially fundamental weakness – they require the honest, active participation of the entire academic community in a spirit of mutual accommodation by all concerned.

REFERENCES

Brudner, H. J., 1968, Computer Managed Instruction. Science. 162, pp. 970-976.

Cherry, C., 1966, On Human Communication: A Review, A Survey and a Criticism. 2nd ed. (MIT Press, Massachusetts Institute of Technology).

Churchman, A. A., 1957, Introduction to Operations Research (Wiley and Sons, Inc., New York).

Garner, W. R., 1962, Uncertainty and Structure as Psychological Concept (Wiley and Sons, Inc., New York).

Vinsonhaler, J. F., and J. M. Hafterson, eds., 1969, Technical Manual for the Basic Indexing and Retrieval System, Appendix I, BIRS 2.5 (Michigan State University, East Lansing, Michigan).

Footnote

The reader should know that this particular list is only a first approximation. Work continues toward a refinement of the list and a more adequate concept of the important properties of such a list.

AN OFF-LINE APPROACH TO COMPUTER AIDED LEARNING

William Russell Broderick

1. INTRODUCTION

There is little doubt that the ideal teaching situation is where you have one teacher and a very small group of students. Perhaps under some circumstances the one-teacher-one-student approach is ideal although this is not universally so. The aims of computer-aided learning contain an implicit and sometimes explicit recommendation that we should attempt to use the computer to attain this ideal. Alas! However, at this point in time this is an expensive solution to a problem and whilst a considerable amount of research is being done in this field on both sides of the Atlantic, it will be some time before national economies are in a state to absorb the high levels of expenditure that this approach demands.

However, there are other benefits of computer-aided learning which it may be possible to attain without recourse to the one-student-one-terminal approach; amongst these are individualised instruction, careful control of management of the learning situation and help for the teacher in planning and administering the work in the classroom. What is more, if individual student terminals are not required then it may prove possible to use the sort of computer which is available in many local education authorities for their normal data processing work.

2. PROPOSED SYSTEM

The system we are proposing relies on the computer to select for each student a work assignment – a Computer aided Learning Task (CALT) – from an analysis of stored information and of data about the student's abilities and attainments. The student works on this task, returns a completed questionnaire which is analysed by the computer system, and the most appropriate next assignment is selected for him. Information about student progress is collected for the benefit of the teacher, as also is information about the effectiveness of the specific sections of the course. The CALT's are in fact modules of a completely programmed course of instruction.

The use of the computer to control the student's path through such a series of programmed tasks allows the logic of the system to be flexible whereas with conventional teaching machines or programmed learning texts the logic has to be defined from the outset, and the conventional system is less adaptive to individual circumstances than is a computer-based system. Further, these more passive systems are less capable, for practical reasons, of continuous up-dating and modification, whereas any particular CALT may be revised as necessary.

It is possible, to use the computer to produce short individual booklets displaying the same information as that obtainable from an on-line teleprinter or CRT display system. As a result of the computer's assessment of previous performance, each student would be provided with a booklet of such a length as to cover the work of the next lesson. It would be quite simple to produce these on a line printer. In practice the fully developed system could work along the following lines:

1) All students would be given an initial CALT booklet setting out a module of instruction, practical work and a diagnostic questionnaire – a programmed learning task.
2) They would record, on a from or mark sensed card, their response to particular questions.
3) Following the end of each lesson, these response forms of cards would be collected and transmitted to the computer centre (by courier or other rapid means).
4) The computer centre would then process the data indicating for each student his response to the set tasks and, depending upon these responses, produce the most suitable selection of the successive programmed sections for his next lesson (overnight turnround is possible using this system).

The student would then proceed with subsequent CALT's following the steps set out in (2), (3) and (4) overleaf until the course was completed. It is extremely important that the students should be given frequent check tests in the process of following their programmed courses. These diagnostic tests and remedial exercises would be produced and marked, for the benefit of both the student and teacher, by the computer. These tests could and should be especially designed to test the particular fields in which the students' performances had been questionable.

3. SELECTING A LESSON LENGTH

The approach which the team at The Royal Liberty School Computer Department is investigating is based upon a series of individualised work tasks which the students will work through and at the end of each task, tests will be administered. These tests will be used both diagnostically and prescriptionally, i.e., to select any remedial work for the student and to prescribe the best approach for his next lesson. Adopting this approach, one is faced with a variety of problems and it is some of these problems that I wish to discuss in this paper.

Let us take a closer look at the one-student-one-teacher situation: here the teacher makes a selection of ideas, usually based upon his own personal teaching experience, and from these ideas he will select those most appropriate to teach the particular student. At the end of a short presentation or discussion, the teacher will usually ask the students some questions and from the answers to these

questions will diagnose what additional explanation is required and also use these answers (often subconsciously) to formulate the best future presentation of material. For an on-line computer aided learning situation, the approach is normally very similar and indeed it is this approach which is perhaps simplest to apply because of the immediacy between the presentation of material (usually in very small quantities) and the study of the student's response to that small section of the material. In an off-line situation one has to form a compromise between the very small work tasks which may last between 3 and 6 minutes, and large work tasks which may last a whole lesson; the small work tasks create administrative problems in the classroom, and if the feedback for the computer system is only coming back once every 24 hours (reasonable frequency, taking into account the practical problems involved) then there is undoudtedly going to be a problem in estimating the amount of work the student is going to achive over the period. This problem will be more difficult in the initial stages than in the later stages when a profile of the student's performance can be built up and used to estimate his rate of working. This rate of working however will vary for different types of material.

An additional problem is met in assessing how long the work tasks ought to be so as to ensure that i) the student does not run out of work during the lesson and ii) he does not have a large amount of work in his hands which he has not completed. The interval between sending work back to the computer system for assessment and receiving his next set of work to be prescribed as a result of this assessment is critical when considered in relation to the amount of work that the student has in hand at the end of a lesson. If the work which the student has not yet completed contains items for which responses are required and if these responses will significantly affect the prescription of future work tasks then it is highly undesirable that the prescription be made before these responses have been analysed by the computer. Yet with an off-line system, data collection has to be timetabled carefully. It is therefore necessary to ensure that either:a) any work task that the student is unlikely to complete does not contain responses which will significantly affect the prescription of future work:or b) the expected duration of the work tasks should be small so that the amount of work outstanding from the computer's point of view can be minimal;or c) if the work tasks are to be of a significant size relative to the duration of the lesson then the questions generating responses have to be embedded in the work task at frequent intervals to provide close monitoring of the student's work. It is this that last course of action that we favour particularly if the interval between sending work back to the computer system and the student receiving his next allocation of work is to be short (overnight turnround for instance).

It is hoped that by adopting this strategy we will be able actually to base the prescription upon the maximum possible amount of work completed by the student. This is clearly the best approximation we can achieve to the immediate feedback normally expected with computer-aided learning, and it is certainly much more reliable than a system whereby all the questioning and testing is done at the end of a work task, for it removes the need to break work tasks into very small modules.

An additional problem arises here when one comes to study the problems of the presentation of remedial work; clearly a remedial section of the material can be very short – five or ten minutes – or it may take a whole lesson. This is dependent upon the student's performance in the diagnostic and prescriptive tests. For these reasons we have decided that in general we will aim at tasks which will take an average student ten minutes, and we have devised an algorithm to study the student's rate of working and present him with individualised learning tasks that he is likely to have completed at the end of the lesson.

4. STRUCTURE OF THE EDUCATION MATERIAL

We consider the learning material capable of being represented by a network, the nodes of which each represent an educational objective that should be attainable in about ten minutes. The amount of educational material in a node will vary considerably with the abilities of the student. The links between nodes in the network are possible entry routes to or exit routes from a node. Therefore a path through the network defines a course of learning. A group of nodes will be combined to give a lesson; a series of lessons will be combined to give a course of learning. Furthermore, there are many well defined algorithms for handling networks within computer systems and we hope that it will prove possible to use the Graph Traverser program (Doran & Michie 1966) to learn (in the machine intelligence sense) the best way through the network for any given student's behavioural characteristics that have been demonstrated to the computer system in the past.

5. CHOOSING AN ASSESSMENT SYSTEM

This approach does present additional problems in that many students arriving at the same node in the network at different times by different paths have to be able to be compared at that point in the network. Making this comparison is a non-trivial problem for if one decides to adopt a taxonomy of educational objectives (e.g., the one by Bloom and others) and to test the student's behaviour at various levels within this taxonomy, then it is quite likely that students following paths at different levels (and of different lengths) have been tested on different criteria. For example, those at a lower level may have been inadequately tested at a higher level in the taxonomy (and perhaps have been inadequately taught at this level as well) whilst those following the path at a higher level would have been more thoroughly tested and taught. As a result of this difficulty, one is presented with a problem in devising a marking scheme for the system. We have a number of possible schemes under review. A basic design criteria however which we wish to build into the marking scheme is that the person writing educational material should be able at any point in time to have a high degree of flexibility in deciding upon the number of marks to be given for many activity. We would then wish to use the computer system to ensure that these marks become comparable when the students arrive at a node which could be reached by any alternative paths. It appears at first sight that a possible solution here may well be to adopt the concept of „the shortest path" to a node and consider this to be entirely analagous to the idea of lowest number of maximum marks obtainable at any particular level of the taxonomy to reach that node. As a student progresses through the network the computer must accumulate for him the maximum number

of marks he can score in the taxonomy and scale his marks down in proportion to the maximum he might have achieved and the maximum that could have been achieved by following the shortest path thus producing comparable marks. However, this system is not as simple as it appears for it is possible for some levels of taxonomy not to have been tested on all posible paths to a node before this node is reached. This situation makes any adjustment of marks nonsensical. Furthermore, it may be possible that the student could arrive at a particular place on the network having answered a relatively large number of questions at a low level and a very small number of questions at a high level. It is also possible that very few of these questions will be exactly the same as the questions asked of students who have proceeded through the network at a different level. Because the students have not been asked the same questions it could well be argued with some considerable force that the same things have not been tested and thus comparison is invidious. To overcome this difficulty it has been decided to adopt two different classifications of marks and questions. We will call these two classes **local** and **global**.

Questions of a **local** type will be used purely for the diagnosis of faults and the prescription of remedial work whereas questions of a **global** type will be given to all students who have studied a particular set of educational objectives and will be used only in prescribing the best future course for the student's progress forward through the network towards the educational objectives that form the goal of the network. Thus because all students arriving at a particular node will have been asked a varying subset of questions from the global list, it will be possible to make comparisons based upon the intersection of all the subsets which will have been achieved at that node. It can of course be argued that a subset of questions would probably have existed under the system proposed in the preceding paragraph. We feel however that by formalising the system we can be sure that the subset can be defined to meet specific requirements. Further implications of the problems here have been investigated and will be reported upon later.

6. SUMMARY

Our philosophy in this research is to devise a system which will be as acceptable to the teachers in the classroom as possible and also which will be capable of being implemented on a wide range of computer systems. The programs used in this system therefore are largely being written in the language ALGOL and our design of educational material is being done in the closest possible cooperation with a group of teachers from all types of secondary schools. We are endeavouring not to impose unnecessary constraints upon the text writer either for administrative or technical reasons and yet we are endeavouring to produce a system in which the teacher as well as the student will have a high level of confidence.

REFERENCES

DORAN, J., & MICHIE, D. (1966). Experiments with the Graph Traverser program. Proc. R. Soc., (A), 294 235-259.

BLOOM, B.S., Ed (1956) Taxonomy of Educational Objectives; Volume I: Cognitive Domain, New York: McKay.

A LOW-COST EXPERIMENT IN COMPUTER AIDED INSTRUCTION

Gordon Michael Bull

1. INTRODUCTION

Using a teletype and a Kodak Carousel Slide Projector, a simple computer aided instruction system has been made available on a slow second generation machine. Although by no means a system suitable for large scale class use it has enabled us to experiment with CAI and with CAI languages.

Two items of software have been written. Firstly, a language has been designed to facilitate the writing of programs and a compiler written for this language. Secondly, a monitoring system to record the progress of individuals through a number of CAI programs. The instruction programs are regarded as a suite or series of programs through which various students are in general progressing in a predetermined order.

2. PROGRAMMING LANGUAGE

The object of the language is to simplify the job of writing CAI programs for this system. There are eleven statements in the language, developed as a result of past experience in writing such programs. Single letter abbreviations may be used for all statements. Bearing in mind the monitoring system, the language allows the writer to include suitable restart points for students who do not complete the entire program at one sitting, but wish to stop and return at some later date. The compiler provides output suitable for input under the monitoring system.

Each statement must start on a new line, but may occupy more than one line.

2.1 *Display*

This takes the form

DISPLAY SLIDE NUMBER b

Where b is an unsigned integer in the range 1 to 80. When obeyed, this causes the slide in position b of the slide tray to be displayed. There are eighty positions in the tray and it may be rotated both clockwise and counter clockwise. The tray will be rotated in whichever direction is the shortest from its current position to position b.

2.2 *Type and Quotes*

This takes the form

TYPE @ any string of characters excluding an at sign. The string may extend over many lines as this one does and is terminated by an at sign thus @.

The statement, when obeyed causes the string, exactly as typed but excluding the @ signs to be output on the teletype.

It is possible to change the string quotes to any character of the programmers choice by use of the statement QUOTE X. This will change the type statement to the form:

TYPE X String X.

The string quotes may be changed at any point in the program and any number of times. If no QUOTE statement occurs @ is assumed as the delimiter.

2.3 *Goto*

This takes the form

GOTO label

The statement causes a break in the normal sequence of obeying statements and transfers to the statement labelled and continues from there. A label is an unsigned integer. Any statement may have at most one label and a label may only appear as a label to one statement. The label is written in front of a statement and is separated from it by a colon.

A more general form of the Goto statement is

GOTO label AFTER (N) TIMES GOTO label.

Where N is an unsigned integer. This statement is used to stop a student from getting stuck in a loop by repeatedly failing to give the correct answer. It is best illustrated by an example.

```
10:TYPE @ WHAT IS THE ANSWER @
   IF @ A@70 @ B @ 80, 90
90:TYPE @ DO YOU WISH TO TRY AGAIN-YES OR
   NO? @
   IF @ YES @ 10, 20.
```

If the student repeatedly fails to respond with either A or B and insists on a retry, he is stuck in a self-made loop and wasting time. What is required is an upper limit on the number of retries allowed – say 3 in this case.

To accomplish this the final statement becomes

IF @ YES @ GOTO 10 AFTER (3) TIMES GOTO 20,GOTO 20.

This may be abbreviated by omitting AFTER and TIMES GOTO giving

IF @ YES @ 10(3) 20, 20.

2.4 *If*

The general form of this statement is

IF REPLY IS @ string @ GOTO label @ string @ GOTO label ..... @ string @ GOTO label, OTHERWISE GOTO label.

Just prior to an IF statement at run-time, input is requested from the teletype. The input string is compared with each of the strings in the if statement. If a match is found, control is transferred using the GOTO statement following the matched string. If no match is found, control is transferred to the label following OTHERWISE.

Abbreviation: all GOTO's may be omitted as may OTHERWISE.

e.g. IF @ P @ 10 @ Q @ 20, 30

2.5 *Right*

This statement is used to record a correct response and

causes the count associated with correct answers to be increased by one. This count together with the number of incorrect responses gives a measure of the students performance. The statement takes the form

RIGHT

2.6 *Wrong*

This is similar to RIGHT and is used to record the number of incorrect responses. It takes the form

WRONG

2.7 *Comment*

This statement takes the form

COMMENT @ any sequence of characters @

This statement is provided to allow the program writer to include a commentary. It is ignored by the compiler and thus has no effect on the program.

2.8 *Any*

The simple form of this statement is

ANY @ string 1 @ @ string 20 ... @ string P@ GOTO N, OTHERWISE GOTO M.

Just prior to an ANY statement at run-time, input is requested from the teletype. The input string is compared with each string in turn. If a match is found, control is transfered to statement labelled N, otherwise to statement labelled M.

In this form the statement is equivalent to

IF @ string 1@ N @string 20 N .... @string P@ N, M.

A more general form of the statement is

ANY (N) FROM @ string 1@ string 2@ ... @ string P@ GOTO S, OTHERWISE GOTO T.

In this case N sequential requests for input are made at run-time.

If the N responses are all different and correct (i.e. in the list) then control is transfered to S. Any duplicate or incorrect responses are marked with an asterisk and after N responses control is transfered to T.

This statement allows the analysis of N replies in random order taken from a possible P correct answers.

P must be greater than or equal to N.

Thus, for example, to analyse the response to the question 'name three vowels' one might write in abbreviated form:

ANY (3) @A@@E@@I@@O@@U@100,200

The most general form of the statement is

ANY (N,M) FROM @ string @...@string P@ GOTOS, OR GOTO Q OTHERWISE GOTO T.

In this case a minimum of N and a maximum of M sequential requests for input are made at run time. If the first N response are correct and different then control is transfered to S. If a response is incorrect it is marked with an asterisk. For each incorrect response a further request for input is given up to a total maximum of M requests. If N correct replies are obtained in greater than N responses, control is transfered to Q. If less than N correct responses are obtained,control is transfered to T.

e.g.Type @ WHICH THREE PLANETS ARE CLOSEST TO EARTH @ ANY (3,5) @ MERCURY @ @ MARS @ VENUS @ 100,200,300.

At run time we might get the following dialoque;

? MOON *
? MARS
? VENUS
? SATURN *
? PLUTO *

The special form 'ANY n' will GOTO n following any input whatsoever.

2.9 *Use & Back*

This statement takes the form

USE Label

and causes a temporary transfer to the labelled statement. Statements following the labelled statement are obeyed until a back statement is met, at which point control returns to the use statement and the program continues on from there. This statement is used for repeated sections of code. For example,if the responses YES or NO are required frequently then we might have typically:

```
IF @ YES @ n @ NO @ m, P
   :
P: USE Q
   :
   :
Q: WRONG
   TYPE @ YOU MUST RESPOND WITH EITHER
   YES OR NO--TRY AGAIN
   BACK
```

2.10 *Finish*

This statement is the last line of a program.

3. ENTRY POINTS

When a student is running a program, he may interrupt by depressing any key on the teletype at a time when input is not expected of the student.
This will cause the message

DO YOU WISH TO STOP?

to be displayed. If the response is NO, then the program will continue as if nothing had happened. If the response is YES then the values of RIGHT and WRONG are output and control returns to the monitoring system (see later). If at a later time the student wishes to continue with the programme he will be restarted at the latest ENTRY point passed through. An entry point is indicated by prefixing the appropriate statement with

ENTRY: or E:

The entry prefix must follow by a label for the statement it prefixed

e.g. 10: E: statement.

The entry prefix causes the address of the object code for that statement to be planted at run-time into a standard location which then forms part of the student's information held by the monitoring system.

4. EXAMPLE

This example is an adaptation of a teaching program which appears in the book 'Conditions for the Existance of Life'' published by ESL, Bristol.

```
DISPLAY 1.
5: IF @ AIR @ 1 @ WATER @ 2 @ FOOD @ 3,4.
1: TYPE @ YES, BUT THAT IS NOT THE REPLY I
WANTED. WHAT DOES THE PICTURE SUGGEST? TRY
AGAIN @
WRONG
GOTO 5 (1) 6.
2: TYPE @ YES, BUT I AM AFTER SOMETHING ELSE.
```

```
WHAT DOES THE PICTURE SUGGEST OTHER THAN WATER? TRY AGAIN @
WRONG
GOTO 5 (1) 6.
4: TYPE @ NO, THERE ARE 3 OBVIOUS ESSENTIALS FOOD, WATER AND AIR. WHICH DOES THE PICTURE SUGGEST @
WRONG
GOTO 5
6: TYPE @ THE ANSWER I AM LOOKING FOR IS FOOD @
W
GOTO 7
3: TYPE @ CORRECT @
R
7: DISPLAY 2
T @ SOME FOOD CAN BE PLACED IN MORE THAN ONE GROUP, FISH FOR INSTANCE CONTAIN FAT AND @
11: IF @ CARBOHYDRATE @ 8 @ PROTIEN @ 9,10
8: @ LOOK AT THE PICTURE AGAIN @
W
GOTO 11 (1) 12
10: @ THERE ARE ONLY TWO OTHER GROUPS CARBOHYDRATES AND PROTIEN. TRY AGAIN @
W
GOTO 11 (1) 12
12: T @ FISH APPEAR IN THE PICTURES FOR FAT AND PROTIEN SO THE ANSWER IS PROTIEN@
W
GOTO 150
9: T @ CORRECT @
R
150: DISPLAY 3
15: IF @ ENERGY @ 13,14
14: T @ READ THE SLIDE AGAIN AND HAVE ANOTHER TRY @
W
GO 15 (1) 16
16: T @ THE ANSWER IS ENERGY @
W
GO 17
13: T @ CORRECT @
R
17: D 4
T @ FROM THE FIGURES BESIDE THE PICTURES YOU CAN TELL THAT OUR DIET CONSISTS SOLELY/PARTLY/MAINLY OF FOOD @
21: IF @ SOLELY @ 18 @ MAINLY @ 18 @ PARTLY @ 19,20
20: T @ RESPOND WITH SOLELY, MAINLY OR PARTLY @
W
GOTO 21 (1) 22
22: @ THE ANSWER IS PARTLY @
W
GO 23
18: T @ HARDLY: LOOK AGAIN AT BANANAS. TRY AGAIN @
W
GO 21 (1) 22
19: T @ GOOD @
R
23: D 5
T @ SO FAR THEN WE KNOW THAT IN THINGS WE EAT THERE ARE 3 KINDS OF FOODS: CARBOHYDRATES, FAT AND PROTIEN AS WELL AS A GOOD DEAL OF ---------@
IF @ WATER 24,25
25: T @ THE ONLY OTHER THING WE HAVE MENTIONED IS WATER @
W
GO 26
24: T @ FINE @
R
26: DISPLAY 6
31: IF @ LARGE @ 27 @ SMALL @ 28,29
29: T @ RESPOND WITH LARGE OR SMALL @
W
GO 31 (1) 130
130: T @ THERE IS ONLY 0.8% LEFT IN FACT @
GO 30
27: T @ YOUR ADDITION NEED SOME ATTENTION. I MAKE THE SUM 99.2% SO THERE IS NOT MUCH MISSING @
W
GO 130
28: TO @ CORRECT – ONLY 0.8% LEFT IN FACT @
R
30: DISPLAY 7
ANY @ VITAMIN @ @ VITAMINS @ 32,33
33: T @ THE ANSWER IS VITAMIN. ORANGES GIVE US THE VITAMIN C FOR EXAMPLE @
W
GO 34
32: T @ GOOD @
R
34: T @ READ THE NEXT SLIDE AND GIVE SIX ANSWERS ONE EACH TIME A? APPEARS @
DISPLAY 8
39: ANY (3) @ CARBOHYDRATE @@ PROTIEN @ @ FAT @ 37,38
38: T @ THE FIRST THREE ARE FOODS @
W
D 2
T @ REMEMBER NOW @
ANY 36
36: DISPLAY 8
T @ TRY AGAIN @
GO 39 (1) 40
40: T @ THE FOODS ARE CARBOHYDRATES, PROTIEN AND FAT @
W
37: ANY (3) @ WATER @@ VITAMIN @ @ VITAMINS @ @ MINERALS @ 41,42
42: T @ WATER, MINERALS, VITAMINS @
W
GOTO 43
41: T @ CORRECT @
R
43: FIN
```

5. MONITORING SYSTEM

The monitoring system is designed to monitor the running of CAI programmes for a number of students.

5.1 *Student Identification*

On entry to the system a student goes through a login sequence which demands a user name and a password. The password is marked by typing it into a masked area ;

e.g.? LOGIN user name
? &&&&&&
If the user fails to login the following output occurs.
USER NAME OR PASSWORD FAILURE
IF YOU HAVE NO USER NAME
USE THE USER NAME 'CASUAL'
?

Users using the name 'casual' do not get a password test and no information is kept after the run. This avoids the system becoming clogged with people who use the system once only and whose statistics etc., are kept by the system.

On leaving the system each user should type

LOGOUT

Following logout the system cycles waiting for a login sequence. The output following the login sequence is discussed in the section on restart points.

5.2 *Statistics*

A number of items are recorded about the overall use of the system. These are as follows. A count on the usage of each program on the system together with the total number of errors made. A count on the number of logins. Information is also gathered for each student as follows:

A list of all the programs used. A count of the number of errors made on each program. The total number of errors made. A count on the number of times logged in.

The statistics may be output by the librarian (any person logging in under the name SYSTEM and passing the password test) by typing STATS in response to the question JOB. (Note on logging in under SYSTEM one is not given the opportunity to use the system only to output the statistics or update the library). Following the output of the statistics the question JOB? is re-output to allow the library to be changed if required. The response NONE to the job question causes an exit equivalent to logging out. The output of statistics can be terminated by use of the interrupt.

5.3 *Restart points and Library*

Using the interrupt facility on the teletype it is possible for a student to terminate a program at any time other than when input is awaited from the teletype (i.e. it is not possible to interrupt whilst the teletype is switched to input mode). The system will record against that student the restart point ready for the next session by that student.
Following student identification at login, the facility exists for the student to restart at the appropriate restart point (which may be the start of a new program if the student left the system having completed a program) or to start a new program (or restart the current one). Thus the following dialogue might occur.

? LOGIN GMBULL
OLD OR NEW? OLD
program title
output from last restart point passed at last session.
or alternatively
? LOGIN GMBULL
OLD OR NEW? NEW
LIBRARY? NO
SECTION NAME? GEOGRAPHY
PROGRAM NAME? GLACIER
title
output from GLACIER
or alternatively
? LOGIN GMBULL
OLD OR NEW? NEW
LIBRARY? YES
list of sections in the library by name
SECTION NAME? GEOGRAPHY
list of programs by name in this section, each name is followed by a brief description of the program
LIBRARY? NO
SECTION NAME? GEOGRAPHY
PROGRAM NAME? GLACIER
title
output from GLACIER

The ability to change programs, interrogate the library and obtain statistics occurs at the end of a program at which point the dialogue is

END OF PROGRAM
STATISTICS? NO
CONTINUE OR BREAK? CONTINUE
Title of next program
output from program
or alternatively
END OF PROGRAM
STATISTICS? NO
CONTINUE OR BREAK? BREAK
LIBRARY?
etc., as in login
or alternatively
END OF PROGRAM
STATISTICS? YES
MINIMUM NUMBER OF QUESTIONS POSSIBLE – X
QUESTIONS ANSWERED – Y
RIGHT – P
WRONG – Q
CONTINUE OR BREAK?
etc.,

If a student opts from a new program then any restart point held is lost and is replaced by the start of the new program.

EVALUATION OF COMPUTER HARDWARE AND SOFTWARE IN A COMPUTER ASSISTED INSTRUCTIONAL ENVIRONMENT

Daniel Burdett and Arnold Schildkret

1. EVALUATION OF COMPUTER HARDWARE IN A CAI ENVIRONMENT

1.1 *Hardware system requirements*

In a real-time system which requires immediate responses, such as the HOSTOS COMMUNITY COLLEGE CAI SYSTEM, failure for extended periods of time cannot be tolerated. To reduce the likelihood of such failures, the technique of hardware system interchange via ATT Telephone lines, and equipment redundancy, will be utilized along with a procedure for switching to the back-up equipment. When one piece of computer equipment ceases to function, a second one should take over the functions, either by means of system interchange, or automatic pre-programmed equipment redundancy takeover.

The HOSTOS COMMUNITY COLLEGE requirement for recovery in case of malfunction of the CAI-CMI system (hardware, software, or applications) is that the malfunctioning system or a back-up system **must** be completely operational within 30 minutes after the initial malfunction. Device independency of software, ease of interchangeability of peripherals, and ease of systems interchange with another similar user configuration are a necessity in any proposed system. In addition, checkpoints in the CAI-CMI on-line system should be taken every three minutes. Recovery should prevent loss of data or specify the data loss. A procedure for recovery should be described with every system proposed. Included in such a procedure should be a planned degraded mode of operation if the number of peripherals available to the central site CPU system falls below the minimum required for normal operation. There is no requirement that back-up equipment be on-site at HOSTOS. The only requirements are stated above.

Because of the nature of the HOSTOS COMMUNITY COLLEGE CAI-CMI System, and the responsibility which HOSTOS must assume in the operation of the HOSTOS CAI-CMI System for its students, the recovery requirements mentioned previously must be strictly adhered to in the design of the computer system. This responsibility includes the consideration that all of the HOSTOS CAI terminals should have interface message switching capability to another user's computer system, in a interchange processor mode. This means that all HOSTOS CAI-CMI/WIS files should be established permanently in the back-up user's site. It should be noted that the student terminals must have integrity in the ability to maintain on-line CAI Education, and that this factor **must** be taken into account in the proposed vendor recovery plan.

1.2 *Computer hardware system comparisons*

To compare the performance and capabilities of proposed computer systems, it is necessary to objectively evaluate the effectiveness of the Central Processing Unit speed efficiency, the CPU Instruction/Mix, the I/O capabilities, and the Telecommunication efficiency. In the following algorithm, the lower the value calculated for each element, the more efficient the element.

Consider now the cycle time of the IBM 360-75, of .75 microseconds per two byte fetch using 2 or 4 way interleaving. The Univac 1108 has a cycle time of .75 microseconds per word fetch, with a minimum of 2-way parallel multi-processing; and a capability of 4-way parallel processing. The following formulas will permit the calculation of the effectiveness of the proposed mix of any third generation computer system. For a somewhat different outlook on hardware evaluation the reader is directed to the IBM Systems Journal, Volume Eight, Number Four, 1969; „A perspective on systems performance evaluation" by M.E. Drummond, Jr.

The following formulas were developed by A. Schilkret. The formulas are:

1.2.1. Processor Speed Efficiency

$$Pe_1 = \left[\frac{1}{(\text{Word Size in Bits}) \times (\text{No. of Words Accessed Cycle})} \times (\text{Cycle Time in Microseconds})\right] \times \left[\frac{1}{(\text{No. of ways of storage interleaving factor per individual Central Processing Unit})}\right] \times 1000$$

NOTE: If Storage Interleaving is none, Storage Interleaving Factor is "1."

The above formula is modified for a multi-processor, parallel storage system.

$$Pe_1 = \left[\frac{1}{(\text{Word Size in Bits}) \times (\text{No. of Words Accessed Cycle})} \times (\text{Cycle Time in Microseconds})\right] \times .50 \times \left[\frac{1}{(\text{No. of ways of Parallel Processing per individual Central Processing Unit})}\right] \times 1000$$

NOTE: To compare character or byte oriented computers to word oriented computers, convert the word to characters. Compare unpacked bytes to unpacked characters.

1.2.2. Processor Instruction Mix Efficiency (all times in Microseconds)

$$Pe_2 = \left[\frac{(\text{Add Time}) + (\text{Divide Time}) + (\text{Multiply Time})}{3}\right] \div \left[10\right]$$

NOTE: Assumes 5 Digits to 5 Digits Field, fixed point, single precision.

1.2.3. Tape Processor I/O Efficiency (Transfer Rate on K Characters to per second)

$$Pe_2 = \left[\frac{100{,}000}{(\text{Tape Speed Transfer Rate in K Bytes/Sec.}) \times (\text{Tape Density})}\right]$$

1.2.4. Disk/Drum Processor Efficiency (Transfer Rate in K Characters to per second)

$$Pe_4 = \left[\left(\frac{100}{\text{(Disk or Drum Transfer Rate in K Bytes/Sec)}}\right) \times \left(\frac{\text{Average (random access time) in Milliseconds}}{100}\right)\right] \times 10$$

1.2.5. Reader/Punch/Printer Efficiency

$$Pe_5 = \left[\frac{1000}{\dfrac{(\text{Reader Speed in CPM}) + (\text{Punch Speed in CPM}) + \dfrac{(\text{Printer Speed in LPM})}{3}}}\right]$$

NOTE: Card Reading is for 80 column card.
Card Punching is for 80 column card.
Printing Speed is 132 sector, alphanumeric print line.

1.2.6. Telecommunication Efficiency

$$Pe_6 = \left[\left(\frac{\text{Average Response Time/ Per Terminal in Seconds}}{}\right) \times \left(\frac{\text{Memory in K required per Terminal to support Terminal W/S in Core Communication W/S}}{}\right)\right] \div 100$$

NOTE: A 2400 baud line was assumed in this text. Core requirements are not converted to bits because character fetch length varies from a character to a word processor.

1.2.7. Computer System Efficiency (CSE)

$$CSE = \left[Pe_1\right] + \left[Pe_2\right] + \left[Pe_3\right] + \left[Pe_4\right] + \left[Pe_5\right] + \left[Pe_6\right]$$

Figure 1

HARDWARE COST/PERFORMANCE SUMMARY CHART

(The lower the total score, the higher the efficiency)

Comparative Characteristics for 161 Terminals

Characteristic	615	RCA 70/45G	RCA 70/60N	H-3200	SIGMA-7	NCR-200	Digital PDP-10	Hewlett Packard HP-2116B	IBM 360-67 [4]	IBM 360/75	Univac 1108
Processor Speed Efficienct (Pe_1)	5.40	90.00	12.00	52.77	12.50	50.00	15.25	100.00	5.85	5.85	5.19
Processor Instruction Mix Efficiency (Pe_2)	1.70	8.92	1.31	33.03	2.31	45.90	.98	1.44	.67	1.37	.57
Tape Processor I/O Efficiency (Pe_3)	2.08 (MTH-301)	1.04 (70/445)	1.04 (70/445)	.86 (204B-9)	2.08 (7322)	1.56 (633-121)	3.47 (TM-20)	1.30 (7970)	1.00 (2401-5)	1.00 (2401-5)	1.79 (UniservoVIII-C)
Disk/Drum Efficiency (Pe_4)	7.88 (DSU-270)	6.41 (70/564)	6.41 (70/564)	1.20 (278)	.06 (7212)	6.51 (655-201)	2.23 (RD-10)	.53 (HP-2771A)	3.15 (2314)	3.15 (2314)	1.92 (9414)
Reader/Punch/Printer Efficiency (Pe_5)	1.25	1.14	1.14	1.18	2.31	1.59	1.53	1.99	2.00	2.00	1.25
Telecommunication Efficiency (Pe_6)	.33[1] (GECOS III)	6.70	6.70	17.88	6.33	12.36[3]	7.68[2]	24.00	6.70	6.70	6.33
Computer System Efficiency	18.64	114.21	28.60	106.92	25.59	106.92	31.14	129.26	19.37	20.07	17.05

NOTES

1) The GECOS III operating system overlays communications core workstorage extensively. GECOS III was designed for a Fortran-based problem solving environment. It would be questionable whether this type of operating system would support a large CAI Installation.

2) This manufacturer does not currently provide CAI software.

3) This manufacturer does not provide CAI software, nor an author language.

4) Does not include calculations for degraded use - used TSS as an operating system. System degradation can be expected to be 31½% for a 161 Terminal System, or near 40% for a 900 Terminal System. Refer to Auerbach IBM System 360-Model 67, page 427:201-101.

2. DETAIL FORMULA ANALYSIS OF EQUIPMENT PERFORMANCE

A. G.E. 615

1. Processor Speed Efficiency

Word Size = 37 bits, Cycle time = 2 microseconds

$$Pe_1 = \left[\frac{1}{(37)\ (1)} \times (2)\right] \times \left[\frac{1}{1}\right] \times 1000$$

$$= \left[\left(\frac{1}{37}\right)(2)\right] \times 1000$$

$$= \left[\frac{2}{37}\right] \times 1000$$

$$= .0054 \times 1000$$

$$= 5.4$$

---

2. Processor Instruction Mix Efficiency

Pe_2 = ADA 4.0

FMP = 16.2
Floating Multiply

FDV = 31.0
Floating Divide

$$Pe_2 = \left[\frac{(4.0) + (16.2) + (31.0)}{3}\right] \div 10$$

$$= \left[\frac{51.2}{3}\right] \div 10$$

$$= 17.01 \div 10$$

$$= 1.70$$

---

3. Tape Processor I/O Efficiency

MTH-301 7 Track Tapes

$$Pe_3 = \frac{100,000}{(60KC) \times (800\ BP_1)}$$

$$= \frac{100,000}{48,000}$$

$$= 2.08$$

4. Disk/Drum Efficiency

DSU-270

$$Pe_4 = \left[\left(\frac{100}{330\ KB}\right)\left(\frac{26}{100}\right)\right] \times 10$$

$$= \left[(.303)\ (.26)\right] \times 10$$

$$= (.07878) \times 10$$

$$= 7.88$$

---

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \frac{1000}{\frac{(900) + (300) + (1200)}{3}}$$

$$= \frac{1000}{800}$$

$$= 1.25$$

---

6. Telecommunication Efficiency

$$Pe_6 = \left[3\right]\left[11\right] \div 100 = .33$$

Note: GECOS III operating system overlays communication core workstage extensively. GECOS III was designed for a Fortran-based problem solving environment (e.g. Dartmouth College). It would be questionable whether the GECOS III operating system would not cause degradation of response time beyond three seconds in a much larger terminal based CAI system than the 161 terminals it used in this example.

B. G.E. 635

A Two-Way Parallel Processor

1. Processor Speed Efficiency

$$Pe_1 = \left[\left(\frac{1}{(36)\ (2)}\ (1)\right) \times \left(.50\tfrac{1}{2}\right)\right] \times 1000$$

$$= \left[\left(\frac{1}{72}\right) \times (.25)\right] \times 1000$$

$$= \left[.0138 \times (.25)\right] \times 1000$$

$$= \left[.00345\right] \times \left[1000\right]$$

$$= 3.45$$

2. Processor Instruction Mix Efficiency

$$Pe_2 = \left[\frac{(1.9) + (6.5) + (6.5)}{3}\right] \div \left[10\right]$$

ADA = Add to A for Add Instruction
FDV = Floating Divide Single Precision for Divide Inst.
FMP = Floating Multiply Single Precision for Multiply.

$$= \left[\frac{14.9}{3}\right] \div \left[10\right]$$

$$= 4.76$$

$$= 4.76$$

---

3. Tape Processor I/O Efficiency

$$Pe_3 = \left[\frac{100{,}000}{(160\ KB) \times (800)}\right]$$

$$= \frac{100{,}000}{128{,}000}$$

$$= .781$$

4.A. Disk/Drum Efficiency

$$Pe_4 = \left[\left(\frac{100}{\text{Disk or Drum Transfer Rate in K Bytes/Sec.}}\right) \times \left(\frac{\text{Average Access Time in Millisecs.}}{100}\right)\right]$$

For G.E. - MDS 200/201 Magnetic Drum Subsystem.

$$Pe_4 = \left[\left(\frac{100}{370\ KB}\right) \times \left(\frac{18}{100}\right)\right] \times 10$$

The higher the value, the better the transfer rate. Use reciprocal value.

The lower the value the access time is, the better this characteristic. Use regular value divided by 100.

$$= \left[\left(.27\right) \times \left(.18\right)\right] \times 10$$

$$= (.0486) \times 10$$

$$= .49$$

4.B. G.E. - DSS-160
Disk Storage

$$Pe_4 = \left[\left(\frac{100}{208}\right) \times \left(\frac{87.5}{100}\right)\right] \times 10$$

$$= \left[(.48) \quad (.875)\right] \times 10$$

$$= .420$$

$$= 4.20$$

---

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \left[\frac{1000}{(900) + (300) + (1200)}\right]$$

CPZ 201 Card Reader
CPZ 201 Card Punch
PRT 201 Printer Subsystem

$$= \frac{1000}{2400}$$

$$= .41$$

---

6. Telecommunication Efficiency

Based upon a proposal submitted by the General Electric Company by James J. Byrne, to Hostos Community College, dated February 16, 1970, it has been extrapolated that 11K of 36 bit G.E.-635 core would be required to handle 161 terminals.

$$Pe_6 = \left[(3) \times (11)\right] \div 100$$

$$= (33) \div 100$$

$$= .33$$

Note: The GECOS III operating system overlays communication core workstorage extensively. GECOS III was designed for a Fortran-based problem-solving environment (e.g. Dartmouth College). It would be questionable whether the GECOS III operating system would not cause degradation of response time beyond three seconds in a much larger terminal-based CAI system than the 161 terminal system used in this example.

C. RCA SPECTRA 70-45-G

1. Processor Speed Efficiency

$$Pe_1 = \left[\frac{1}{(8)\ (2)}\ (1.44)\right] \times \frac{1}{1} \times 1000$$

$$= \left[\frac{1}{16} \times 1.44\right] \times 1000$$

$$= \left[\frac{1.44}{16}\right] \times 1000$$

$$= (.09) \times 1000$$

$$= 90$$

---

2. Processor Instruction Mix Efficiency (5 digits to 5 digits)
(Refer to Auerbach, Page 715 = 051 100)

Add Decimal = 15.36 + 1.11 (5)
= 15.36 + 5.55
= 20.91

Multiply Decimal = $28.49 + \left[9.52\ (5)\right] + \left[.59\ (5)^2\right]$
= 28.49 + 47.60 + 14.75
= 90.84

Divide Decimal = $26.33 + \left[19.14\ (5)\right] + \left[1.35\ (5)^2\right]$
= 26.33 + 95.70 + 33.75
= 155.78

$$Pe_2 = \left[\frac{20.91 + 90.84 + 155.78}{3}\right] \div 10$$

$$= 89.17 \div 10$$

$$= 8.92$$

---

3. Tape Processor I/O Efficiency

$$Pe_3 = \frac{100{,}000}{(60\ KB) \times (1600)}$$

$$= \frac{100{,}000}{96{,}000}$$

$$= 1.04$$

---

4. Disk/Drum Efficiency

$$Pe_4 = \left[\left(\frac{100}{156K}\right) \times \left(\frac{75}{100}\right)\right] \times 10$$

$$= .641 \times 10$$

$$= 6.41$$

---

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \left[\frac{1000}{\frac{(1435 + 100 + 1100)}{3}}\right]$$

$$= 1.14$$

Note: Above speeds are peak speeds.

---

6. Telecommunication Efficiency

The IBM-360 Formula for Coursewriter III was used in lieu of a formula not provided by RCA. Approximately 223,353 bytes of core would be required for a 161 terminal system.

$$Pe_6 = \left[(3) \times (223)\right] \div 100 = 6.7$$

D. RCA SPECTRA 70/60 N

Processor efficiency calculation assumes 131,072 byte module of each 263,144 byte bank of main memory is "shaded memory".[1] Shaded memory is basically comparable to 2-way interleaving on IBM's 360 Model 75. Because the RCA processor concept is still a traditional single processor CPU, interleaving is effectively to 2-way, even though there would be five (5) shaded memories in the proposed 1 million byte processor. To have more than 2-way effective interleaving, five input-output control logic units would be required. Refer to G.E.-600 family for this electronic concept of computer hardware.[2]

---

1. Processor Speed Efficiency

$$Pe_1 = \left\langle\left[\frac{1}{(8)\ (4)}\right] \times (.765)\right] \times \frac{1}{2}\right\rangle \times 1000$$

$$= .024 \times \tfrac{1}{2}$$

$$= (.012) \times 1000$$

$$= 12.000$$

$$= 12.0$$

NOTES:

1 Refer to Auerbach, Page 718:011.100

2 G.E.-615/635 Information Systems Manual CPB-371F

---

2. Processor Instruction Mix Efficiency (5 Decimal Digits)

(Refer to Auerbach, Page 718:051.100 Fixed Point, Single Precision)

Add Decimal = 9.18 + 0.25 D
= 9.18 + 1.25
= 10.368

Multiply Decimal = 11.51 + 0.29 D
= 11.51 + 1.45
= 12.96

Divide Decimal = 14.73 + 0.29 D
= 15.98

$$Pe_2 = \left[\frac{(10.37) + (15.98) + (12.96)}{3}\right] \div 10$$

$$= \left[\frac{39.31}{3}\right] \div 10$$

$$= (13.10) \div 10$$

$$= 1.31$$

---

3. Tape Processor I/O Efficiency

$$Pe_3 = \frac{100.000}{(60\ KB) \times (1600)}$$

$$= 1.04$$

---

4. Disk/Drum Efficiency 70/590-4

$$Pe_4 = \left[\left(\frac{100}{156K}\right) \times \left(\frac{75}{100}\right)\right] \times 10$$

$$= 6.41$$

---

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \left[\frac{1000}{1435 + 100 + 1100}\right]$$

$$= 1.14$$

Note: Above speeds are peak speeds. Formula for calculating average peripheral speeds was not available.

---

6. Telecommunication Efficiency

The IBM-360 Formula for Coursewriter III was used in lieu of a formula not provided by RCA. Approximately 223,353 bytes of core would be required for a 161 terminal system.

$$Pe_6 = \left[(3) \times (223)\right] \div 100 = 6.7$$

E. HONEYWELL 3200

1. Processor Speed Efficiency

The H-3200 character is a 9-bit character consisting of 6 data bits, 2 punctuation bits, and one parity bit used by the automatic checking circuitry.[1]

$$Pe_1 = \left[\left(\frac{1}{(9)\ (2)}\right) \times (1.0)\right] \times 1000$$

$$= \left[\frac{1}{.18}\right] \times 1000$$

$$= 52.77$$

---

2. Processor Instruction Mix Efficiency

Decimal Add - A/A,B = 14
Fixed Point

Decimal Multiply - M/A,B = 91

Decimal Divide - D/A,B = 286[2]

$$Pe_2 = \left[\frac{14 + 286 + 91}{3}\right] \div 10$$

$$= (330.3) \div 10$$

$$= 33.03$$

---

3. Tape Processor I/O Efficiency 204B-9 Tape Drive

$$Pe_3 = \frac{100,000}{(96\ KC) \times (1200)}$$ (3)

$$= \frac{100,000}{115,200}$$

$$= .86$$

---

4. Disk/Drum Efficiency

$$Pe_4 = \left[\left(\frac{100}{416\ KC}\right) \times \left(\frac{50}{100}\right)\right] \times 10$$ (4)

$$= \left[(.2404) \times .50\right] \times 10$$

$$= (.1202) \times 10$$

$$= 1.20$$

NOTES:

1 Honeywell Series 200, Model 3200 - Summary Description, Page 10

2 Ibid, Page 45

(3) Ibid, Honeywell Series 200; Model 200 Summary Description, Page 25.

(4) Honeywell Hardware Bulletin #111.0005 1300.1 A81, dated January, 1969. Average Seek Time is 50 milliseconds.

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \left[\frac{1000}{\frac{1100 \text{ LPM} + 1050 \text{ CPM} + 400 \text{ CPM Punch Reader}}{3}} \right]$$

$$= \frac{1000}{\frac{2550}{3}}$$

$$= \frac{1000}{850}$$

$$= 1.176$$

$$= 1.18$$

---

6. Telecommunication Efficiency

$$Pe_6 = \left[(3) \times (596) \right] \div 100$$

$$= (1788) \div 100$$

$$= 17.88$$

F. SDS - SIGMA 7

1. Processor Speed Efficiency

Three-way access was proposed. This provides a 3-way memory bus structure. The effect of this electronically is the creation of a 3-way interleaved processor.

$$Pe_1 = \left\langle \left[\left(\frac{1}{(32)\ (1)} \right) \times (1.2) \right] \times \left[\frac{1}{3} \right] \right\rangle \times 1000$$

$$= .0375 \times \frac{1}{3} \times 1000$$

$$= .0125 \times 1000$$

$$= 12.50$$

---

2. Processor Instruction Mix Efficiency [1]

Decimal Add = 17.4 + 0.4 D

= 17.4 + 2.0

= 19.4

D = Number of Decimal Digits

Decimal Multiply / DM = 19.3 + 0.4 Mm

= 19.3 + (0.4) (5 x 5)

= 19.3 + 10

= 29.3

M = Number of digits in Multiplicand

m = Number of digits in Multiplier

Decimal Divide / (DD) = 19.3 + 0.5 VQ

V = Number of digits in divisor including sign

Q = Number of digits in quotient

= 19.3 + 0.5 (5 x 5)

= 19.3 + .5 (25)

= 19.3 + 1.25

= 20.55

$$Pe_2 = \left[\frac{19.4 + 29.3 + 20.55}{3} \right] \div 10$$

$$= \left[\frac{69.25}{3} \right] \div 10$$

$$= \left[23.08 \right] \div 10$$

$$= 2.31$$

---

3. Tape Processor I/O Efficiency

7322 Magnetic Tape Unit

$$Pe_3 = \left[\frac{100.000}{(60 \text{ KB}) \times (800)} \right]$$

$$= \frac{100.000}{48.000}$$

$$= 2.08$$

---

4. Disk/Drum Efficiency 7212 Fixed Head Storage Unit

$$Pe_4 = \left[\left(\frac{100}{3{,}000 \text{ KC}} \right) \times \left(\frac{17}{100} \right) \right] \times 10$$

$$= \left[(.033) \times (.17) \right] \times 10$$

$$= \left[.00561 \right] \times 10$$

$$= .056$$

---

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \frac{1000}{\text{400 CPM Reader} + \text{600 CPM Printer} + \text{300 CPM Punch}}$$

$$= \frac{1000}{433}$$

$$= 2.31$$

NOTES:

[1] Auerbach, Page 740:001.061

6. Telecommunication Efficiency

$$Pe_6 = \left[(3)\ (211) \right] \div 100$$

$$= 633 \div 100$$

$$= 6.33$$

Note: 131,032 core memory was proposed for a 100-terminal system. Core has been extrapolated to 210,961 for a 161 Terminal dedicated CAI system. The Sigma-7 system is limited to a maximum of 200 competing users.(1)

G. NCR - CENTURY 200

1. Processor Speed Efficiency

$$Pe_1 = \left[\left(\frac{1}{(8)\ (2)} \right) \times (.80) \right] \times 1000$$

$$= \frac{1}{16} \times .80 \times 1000$$

$$= .05 \times 1000$$

$$= 50$$

---

2. Processor Instruction Mix Efficiency

Add/Substract 15.2 microseconds + 7.2 D D = No digits
Multiply 3780 microseconds for 5 digits
Divide 9930 microseconds for 5 digits

$$Pe_2 = \left[\frac{51.2 + 9930 + 3780}{3} \right] \div 10$$

$$= \left[\frac{13761}{3} \right] \div 10$$

$$= \left[4587 \right] \div 10$$

$$= 45.9$$

---

3. Tape Processor I/O Efficiency

633-121

$$Pe_3 = \frac{100.000}{(80) \times (800)}$$

$$= \frac{100.000}{64.000}$$

$$= 1.56$$

NOTE:

(1) Auerbach, SDS Sigma 7, Summary Report, Page 740:001.010

---

4. Disk/Drum Efficiency 655-201

$$Pe_4 = \left[\left(\frac{100}{108\ KC} \right) \times \left(\frac{70.44}{100} \right) \right] \times 10$$

$$= \left[(.925) \times (.704) \right] \times 10$$

$$= \left[.6512 \right] \times 10$$

$$= 6.51$$

---

5. Reader/Punch/Printer/Efficiency

Note: 1) #686-Cards-Read/Punch punches at 82 cards/minute for punching 80 columns. (1)

2) #640-Printer-1500 lines per minute per 48 contiguous characters. (2)

3) #686-Card Read/Punch reading speed as indicated in NCR proposal is 300 CPM.

---

(1) Auerbach, NCR Century Series, Page 620:072.100

(2) Auerbach, NCR Century Series, Page 620:081.100

$$Pe_5 = \left[\frac{1000}{\frac{(300) + (82) + (1500)}{3}} \right]$$

$$= \frac{1000}{627}$$

$$= 1.59$$

---

6. Telecommunication Efficiency

Core proposed was 128 K Byte memory for a 50-terminal teletype system. In lieu of a formula, core is extrapolated to be 412 K Bytes for a 161 terminal system.

$$Pe_6 = \left[(3) \times (412) \right] \div 100$$

$$= \left[1236 \right] \div 100$$

$$= 12.36$$

H. DIGITAL EQUIPMENT CORP. PDP-10

1. Processor Speed Efficiency

$$Pe_1 = \left\langle \left[\left(\frac{1}{(36) \times (1)} \right) \times 1.0 \right] \times \frac{1}{2} \right\rangle \times 1000$$

Note: Digital Equipment proposal contains a statement on Asynchronous Operation. (Refer to Page 5 of Digital Proposal to Hostos Community College, dated December 15, 1969.) It is assumed that the proposed 64 K total core memory would be used in two way interleaving of banks of 32 K core each, by means of a hardware switch.

$$Pe_1 = \left[\frac{1}{36} \times \frac{1}{2} \right] \times 1000$$

$$= \left[.0305 \times \frac{1}{2} \right] \times 1000$$

$$= \left[.01525 \right] \times 1000$$

$$= 15.25$$

---

2. Processor Instruction Mix Efficiency

$$Pe_2 = \left[\frac{(2.53 + (10.6) + (16.2)}{3} \right] \div 10$$

Note: From Digital Proposal to Hostos, Page 4-15

"Fixed Point Add" Execution Time = "2.53 microsec."

"Fixed Point Multiply" Execution Time = "10.60 microsec."

"Fixed Point Divide" Execution Time = "16.2 microsec."

"Average using random fixed point numbers."

$$Pe_2 = \left[\frac{29.33}{3} \right] \div 10$$

$$= \left[9.77 \right] \div 10$$

$$= .977$$

---

3. Tape Processor I/O Efficiency

The proposal recommends the TU-55 DEC tape transport, which is a fixed-address magnetic tape storage system which provides random access for high speed reading or writing of files. Because this type of tape transport is not comparable to other manufacturers, the TM 20 magnetic tape transport was substituted in this calculation step. The TU 20 reads or writes at the rate of 36 KC, and records at densities of 200, 556, and 800 BPI. Seven or nine channel halfinch industry standard tape is used.

$$Pe_3 = \frac{100,000}{(36\ KC) \times 800}$$

$$= \frac{100,000}{28,800}$$

$$= 3.47$$

---

4. Disk/Drum Efficiency RD10 Swapping Disk File [1]

$$Pe_4 = \left[\left(\frac{100}{(76\ KC)} \right) \times \left(\frac{17}{100} \right) \right] \times 10$$

$$= (1.31) \times (.17) \times 10$$

$$= .2227 \times 10$$

$$= 2.23$$

---

[1] Digital PDP-10 Systems Description Booklet

---

5. Reader/Punch/Printer Efficiency

No card reader, punch, or printer was proposed. The CR10 Card Reader at 1000 CPM is used in this example. The LP10A Line Printer at 300 lines per minute, 64 character font, 132 columns/line is used in this example. Because no card punch exists in the product line, none is used in this example.

$$Pe_5 = \frac{1000}{\frac{1000 + 300}{2}}$$

$$= \frac{1000}{650}$$

$$= 1.53$$

---

6. Telecommunication Efficiency

The proposal states that one PDP-10 system of 64 K memory will serve as a freestanding 40 terminal system. It has been extrapolated that four (4) Digital PDP-10 Systems would be required for a 161 Terminal System; and that 256 K of core would be required in the total system.

$$Pe_6 = \left[(3) \times (256) \right] \div 100$$

$$= \left[786 \right] \div 100$$

$$= 7.68$$

I. HEWLETT-PACKARD HP-2116-B

1. Processor Speed Efficiency

$$Pe_1 = \left[\left(\frac{1}{(16) \times (1)} \right) \times (1.6) \right] \times 1000$$

$$= \left[\left(\frac{1}{16} \right) \times (1.6) \right] \times 1000$$

$$= (.100) \times 1000$$

$$= 100$$

---

2. Processor Instruction Mix Efficiency [1]

Add 3.2
Multiply 19.2
Divide 20.8

$$Pe_2 = \left[\frac{(3.2) + (20.8) + (19.2)}{3}\right] \div 10$$

$$= \left[\frac{43.2}{3}\right] \div 10$$

$$= \left[14.4\right] \div 10$$

$$= 1.44$$

---

3. Tape Processor I/O Efficiency

7970 Magnetic Tape Transports were proposed. However, the only data available was applicable to the 7980 Magnetic Tape Transport. Therefore, the 7980 unit was used for this example. Transfer rate was calculated by multiplying 800 BPI density x 120 IPS Tape Speed, yielding 96 KC as a transfer rate.

$$Pe_3 = \left[\frac{100.000}{(96) \times (800)}\right]$$

$$= \frac{100.000}{76.800}$$

$$= 1.30$$

---

4. Disk/Drum Efficiency

HP-2771A Disk

$$Pe_4 = \left[\frac{100}{(160\ KC)} \times \frac{8.35}{100}\right] \times 10$$

$$= \left[(.625) \times (.0835)\right] \times 10$$

$$= \left[.0525\right] \times 10$$

$$= .53$$

---

5. Reader/Punch/Printer Efficiency

HP-2779A High-Speed Punch Card Reader - 1000 CPM is substituted for the proposed HP-2731A Paper Tape Reader in this example.
The HP-2754B Teleprinter that was proposed prints at 10 characters a second. Extrapolating this to minutes, we obtain 60 characters a minute, or one (1) line per minute.
No summary punch is available on the HP-2116-B System.

$$Pe_5 = \left[\frac{1000}{\frac{(1000) + (1)}{2}}\right]$$

$$= \frac{1000}{500.5}$$

$$= 1.99$$

---

[1] Instruction Times from Hewlett Packard, 2116B, "A powerful generalpurpose computer for scientific, real-time, and instrumentation applicators.

6. Telecommunication Efficiency

The proposed HP-2116B System will handle 32 terminals as stated on Page 3 of the Hewlett Packard Proposal to Hostos Community College, dated December 18, 1969. Sixteen (16) K of core is required to handle 32 terminals on one HP-2116-B System. Extrapolating the requirements for a 161 terminal system we require five (5) HP-2116-B Systems, with a total core requirement of 80 K.

$$Pe_6 = \left[(3) \times (80)\right] \div 10$$

$$= (240) \div 10$$

$$= 24.0$$

J. IBM 360/67

1. Processor Speed Efficiency

$$Pe_1 = \left[\left(\frac{1}{(8)\ (8)}\right) \times (.75) \times \frac{1}{2}\right] \times 1000$$

$$= \left[\left(\frac{1}{64} \times .75\right) \times \frac{1}{2}\right] \times 1000$$

$$= \left[.0117 \times \frac{1}{2}\right] \times 1000$$

$$= \left[.00585\right] \times 1000$$

$$= 5.85$$

---

2. Processor Instruction Mix Efficiency - 5 Digits to 5 Digits

Fixed Point, Single Precision

Add = 3.88 + 0.3 D
= 3.88 + 1.5
= 5.38

Multiply = 4.25 + 2.1 D + 0.3 D^2
= 22.25

Divide = 6.44 + 2.3 D + 0.6 D^2

D = The number of decimal digits in the Operand. [1]

---

[1] Auerbach, IBM System 360, Model 67, Page 427:051:300

$$Pe_2 = \left[\frac{5.38 + 32.94 + 22.25}{3}\right] \div 10$$

$$= \left[\frac{20.19}{3}\right] \div 10$$

$$= \left[6.73\right] \div 10$$

$$= .673$$

---

3. Tape Processor I/O Efficiency

IBM Model 2401-5 Tape Drives

$$Pe_3 = \frac{100.000}{(61.2\ KB)\ \ (1600)}$$

$$= 1.00$$

---

4. Disk/Drum Efficiency

IBM 2314 Disks

$$Pe_4 = \frac{100}{277} \times \frac{87.5}{100}$$

$$= 3.15$$

---

5. Reader/Punch/Printer Efficiency

$$Pe_5 = \frac{1000}{\frac{600 + 300 + 600}{3}}$$

$$= 2.00$$

Note: For IBM 2501 Reader/Punch estimated at 600 CPM Read, 300 CPM Punch, and a 1403 Printer at 600 CPM.

---

6. Telecommunication Efficiency

$$Pe_6 = \left[(3)\ \ (223) \right] \div 100$$

$$= 6.7$$

K. IBM - MODEL 75 TWO WAY INTERLEAVING

1. Processor Speed Efficiency

$$Pe_1 = \left\langle \left[\left(\frac{1}{(8)\ \ (8)} \right) \ (.75) \right] \times \left[\frac{1}{2} \right] \times 1000 \right\rangle$$

$$= \left[\left(.0117 \right) \left(\frac{1}{2} \right) \right] \times 1000$$

$$= 5.85$$

---

2. Processor Instruction Mix Efficiency

$$Pe_2 = \frac{(4.87) + (22) + (15)}{3} \div 10$$

$$= 1.37$$

Note: From Auerbach Computer Characteristics Digest October, 1969, page 34. Add time is an average of 2.3 microseconds best case, plus 7.3 - microseconds worst case. The divide time, is an average of 9.0 microseconds best case, and 33 microseconds worst case. The multiply time is an average of 5.1 microseconds best case and 25 microseconds worst case.

3. Tape Processor I/O Efficiency

$$Pe_3 = \frac{100.000}{(61.2\ KB)\ \ (1600)}$$

$$= \frac{1}{(61.2)\ \ (1600)}$$

$$= 1.00$$

Note: 61.2 KB represents a rate for 1.000 character blocks for the IBM Model 2401 - 5. Proposed Tape Drives. (Auerbach, Computer Characteristics, October, 1969, page 63.)

---

4. Disk/Drum Efficiency

For IBM 2314 Disk Storage

$$Pe_4 = \left[\left(\frac{100}{277} \right) \times \frac{87.5}{100} \right] \times 10$$

$$= \left[(.36) \times (.875) \right] \times 10$$

$$= \left[.315 \right] \times 10$$

$$= 3.15$$

---

5. Reader/Punch/Printer Efficiency

For IBM 2501 Reader Punch estimated at 600 CPM Read; 300 CPM Punch; and 1403 Printer at 600 LPM.

$$Pe_5 = \frac{1000}{\frac{(600) + (300) + (600)}{3}}$$

$$= \frac{1000}{500}$$

$$= 2.00$$

---

6. Telecommunication Efficiency

IBM System 360 Telecommunication Efficiency for 161 Terminals

$$Pe_6 = (3)\ \ (223) \div 100$$

$$= 6.7$$

---

7. CSE = (4.65) + (1.37) + (1.00) + (3.15) + (2.00) + (6.7)

For IBM Model-75

$$= 18.87$$

L. UNIVAC 1108 (With 2-Way Parallel Processing)

A Parallel Processor

1. Processor Speed Efficiency

$$Pe_1 = \left\langle \left[\frac{1}{36)\ (1} \left(.75\right) \right] \times \left[(.50)\ \tfrac{1}{2} \right] \right\rangle \times 1000$$

One full word fetch of (36 bits) 6 characters

$$= \left[\left(.020775\right) \times \left(.25\right) \right] \quad 1000$$

$$= 5.19$$

---

2. Processor Instruction Mix Efficiency

$$Pe_2 = \frac{(2.3) + (11.6) + (3.9)}{3} \div 10$$

$$= .57$$

---

3. Tape Processor I/O Efficiency

For Uniservo Viii - C Univac

$$Pe_3 = \frac{100.000}{(69.6\ KB)\ \ (800)}$$

$$= \frac{100.000}{55680}$$

$$= 1.79$$

---

4. Disk/Drum Efficiency

For Univac 9414 Disk

$$Pe_4 = \left[\left(\frac{100}{312}\right) \times \left(\frac{60}{100}\right) \right] \times 10$$

$$= \left[(.32)\ \ (.60) \right] \times 10$$

$$= 1.92$$

---

5. Reader/Punch/Printer Efficiency

For 1108 Punched Card Subsystem Reader is 900 CPM Effective;
Punch is 300 CPM effective output;
Printer Model 0758-00, is 1200 Lines per minute effective speed.

$$Pe_5 = \frac{1000}{\frac{(900) + (300) + (1200)}{3}}$$

$$= 1.25$$

---

6. Telecommunication Efficiency

Univac System 1100 Telecommunication Efficiency for 161 Terminals

$$Pe_6 = (3)\ (211) \div 100$$

$$= 633 \div 100$$

$$= 6.33$$

---

7. CSE for UNIVAC 1108

$$= (3.12) + (.57) + (1.79) + (1.92) + (1.25) + (6.33)$$

$$= 14.98$$

---

3. DETAIL ANALYSIS OF SOFTWARE COST/PERFORMANCE

SOFTWARE COST/PERFORMANCE SUMMARY

Number of Terminals	Honeywell Cost Terminal/ Mo.	Univac Cost Terminal/ Mo.	IBM Cost Terminal/ Mo.	Honeywell Core	Univac Core	IBM Core
161	$179	$18.46	$49.12	72K	212K	594K
480	$152	$11.43	$19.77	184K	428K	938K
960	$146	$ 9.73	$12.22	352K	758K	1478K

Note: Core Figure includes core requirements for Terminal Software, CAI Software and the Operating System.

A. HONEYWELL

EVALUATION COST/PERFORMANCE
HONEYWELL MOD-4 SOFTWARE AND HAL
IN A CAI ENVIRONMENT

SUMMARY EVALUATION OF COST/PERFORMANCE
HONEYWELL MOD-4 SOFTWARE AND HAL
IN A CAI ENVIRONMENT

By using the following formula, a cost of software per terminal in a CAI environment can be calculated:

$$\text{Software Cost per Terminal in Levels of Systems Design} = \frac{\text{Total Cost of Software for Terminal Design}}{\text{Number of Terminals}}$$

The following table is a summary of software cost per level of terminal design:

HONEYWELL SOFTWARE SUMMARY TABLE

Number of terminals	Cost of Honeywell Software per Terminal in Dollars/Mo. (7-Year Lease)
161	$179
480	$152
960	$146

HONEYWELL ELEMENT[1]	161 TERMINALS		480 TERMINALS		960 TERMINALS	
	CORE REQUIRED	COST/MONTH	CORE REQUIRED	COST/MONTH	CORE REQUIRED	COST/MONTH
1. MOD-4 Resident Monitor	40K	$ 1587.20	40K	$ 1587.20	40K	$ 1587.20
2. MOD-4 Partition-1	8K	317.36	8K	317.36	8K	317.36
3. MOD-4 Partition-2	8K	317.36	8K	317.36	8K	317.36
4. MOD-4 Input Files, Output Files, Job Classes, Job Queue Items	19K	754.73	19K	754.73	19K	754.73
5. MOD-4 User Operating Area	56K	2221.52	56K	2221.52	56K	2221.52
6. HAL Author Language Resident	32.7K	1297.21	32.7K	1297.21	32K	1297.21
7. Terminal VIP Message Handling and Software[2]	563.5K	22354.00	1680K	66645.00	3360K	133291.00
TOTAL	727.2K	$28848.00	1843.7K	$72939.00	3523.7K	$139785.00

(1) Core cost is $10,394 per 262K on a 7-year lease. This comes to $39.67 per month per 1K of core. Refer to Honeywell proposal to Hostos Community College dated March 26, 1970.

(2) Refer to Honeywell's Author Language System for Computer Assisted Instruction - Honeywell Education Application Summary.

SOFTWARE COST/PERFORMANCE

HONEYWELL - MOD 4 + HAL 161 Terminal System

Resident Element	Core Requirement
1. MOD 4 Resident Monitor	40K
2. MOD 4 Partition-1	8K
3. MOD 4 Partition-2	8K
4. MOD 4-8 Input Files	19K
5. MOD 4-6 Output Files	
6. MOD 4-26 Job Classes	
7. MOD 4-10 Job Queue Items	
8. MOD-4 User Operating Area	56K
9. HAL - Author Language Resident	32,768K
10. 161 Terminal VIP Message Handling & Software	563,500K
TOTAL	727,268K

HONEYWELL - MOD 4 + HAL 480 Terminal System

Resident Element	Core Requirement
1. MOD 4	131,000K
2. HAL - Author Language Resident	32,768K
3. 480 Terminal VIP Message Handling & Software	1,680,000K
TOTAL	1,843,768K

1. Honeywell Series 200 MOD 4 Operating System
2. Honeywell MOD 4, Page 2-1
3. Honeywell MOD 4, Page 4-3

HONEYWELL MOD 4 + HAL 960 TERMINAL SYSTEM

Resident Element	Core Requirement
1. MOD 4	131,000K
2. HAL - Author Language Resident	32,768K
3. 960 Terminal VIP Message Handling & Software	3,360,000K
TOTAL	3,523,768K

Non-Resident System Components

Element	Core Requirement
1. MOD-4 Input Reader	12K
2. MOD-4 Scheduler	8K
3. MOD-4 Transitional Monitor	8K
4. MOD-4 Output Writer	8K

Dependent Components - Within User Partitions

Element	Core Requirement
1. MOD-4 Cobol Compiler	56K
2. MOD-4 Fortran Compiler	28K
3. MOD-4 Assembler	36K
4. MOD-4 Easycoder	44K
5. MOD-4 Report Generator	28K

CRITERIA EVALUATION OF MOD-4 SOFTWARE

Element	Evaluation Factor Value
1. Fixed Task Partitions	.10
2. Variable Task Partitions	.20
3. Spooling Capabilities	0.00
4. Real Time Checkpoint and Recovery	.05
5. Remote Access File Management	.05
6. Full Background Utilities	.05
7. Compile and Execute Capability of Author Language (One-Pass Compile)	0.00
8. Total Communications Software	.50
9. Mass Storage Rollout and Rollback	.05
10. Systems Logging	.05
11. Total Hardware Support	.05
12. Storage Dynamically Assignable	.20
13. Executive Program Dynamically Assignable	0.00
14. Swapping of Priority Programs in Core	.20
15. User Programs with Integrity	.10
16. CAI Terminal Handler - Software	.05
17. Complete File Supervisor	.10
18. Data Handler Routine	.05
19. Process Files at Item or Block Levels	.05
20. Processor Interface Routines	.30
21. Real Time Priority System	.62

(NOTE: Thirteen levels of assignable priority are required. MOD-4 has only four levels of priority. Refer to MOD-4 Operating System Summary Description, Page 3-3.)

22. CMI W/S - 3 Minute Checkpoint	0.00
23. CAI Frame Handler - 3 Minute Checkpoint	0.00
TOTAL	2.77

HAL - AUTHOR LANGUAGE -
CORE CALCULATION SUPPLEMENT
DEDICATED CAI SYSTEM

FORMULA	ELEMENT	No. OF CHARACTERS
1.	HAL-AUTH. LANGUAGE RESIDENT	32,768[1]
2.	161 Terminal Vip-Message Handling + Software	563,500
	TOTAL	596,268

NOTE:

"The minimum hardware requirements for this system are: A Honeywell-type 1201 central processor with 32,768 characters of memory, and the specific capabilities required by the systems.

With this minimum configuration, up to six VIP devices can operate simultaneously on either one or two communication lines.

In order to increase the number of terminals, additional memory is required. Any increase above six VIP devices requires a basic 3,000 characters plus 500 characters for each display terminal."[2]

---

(1) Honeywell's Author Language System for Computer Assisted Instruction, CRP-0326, Page containing "Equipment Requirements."

(2) Ibid, Page containing "Equipment Requirements".

B. UNIVAC

EVALUATION COST/PERFORMANCE
UNIVAC - EXEC - 8 SOFTWARE
IN A CAI ENVIRONMENT

SUMMARY EVALUATION OF COST/PERFORMANCE
UNIVAC 1108 - EXEC 8 SOFTWARE +
COPI - II IN A CAI ENVIRONMENT

UNIVAC SOFTWARE SUMMARY TABLE

Number of Terminals	Cost of Univac Software per Terminal in Dollars/Mo. (Educational Discount)
161	$18.46
480	11.43
960	9.73

UNIVAC - 1108

ELEMENT	161 TERMINALS		480 TERMINALS		960 TERMINALS	
	CORE REQUIRED	COST/MONTH	CORE REQUIRED	COST/MONTH	CORE REQUIRED	COST/MONTH
1. Exec - 8	65,000	$1259	65,000	$1259	65,000	$1259
2. COPI-II Elements 1 thru 4	90,804	$1060	270,720	$3157	541,440	$6313
3. Buffer Pools	21,000	$ 245	57,000	$ 664	117,000	$1363
4. COPI-II Fixed Storage Requirements	35,000	$ 408	35,000	$ 408	35,000	$ 408
TOTAL	211,804	$2972	427,720	$5488	758,440	$9343

Note: 1. Main memory of the 1108 would cost Hostos $19.37 per 1 K of core/month rental.

2. LCS memory of the 1108 would cost Hostos $11.65 per 1 K of core/month rental.

3. Primary-main memory 1108 storage of 262 K is used for storing Exec-8 and other operating software. This main memory core has a list rental of $10,000/month; less 25% Educational discount; less 35% eight-year lease discount. Therefore, 10,000 less 25% is $7500; $7500 less 35% is $5075.

The large core storage of the Univac 1108 is called unitized channel storage. It has a cycle time of 2.25 microseconds on the 1108. The cost for 262 K modules of LCS is $6260/mo. list rental, less 25%, less 35%. It therefore would rent to Hostos for $3052/month.

SOFTWARE COST/PERFORMANCE
UNIVAC EXEC-8 + COPI-II
For 161 Terminals

Elements	No. of Words
1. thru 4. 564 x 161 =	90,804
5. Buffer Pools	21,000
6. Copi - II Fixed Storage Requirement	35,000
7. Exec - 8 Fixed Storage Requirement	65,000
TOTAL	211,804

SOFTWARE COST/PERFORMANCE
UNIVAC EXEC-8 + COPI-II
For 480 Terminals

Elements	No. of Words
1. thru 4. 564 x 480 =	270,720
5. Buffer Pools	57,000
6. Copi - II Fixed Storage Requirement	35,000
7. Exec - 8 Fixed Storage Requirement	65,000
TOTAL	427,720

SOFTWARE COST/PERFORMANCE
UNIVAC EXEC-8 + COPI-II
For 960 Terminals

Elements	No. of Words
1. thru 4. 564 x 960	541,440
5. Buffer Pools	117,000
6. Copi - II Fixed Storage Requirement	35,000
7. Exec - 8 Fixed Storage Requirement	65,000
TOTAL	758,440

In evaluating CAI Author Languages, Univac's COPI II was evaluated in terms of resident core requirements. To estimate the resident core storage requirement for COPI II, we must understand the following table:

Estimating Resident Core
Storage Requirements - COPI II

Formula	Element	# of Words*
1.	Terminal Message buffer workstorage (exec 8)	512 words/terminal
2.	Line table workstorage	7 words/terminal
3.	COPI II communications workstorage to identify terminals	25 words/terminal
4.	COPI II Pooling Terminals for buffer quering work-storage	20 words/terminal
	TOTAL	564 words/terminal

5. For every 25 terminals 3K for buffer pool words (queue message pool) (similar to IBM's Line Data Area work-storage).

6. Core Resident for COPI II 35K words this consists of 14K for Instruction Bank of COPI II, and 21K for Data Bank in COPI II.

7. Core Resident is Exec-8 the operating System for Communications processing, which requires 65K words.

C. IBM

EVALUATION COST/PERFORMANCE

IBM - MVI SOFTWARE

IN A CAI ENVIRONMENT

SUMMARY EVALUATION OF COST/PERFORMANCE

IBM - 360 MVT - SOFTWARE + COURSEWRITER

III IN A CAI ENVIRONMENT

IBM SOFTWARE SUMMARY TABLE

Number of Terminals	Cost of IBM Software per Terminal in Dollars/Mo. (Educational Discount)
161	$49.12
480	19.77
960	12.22

IBM - MVT + COURSEWRITER III COST/PERFORMANCE TABLE

ELEMENT	161 TERMINALS CORE REQUIRED	161 TERMINALS COST/MONTH	480 TERMINALS CORE REQUIRED	480 TERMINALS COST/MONTH	960 TERMINALS CORE REQUIRED	960 TERMINALS COST/MONTH
1. MVT	262 K	$6471	262 K	$6471	262 K	$ 6471
2. Coursewriter III	108 K	$ 468	108 K	$ 468	108 K	$ 468
3. Terminal Message Handling Area	224 K	$ 970	568 K	$2460	1,108 K	$ 4798
TOTAL	594 K	$7909	938 K	$9399	1,478 K	$11,737

Notes:

1. Main core storage of 262 bytes was used for MVT costs. Large Core Storage was used for the remainder of core requirements. Large Core Storage costs $4.33/month on a 30% discounted basis per 1 K of core.

2. The following core costs were obtained from Larry George, IBM Account Representative, on April 2, 1970.
   - For Code 2365 Main Core Storage: 262,144 bytes at $9245/month list rental; less 30% educational discount is $6471/month.
   - For Code 2361 Large Core Storage: 1,048,576 bytes (4 microsecond cycle time) $6495/month list rental or less 30% at $4546/month.
   - For Code 2361 Large Core Storage of 2,097,152 bytes at $10,990/month list, or less 30% at $7683/month rental.

SOFTWARE COST/PERFORMANCE

IBM - MVT + COURSEWRITER III

161 TERMINAL SYSTEM

Resident Element	Core Requirement
1. MVT - Fixed Area	
2. MVT - SVC Transient Area	
3. MVT - I/O Supervisor Transient Area	
4. MVT - System Queue Area	262 K
5. MVT - Link Pack Area	
6. MVT - Dynamic Area 15 Storage Keys per each 2 K Block[1]	
7. Coursewriter III (Including BTAM)	108 K[2]
8. 161 Terminal Messages/Handling Area (Suggested Method of Calculation)	223,353
TOTAL	593,353

[1] IBM System/360 Operating Systems, Program Logic, Form Y28-6658-2.

[2] Coursewriter III for System 360, Version 2, Application, Description Page II.

SOFTWARE COST/PERFORMANCE

IBM -MVT + COURSEWRITER III

480 TERMINALS

Element	Core Requirement
1. MVT	262,000
2. Coursewriter III (incl. BTAM)	108,000
3. 480 Terminal Message-Handling Area (Suggested Method of Calculation)	567,724
TOTAL	937,724

SOFTWARE COST/PERFORMANCE

IBM - MVT + COURSEWRITER III

960 TERMINALS

Element	Core Requirement
1. MVT	262,000
2. Coursewriter III	108,000
3. 960 Terminal Message-Handling Area (Suggested Method of Calculation)	1,107,570
TOTAL	1,477,570

DEFBLKS FOR 161 TERMINALS

IBM COURSEWRITER III

REVISED METHOD OF CALCULATION

Element	Revised Method of Calculation
Fixed	660
(64) (4 Tapes)	264
(2048) (20 Subroutines)	40,000
(64) (1 Type line)	64
(385) (161 Terminal Connect Lines)	61,985
(123) (40%) (161 Teleprocessing Buffers for 500 Character Screen)	7,956
(263) (8 Disk Drive Units)	2,104
(2048)(25%) (161 Line Data Areas)	82,720
Fixed Storage Requirement	28,600
TOTAL	223,353

DEFBLKS FOR 480 TERMINALS

IBM COURSEWRITER III

REVISED METHOD OF CALCULATION

Element	# of Bytes
Fixed	660
(64) (4 Tapes)	264
(2048) (20 Subroutines)	40,000
(64) (1 Type Line)	64
(385) (480 Terminal Connect Line)	184,800
(124) (110%) (480 Teleprocessing)	65,472
(Buffers for 500 Character Screen)	
(263) (8 Disk Drive Units)	2,104
(2048) (25%) (480 Line Data Areas)	245,760
Fixed Storage Requirement	28,600
TOTAL	567,724

DEFBLKS FOR 960 TERMINALS

IBM COURSEWRITER III

REVISED METHODS OF CALCULATION

Element	# of Bytes
Fixed	660
(64) (4 Tapes)	264
(2048) (20 Subroutines)	40,000
(64) (1 Type Line)	64
(385) (960 Terminal Connect Lines)	370,600
(124) (170% (960 Teleprocessing Buffers for 500 Character Screen)	202,368
(263) (8 Disk Units)	2,104
(2048) (25%) (960 Line Data Areas)	491,520
TOTAL	1,107,570

SOME EXAMPLES OF COMPUTER-BASED INSTRUCTIONAL SYSTEMS

Alfred Colyer, John Gilligan and Arthur Oppenheimer

1. INTRODUCTION

One common and interesting feature of the schemes I shall try to describe is that they are all attempts to solve existing education and training problems. That is, rather than having to think up educational applications for our machines, we were fortunate in finding people for whom the computer seemed to offer a solution for at least some of their difficulties.

As a result of being involved in real-life situations for several years, we have reached a point of view, about which I would like to make some general remarks before going on to describe the applications.

When one works every day with computers, it is scarcely avoidable to get very excited about their present and projected capabilities. Large-scale operating systems, time-sharing, graphics, hardware etc. can so far occupy the foreground of our thinking, that it is worth reminding ourself that „Computers in Education", for example, is primarily about education, not about computers.

I am sure that we have suffered, and probably continue to suffer, far too much from the computer expert who is, for example, more fascinated by solving the problems of implementing an „author-language" than he is either concerned or informed about its educational feasibility.

In contrast, it is easy to see that this approach would not be tolerated in a business situation. Any business is known to consist of a large number of inter-related sub-systems and when the implementation of computers is being considered, it is common and sensible practice to stand back and try to comprehend as large a canvas as possible. It is not obvious that this message has got through to the world of education and its associated computer experts to the same extent. In certain subject areas, the computer offers the opportunity for a genuine and beneficial revolution, and this is an opportunity which we are either slow or reluctant to face.

If the failure to think comprehensively enough is one danger, a second danger could be our ability to deceive ourselves concerning the subtlety and practicability of our technology. Except where a computer is being directly programmed by the student, most dialogue between the student and a computer can only be crude or impractical for many years to come. Similarly, information produced by the computer about students is unlikely to be useful without interpretation and amplification by a much more intuitive, i.e. human, mind. Relevant to the last point is a recently-reported complaint by an eminent American psychologist, Dr. Ogden Lindsay, that the computer's ability to store statistics and cross-refer was creating a genre of case-study generalisations which was undermining the psychologist's belief in the unique qualities of every human being.

Because individual human beings are unique and because, in our context, we are largely ignorant of how they learn, it seems sensible to build flexibility and open-endedness into any eductional system we devise. The open-ended system works within a broadly-defined frame-work, is flexible to the ad hoc needs of the individual, and is capable of day-to-day development.

Additional advantages of open-ended systems are safety, higher possibility of benefitting the community, and relative financial economy. Perhaps the most open-ended of all educational computer applications is the one which started at Dartmouth College in 1964. This was a safe thing to do since it was evidently feasible; it has been relatively economical in terms of software development and cost/benefit during operation; and it has been of obvious benefit to the community. There was never any doubt of its educational worth, since it was concerned, not with brain-washing facts or doctrines, but with helping students to discover basic intellectual concepts for themselves.

Finally, as we all know, it is possible for a computer-based open-ended system to provide feed-back on its own efficiency, thus pointing out where amendments can be made.

I hope that these few preceding remarks help to illustrate the frame of mind in which we have approached the following fairly large-scale experiments.

The first application may perhaps seem rather introspective, and therefore of limited interest, since it concerns the teaching of a computer language; but it should provide a means of illustrating some of the systems considerations that have already been mentioned. If using a computer to do computer-training seems an obvious thing to do, exactly how to go about it effectively and economically is not always so clear.

The second illustration is of a computer assisting in the management of a highly complex individualised mathematics teaching project in secondary schools. I hope to show that the subject-matter is not important: it could be individualised anything, since the main role of the computer is not to teach, but to prescribe a unique programme of work for every student and to report on the efficiency of the teaching material and methods.

Finally, I would like to draw attention to other experiments where we have been involved in the use of terminals in schools and colleges. With the exception of the mathematics management scheme, which goes live in a few weeks' time, all the schemes I mention have been running in an educational environment for some time.

2. TEACHING OF A COMPUTER LANGUAGE

The first example, then, is of computer training at Brighton Polytechnic. The task at Brighton was to train

students to write and test off-line FORTRAN programs. These were undergraduate students in mathematics, engineering, chemistry, physics etc. Learning about computing was incidental to their main studies and, in fact , it was unlikely that they could spare more than two or three hours per week for this activity. It was required that within about ten weeks of his first year at the College the student would be brought to the point where he could easily find out for himself anything else he wanted to know about using the language.

The previous method of using highly-qualified staff from the Computer Department to give elementary lectures on FORTRAN to computing novices proved quite unsatisfactory: this is, of course, a common problem at the university and technical college level.

The system we devised was based on the use of a programmed-text, a student's workbook, and a suite of specially-prepared monitoring software. The programmed text, written by a colleague, Mr. John Watters, is one which has been available on the open market for some time.
The purpose of the workbook is to advise the student on the purpose of the whole package, how to prepare his programs for computer runs, how to interpret results from the printer, and so on. The software, designed for running under the GEORGE Operating System, supervises the compiling and running, and subsequent judging, of students' programs, and produces reports for the student himself and for the teaching staff. The teachers can easily be kept informed of the efficiency of individual students and of performance of all students on individual exercises; the latter, of course, gives an indication of the effectiveness of the teaching material.

The main computing hardware consists of a 1905E processor with 32K words of core, and two 4 million-word Exchangeable Disc Stores. This implies, of course, the use of the GEORGE 2 Operating System. The scheme was originally devised for running under GEORGE 3, so many amendments and extensions to the monitoring software (i.e. the applications program) had to be made. Even so, the use of the GEORGE 2 Operating System makes life a great deal easier for both students and staff.

A brief impression of the system from the student's viewpoint may perhaps be gained from the following two extracts from the student's workbook.

„Self-instruction is accomplished using a programmed text, that is a book written so that the student is taken step by step through the course. At each critical stage the student is asked a question based on the information he has just learned; and by replying and checking his response against the given answer, he is constantly monitoring his progress and finding out how well he understands the material. Clear directions are given and the student, in his own time and at his own pace,can work his way through the book."

Besides doing the exercises in the programmed-text, the student writes programs for problems given to him by the teaching staff. These are the programs which must be run on the machine and, here, the workbook goes on to tell him that

„In this scheme we have tried as far as possible to provide the student with a true life picture of program development and testing. The student analyses an exercise then flowcharts and codes the problem; test data is written and both program and test data punched on to cards. The program is then run against this test data and corrected if necessary until he is satisfied that the program is working correctly. Then the same program is run without the student's test data in what we call the „judging run". The „judging run" runs the program using a set of test data for this exercise supplied by the system, then compares the results produced by the program with the anticipated correct output. Thus for each exercise the system uses the judging run to provide a definite check on the student's program. This ensures that he has produced an accurate working program and has an absolute measure of the success of his work."

In this completely self-paced scheme, whenever he is ready the student takes his program pack along to the computer. Since the student's program is prefaced by pre-punched GEORGE macro cards, it slips naturally into the computing job-stream; that is, it has no disrupting effect on the other data-processing work of the college.

One useful side-effect of this self-paced instruction has already been a significant smoothing of the computing load, since students' programs now come along at random times.

From the lecturer's point of view there is the relief from a tedious lecturing load, plus the ability to obtain selective information about the student's performance from the machine. For example, a print-out on a particular exercise for a particular student may show a string of characters such as CCCRCCRCJCS, where C=unsuccessful compilation, R=unsuccessful run, J=unsuccessful judging run, S=successful judging run. So, in this hypothetical exercise, the student is shown to have taken eleven runs before getting his program to work; cause, one would think, for the lecturer to investigate how other students performed on this exercise, or how this particular student fared on other exercises.

Many enhancements are already in hand, and are relatively easy to incorporate, since this is not a rigidly-defined system. Amendments can be made to the programmed-text, to the workbook, or built into the software; and as more experience is gained, this will inevitably be done.

It seems hardly necessary to dwell on the economics of this system since, if running jobs on a computer is regarded as a sine qua non of programmer-training, and using an operating system the only smooth way of handling these jobs, then the additional cost involved is minimal.

In summary, then, the main characteristics of the Brighton scheme are that:

all training is based on individualised self-paced instruction;

all students have unrestricted access to the computer;

teachers have immediate access to the kind of information which enables them to isolate areas of inefficiency in the learning system.

3. INDIVIDUALISED MATHEMATICS INSTRUCTION

The next scheme I wish to describe is one which we usually refer to as the Tunbridge Wells project, since it is based on a scheme of individualised mathematics instructions started several years ago at Ridgeway School, Tunbridge Wells, by the senior mathematics teacher, Mr. Bertram Banks.

The system is based on the single idea that a child learns best if allowed to work at his own pace on material suited to his ability level. Although, in these days, this idea is by

no means new, it takes a lot of determination and some organisational changes to carry it through. But if the management of large numbers of children, all working at different speeds at different topics, can be handled, this is well worth doing. The alternative is orthodox classroom teaching by, often in these days, non-specialist teachers, and the consequences of this can be disastrous.

The crux of this auto-instructional scheme, as with any other similar system, is the provision of enough high-quality individualised material. Fortunately, other neighbouring schools have joined in the project over the last two or three years, and there exists a panel of teachers working and meeting in their own time, to provide a large catalogue of small modules of instruction on different mathematical topics. All newly-submitted material is tested in the member-schools before being incorporated into the scheme. The age range being covered at the moment is the three years 12-14, and the number of tasks so far included is 300. Ultimately, it is hoped that the scheme will cover five or six years of secondary education, when the number of tasks is expected to be about 1000.

These hundreds of tasks are arranged in eight horizontal levels of difficulty, and a number of vertical streams of mathematical activity. The main problem is, for each child, to select from the network a set of tasks exactly suited to that child's level of ability.

A task may involve working with a programmed text, programmed audio-tapes, with apparatus etc. Every task has built-in tests, many of which the student himself marks.

A complete set of 12 tasks may take a child several weeks. When he reaches this point he is given a comprehensive test, which the teacher marks. Up to now, the teacher has used these test marks together with his knowledge of previous performance, to determine the composition of the child's next group of tasks. But, clearly, selecting a new set of tasks, even from the several hundred so far available, becomes an impossible job for the busy teacher. It is clear that, from the test marks provided by the teacher, the computer can quickly prescribe a new set of work for any child. An example of a teacher conversing with a computer, via an on-line typewriter, to perform this function is shown in Figure 1.

In practice, although the quickest method of getting the teacher's marks to the computer is via the typewriter, it is expected that the line-printer and the postal service will be fast enough for sending the computer's reply. The typewriter can be used by the teacher subsequently, if necessary, for over-ruling any of the computer's choices.

Besides the obvious advantages of helping the teacher in his local situation, this system has other clear advantages when large numbers of schools are using the system. The number of schools in the scheme will, in fact, be increasing significantly during 1970-71, and it is obvious that massive evaluation of the teaching material will be taking place. Such evaluation will only be meaningful if the large volume of data thus created is collected centrally; from the amount of data potentially available to the computer, much can be learned about where improvements can be made to the system.

In an educational environment, where money is always difficult to find, there is some attractiveness in a method such as this which promises to supervise large-scale individualised mathematics education at a cost of only about £3. per child per year.

Schemes which were just dreams two or three years ago are now becoming real, and schemes of this size and promise reinforce the point made earlier concerning the need to think comprehensively about educational systems. One specific question may be worth asking in this connection. What is the purpose of setting a child an examination in mathematics? Is it to find out how much mathematics he knows? A system such as the one I have just described always has this information instantly available. And this kind of system does not just work for mathematics: it is a pattern for managing individualised instruction, and could therefore be appropriate for other academic subjects. So could the human race be spared many of the examinations it has to suffer, along with the false view of education that examinations can give? And, if we reject the educational argument, there is another question. Can we cope administratively with the examination system much longer, anyway?

4. TWO OTHER EXPERIMENTAL SYSTEMS

I would like to finish by referring to two other experimental systems, one at the Thomas Bennett School, Crawley; and the other at Kingston Polytechnic.

At the Thomas Bennett School we have gained valuable experience of working for a whole year in a secondary school environment, using Teletypes for teaching students and collecting report data for the teacher.

The two Teletypes at the school are linked through the public telephone network to the ICL 1903 at Hedsor Park. They are used by pupils working through the first year of the Nuffield biology course to test themselves on their understanding of the programmed-text material they have studied and the associated experiments they have completed. The pupils' answers are scored immediately and their final score tells them whether they can continue with the course or whether they must first see their teacher. Each pupil's answers to the tests are stored by the computer in an individual student record to which the teacher has access to check on each pupil's progress.

Any measures of success we have in this experiment are purely subjective ones and I make no apologies for this: there is a limit to the time one can spend in trying to quantify the obvious. And what is obvious is this:

a) the system is motivating for the children and they are anxious to get through their work so that they can take their tests on the computer
b) children who cannot normally spell, learn to spell for the computer (even though we have arranged that they don't have to)
c) the system is useful in producing reports for the teacher, reports which he would otherwise have to work out for himself since, in an individualised learning situation, he needs this information.

But this is still a small-scale experiment: only one biology class of about 33 children is involved, and the terminals are on-line to the computer for a total of no more than three hours per week.

Nevertheless, we have here a school situation in which the computer has come to be accepted as part of the environment. We are in a position to start studying the effects that different methods of data collection have on the children's work. And as more programmed teaching material, both in Biology and in General Science, becomes available there will be no difficulty in building it into the system.

The final example is based on work done by two

Fig. 1

```
←SCHEDULE
 CLASS?
←2BI
 NAME?
←KEVIN TUPNER
 ON SET 5
 WHEN DID HE FINISH?
←15/12/69
 ACTUAL SCORE,POSSIBLE SCORE?
←22 25

 DATE STARTED          01/10/69
 DATE FINISHED         15/12/69
 TEST SCORE              88%
 MEAN DIFFICULTY        4.82

 TYPE CODES FOR COMPLETED TASKS
A 1←1
  2←1
  3←3
  4←1
B 1←1
  2←1
  3←3
  4←1
C 1←1
  2←1
  3←2
  4←1

 I SUGGEST NEW SET AS FOLLOWS
A 1 NS.5.00 P/B        BINARY NUMBER 1
  2 NS.5.01 P/B        BINARY NUMBER 2
  3 NS.5.02 P/B        BINARY NUMBER 3
  4 NC.6.01 P/B        LOGARITHMS 1
B 1 NC.6.02 P/B        LOGARITHMS 2
  2 GP.5.10 W/S        CENTRE OF TRIANGLE
  3 GP.5.11 W/S        CENTRE OF CIRCLE
  4 GS.5.12 W/S        CYCLIC QUADRILATERAL
C 1 GS.5.20 TAPE 24    HOW MUCH LARGER?
  2 GS.5.30 W/S        DUALITY
  3 TO.3.11 W/S        NETWORKS 1
  4 TO.3.12 W/S        NETWORKS 2

 DO YOU WISH TO OVERRIDE
 ANY OF MY CHOICES?
←YES
 WHICH TASK?
←C4
 REPLACEMENT?
←DECODING 1
 IS THAT ALL?
←YES

 O.K NEW SET AS FOLLOWS
A 1 NS.5.00 P/B        BINARY NUMBER 1
  2 NS.5.01 P/B        BINARY NUMBER 2
  3 NS.5.02 P/B        BINARY NUMBER 3
  4 NC.6.01 P/B        LOGARITHMS 1
```

Teacher types codes to tell the Computer that the tasks just tested for the student were

Too difficult	: 3
Too easy	: 2
Just right	: 1

Computer prescribes new Set of tasks

lecturers in the Business Studies Department at Kingston Polytechnic. We are all well aware of the growing number of students following courses in Social Science: one of the unfortunate consequences of doing social studies is that one has to learn some statistics: many of these students have little mathematical background, and, in these circumstances particularly, statistics is a difficult subject to teach.

Again the lecturers sought to overcome the learning difficulties by using programmed-instruction. And to save themselves from the repetitive and trivial task of checking simple exercises, it seemed worth using an on-line Teletype for just this purpose. At the time of preparing this paper, we are in the middle of a six-week experiment using two Teletypes at the Polytechnic connected to the 1903 at Hedsor Park.

Figs. 2 & 3 show the start we have made at Kingston. Provided one remembers that the small arrow precedes every entry typed by the student, these two examples should be easy to follow. First, the student is taken through the reasoning of an example, and the computer checks his ability to make the required calculations; then, he is interrogated on his knowledge of the formulae for several statistical functions.

Not only did this strike us as a sensible use for a typewriter terminal, but we were also attracted by the fact that the terminal was to be used in conjunction with a programmed text. This suggested to us an economical use of a computer, and the objectivity that programmed instruction implies.

Now, before concluding, I must thank all our friends at Brighton Polytechnic, Kingston Polytechnic, Thomas Bennett School, Ridgewaye School and elsewhere who have helped and continue to help us so much in this work. What we may have done in the way of experimental software follows entirely from the innovative ideas being worked upon in these educational institutions and this is, I think, the right way round.

Fig. 2

```
PROBLEM AND MODE PLEASE
← 11
     F     M                    F*M
    10    75  ←                750
    17    85  ←               1445
    39    95  ←               3705
    24   105  ←               2520
     7   115  ←                805
     3   125  ←                375

SF=←100
SF*M= ← 9600
X= ← 96.00
     F     M          M-X          (M-X)**2          F((M-X)**2)
    10    75  ←     -21  ←          441  ←             4410
    17    85  ←     -11  ←          121  ←             2057
    39    95  ←      -1  ←            1  ←               39
    24   105  ←       9  ←           81  ←             1944
     7   115  ←      19  ←          361  ←             2527
     3   125  ←      29  ←          841  ←             2523

S(F(M-X)**2)= ← 13500
SD ← 11.62


DO YOU WANT TO DO ANOTHER PROBLEM ? ← YES
DO YOU WISH TO READ IN DIFFERENT DATA? ← NO1
PLEASE RETYPE YOUR REPLY ← NO
```

Fig. 3

```
PROBLEM AND MODE PLEASE
← 12
WHAT IS EQUATION FOR MEAN
X=← S(M*F)/SF
MEAN= 96.00
WHAT IS EQUATION FOR STANDARD DEVIATION
SD=←SQRT(S(F*(M-X)**2)/SF)


STANDARD DEVIATION = 11.62

PROBLEM AND MODE PLEASE
← 12
WHAT IS EQUATION FOR MEAN
X= ← S(M--X)/SF
WRONG ANSWER TRY AGAIN
WHAT IS EQUATION.FOR MEAN
X= ← S(M*F)/SF
MEAN= 96.00
WHAT IS EQUATION FOR STANDARD DEVIATION
SD= ← SQRT(S(F*(M-X)/SF))
WRONG ANSWER TRY AGAIN
WHAT IS EQUATION FOR STANDARD DEVIATION
SD=←SQRT(S(F*(M-X)**22
CANCELLED
← SQRT(S(F*(M-X)**2)/SF)
STANDARD DEVIATION = 11.62

DO YOU WANT TO DO ANOTHER PROBLEM? ← YES
DO YOU WISH TO READ IN DIFFERENT DATA? ← YES
NEW DATA FOR PROBLEM 1 HAS BEEN READ IN
```

THE TEACHING OF CLINICAL DIAGNOSIS USING A COMPUTER BASED SYSTEM

F. T. de Dombal

1. AVAILABILITY OF 'CONVENTIONAL' RESOURCES

In the sphere of Medical Education, the United Kingdom has until recently been singularly fortunate in respect of its clinical facilities. This particularly applies to the numbers of **patients** available for study, so that in the U.K. students have until recently enjoyed adequate opportunities for practising their diagnostic skills. Moreover, the number of medical **teachers** available has until recently ensured a generally acceptable degree of student supervision during this formative period.

All of this is now changing. The student population is rapidly expanding – in some teaching centres to twice or three times the 1969 figures – and considerable problems ensue as these expanding hordes of students face a relatively static population of medical teachers and patients.

Indeed, in some hospitals the number of patients suitable for teaching is actually decreasing. This is because teaching hospitals in the U.K. are becoming increasingly specialized in their patient intake; so that patients with common but 'non-academic' ailments tend to be admitted to 'non-teaching' hospitals, physically separated from the larger teaching centres, and well away from any students!

This in turn means that a) individual students tend to have decreasing opportunities to practise their newly-acquired diagnostic skills; b) the degree of supervision and control of each individual student by his clinical teachers may fall to an unacceptably low level; and c) the **type** of patient seen by the student may not be representative of the patients he will encounter in day-to-day clinical practice in his subsequent career.

Clearly, therefore, there is a major problem about to confront clinical teachers in the U.K. during the next few years – and a problem, moreover, which conventional teaching techniques may not readily be able to solve. And, this being so, there is therefore an equally clear case for the exploration of 'unconventional' teaching techniques (such as those involving computer-based systems) in the sphere of Medical Education in the U.K.)

2. POSSIBLE USES OF THE COMPUTER

Dr. Zinn, in his important paper (see page III/235) comments cogently upon the 'dimensions' of computer use in a comparable situation, but one or two specific comments about possible users in the sphere of clinical medicine might be desirable.

First, the 'tutorial' mode, in which the student works through a **linear or simple branching program** under close computer control. This type of program would seem to have relatively little place in teaching clinical diagnosis by computer. Apart from the expense of a computer based system – its use for this sort of teaching program is in my own view uneconomic – diagnosis is an individual process, the 'facts' of which are difficult to teach in didactic fashion because they themselves are so ill-understood. Ideally, therefore, computer-based systems employed to help teach clinical diagnosis should be 'student-orientated' in which the student is encouraged wherever possible to develop his own initiative.

The same considerations apply to the **provision of stereotypes** of various diseases for students to practise diagnosis. This may have some intrinsic merit – and indeed our own students seem to find it enjoyable and valuable – but it can be done infinitely more easily and cheaply without the aid of a computer. One further way in which the computer can help the student is by providing an **on-line statistical analysis** of any attribute sub-set which the student happens to have collected. Our own experience with this is fragmentary, and it is by no means certain that this is helpful to students in the long run.

Turning now towards 'student-orientated' modes of use, such as **simulation and game situations**, it would seem that these could be highly relevant to this type of teaching. We ourselves in Leeds have some experience in these fields. The details of these systems have already been reported elsewhere (de Dombal et al, 1969, a,b,c,). Clinical information about a number of patients is collected and stored in the computer. The student selects a patient for diagnosis; and obtains the relevant clinical information from the 'patient' via the computer – just as he would at the bedside – by taking a history, asking for physical findings, and so on. Each item of information is released by the computer in response to an appropriate question put by the student.

The student can also obtain, (if he wishes), helpful information from the computer about the meanings of terms used and about the relevance of certain symptoms and signs. Finally, if he feels he is not making much progress, the student may switch to a special remedial sequence merely by asking for 'Help'.

A large number of students have already used the system; and a 'library' of suitable cases has been established. Additional cases who are present on the wards may be added to the computer 'library' in less than two hours. The cost of preparing each **additional** hour of teaching material is therefore less than £1.

3. PROBLEMS IN IMPLEMENTATION

It may now be helpful to deal with some problems encountered in implementing such a computer based system.

3.1 *Lack of data-base*

By far the most serious problem encountered came with

our realization that much of our 'conventional' diagnostic teaching was based upon very scanty information both as regards the teaching of diagnosis and as regards the diagnostic process itself, as it is currently practised on the wards. Such has been the difficulty engendered by this lack of information that the major portion of our own effort is now concentrated in this sphere – that of investigating the diagnostic process itself – with fascinating results.

3.2 *Peripheral terminals*

In our current efforts, we have reverted to the use of the teletype – abandoning the use of more sophisticated equipment. This is not only on account of the cost of such equipment; but is chiefly because the random access audio-visual device we formerly employed took an unacceptably long time to respond – at least in this situation, and in the audio mode – to the students' query. Perhaps with the advantage of hindsight it would have been more suitable to have a random access audio disc – rather than to move audio tape physically from spool to spool in response to each student query.

3.3 *What are we trying to simulate?*

Here the work of Fraser (1966) is very relevant to our own concept of simulation. Some diagnostic simulator systems make what is in our view an error of judgment in attempting a 'whole-task' simulation. We, on the other hand, have aimed at a 'part-task' simulation. That is to say we are not trying to teach 'diagnosis' – whatever that may be! - as such, but instead we are trying to teach **selected aspects** of diagnosis, such as specific techniques of patient interrogation and so on.

4. PLACE OF COMPUTERS IN TEACHING CLINICAL DIAGNOSIS

It is this concept of 'part-task' teaching and simulation which surely provides the key to this final problem. The place of the computer is surely in an integrated teaching system involving real-life as well as artificial situations. To suggest that a computer by itself can teach clinical diagnosis – so that a student reaches the level of competence required from a medical practitioner, is as absurd as the suggestion that simulated training is adequate to permit a student pilot to carry fare-paying passengers between Leeds and Amsterdam. However, in some aspects of diagnostic teaching – such as in part-task simulation – computer-based systems may well have a role within an integrated curriculum. In the meantime, whatever the effect upon the students, the effect upon their teachers has been profound!

REFERENCES

de Dombal, F.T., Hartley, J.R. and Sleeman, D.H. (1969). Lancet 1, 145.

De Dombal, F.T., Hartley, J.R. and Sleeman, D.H. (1969). Brit. J. Surg. 56, 754.

de Dombal, F.T., Woods, Patricia, A, and Hartley, J.R. (1969). Brit. J. Surg. 56, 625.

Fraser. T.M. (1966). NASA SP-102 p78 et seq.

Zinn, K. (1970), „Instructional use of computers; A critical examination with recommendation for action". IFIP World Conference on Computer Education, III/235-240.

THE USE OF CRT SUBSYSTEMS IN A PROGRAMMED LEARNING ENVIRONMENT FOR OPERATIVE TRAINING

Christopher John Howson

1. INTRODUCTION

The next decade will, it is felt, see a sharp increase in the use of on-line techniques in the commercial data processing sector, as opposed to the scientific field. Such a development is being forced by steadily increasing costs of clerical staff, desires to shorten the working week, yet tempered by the need to increase efficiency and competitive operation. There is also a rapid growth in the understanding of the economies to be obtained by centralising operations, yet maintaining diversified information outlet points.

Notable examples of the development of real time systems are to be found in the major airlines, where reservations, passenger check-in and aircraft weight and balance control are being handled by on-line terminals. There are other applications: banking, insurance, public utilities, criminal detection, mail order houses, and so on, where the requirement for the rapid collection and dissemination of information has taken the computer out of its computer centre and into the second by second operation of business life.

This ingression of the computer on to the "shop floor" has placed new demands on those who must use the system as part of their daily activities. This paper is concerned with those demands and, where CRT terminal systems provide the man/machine interface, the use of such a system to provide an efficient education media.

2. THE PROBLEMS OF THE OPERATIVE

The ultimate success of a real time system lies, not with the analyst and programmer, although they play the major role during the design and implementation stages, but in the ability of those who, on a day to day basis, are able to use the facilities offered by the system in the most efficient manner. Therefore, there is a new sense of responsibility placed upon the operative. This responsibility may never be stated, but it is implied and is quickly seen as such by the operative. The degree of responsibility may be measured either by the extent of the integration of the computer system into the operative's work or by the financial investment in the system, some of which must be recouped by increased operative efficiency. Such increases in efficiency may only be attained by dexterity in the use of the system facilities by the operative.

If the business environment in which the computer is to operate is either complex or intricate, two further problems, which are definable as individual problems but are constituent elements of the responsibility problem, become apparent. Firstly, there is the problem of learning, understanding and committing to memory a number of entries and responses to and from the system and secondly, that of manipulating the various entries and responses to meet varying circumstances within the work pattern of the environment.

The physical use of the terminal device may, initially, cause some problems. The majority of terminals contemplated for commercial real time systems utilise a keyboard, either numeric or one similar to a typewriter. The number of adults who have never come into continued contact with an adding machine or typewriter and who view the use of such a keyboard with some alarm is surprising. The author, when involved with a large system found that Americans, Canadians and those from certain European countries were more familiar with the use of keyboards than were their U.K. contemporaries. This was attributed to two major factors; Americans and Canadians, both male and female, were introduced to typewriters during their High School days; in Europe, the use of secretaries and clerical typing pools is less evident than in the U.K. Such was the lack of this particular expertise, that special courses had to be devised for the U.K. operatives in keyboard dexterity.

There is a problem of technique to be overcome, that is, integrating the terminal into the work process, so that it replaces, say, a pencil or piece of paper. This aspect can be particularly important in an environment where the operative must undertake a conversation with a third party. For example, in an airline reservation situation, the operative must extract information from the prospective passenger and simultaneously translate this into inputs to the computer system, and pass computer responses to the passenger.

The final major operative problem is psychological. It concerns job security and work motivation. Until recently, automation has occurred primarily within the confines of factory processes. Where automation has occurred within commercial spheres, it has been within closely defined spheres of traditional batch processing applications – payrolls, sales accounting, statistics, stock control and so on. Such applications, although causing some traditional fears of redundancy and changes in working methods, cause little upheaval and affect relatively few people. However, with the development of highly integrated real time systems, embracing many related work areas, the computer is no longer a remote object but sits on the operative's desk. The apparent potential threat of the computer to change and affect the most intimate work processes can serve to heighten the normal fears of computerisation.

In summary, the training plan must overcome the following operative problems:

Responsibility

- Knowledge of system facilities
- Dexterity in use of system facilities in varying environmental situations.
- Operation of terminal devices
- Technique of integrating the terminal into work processes
- Psychological problems.

3. EDUCATIONAL PROBLEMS

Any of the problems stated above can produce fairly heavy demands on educational techniques. When the various problems cannot be readily subdivided and must be viewed as a total problem, the techniques employed in, and the phasing of, any training plan take on a highly complex form.

The most difficult technical problem to overcome, and one which forms the foundation stone on which the solution to many of the operative's problems can be built, is the provision of a realistic training media. This is true in nearly all adult training situations but it is even more vital in a real time computing environment.

The problems of committing to memory the system facilities and gaining dexterity in their use must form separate educational steps. However, dexterity of use and integrating the terminals into the work processes are better thought of as closely allied teaching problems since, when operating in the field, they are inter-dependent.

The operation of the terminal device can be treated either as a separate subject, or can be woven into the fabric of the training plan.

Fulfilling the responsibility problem should be a continuing aim of the training plan, which should be carefully phased so that confidence in both the system and its use is steadily built up.

The psychological problem must be dealt with at an early stage, even before detailed instruction is commenced. Perhaps the best way to meet this need is to provide a series of short presentations on the system the way in which it operates, its benefits – both to the company and to individuals, and so on. The essence is to present the system as just another business tool, which, if used correctly, will provide faster and easier working conditions.

Apart from satisfying the needs of the operative, certain other criteria must be met which could have a profound effect on the educational techniques employed. One of the major criteria is how the training should be presented. That is, what percentage of the total effort will be expressed in theoretical training and what percentage in practical training. Also, where and how will the balance between theoretical and practical be maintained?

Another very important factor is the number of operatives to be trained. In a large organisation, there may be two thousand people who must receive training in various system facilities. They may be spread over very wide geographical areas. This point poses critical questions: Can training be done on site? Should there be regional training centres? What is the maximum time span between receiving initial instruction and the system going live – should there be refresher courses? If so, when and where? How can a man's knowledge be maintained before he receives a refresher course? A further question concerns the instructors who are to design and present the courses. Should they be computer specialists, or should they be drawn from the field and given specialised training by the computer team?

Although these problems are subsidiary to the main theme of this paper, they are included to illustrate the various points which must be considered when an operative training plan for a real time system is being formulated.

4. A POSSIBLE SOLUTION – BASIC CONSIDERATIONS

It has been shown that the training plan must meet several criteria, from both the operative's viewpoint and that of the educational techniques. Further, it has been shown that if a realistic training media can be utilised, the solutions to many of the problems are well on the way to being found. Therefore, the solutions lie in building the training plan around the terminal device itself, and in using the terminal as a prime teaching media.

Having reached this basic decision, further questions of a more technical matter must be considered. Firstly, in the majority of cases, a terminal is of little use by itself. It requires a computer to drive it, in some cases to form characters on a CRT screen and so on. Further, the computer must have sufficient facilities to be able to drive several terminals simultaneously, so that a class of students may receive instruction, and it must be able to hold the programs and data required for the training sessions, either in core or on peripheral storage devices such as disc or drum.

It may seem preferable to use the operational system and hardware to provide the training facilities. However, the programs and hardware for the operational system may not be available in sufficient time to allow all the operatives to be efficiently trained before the system is due to cutover to live operation. Therefore, a separate computer is required, which need not be as powerful as the main system computers. If the terminal devices, in their operational role, are designed to be controlled by satellite computers, then the satellite computer may be used to act as a stand-alone system for training.

The number and location of the operatives to be trained may demand more than one training unit, each equipped with a small computer and a number of terminals. Once the initial training has been completed, the standalone computers can then be used for their operational purpose in the system. Adopting this method means that there may be no need to obtain additional equipment for training needs; certain components of the operational hardware on early delivery will suffice.

The basic system philosophy is to use the computer and its terminals as a teaching machine. Therefore, the operational system facilities must be reduced and segmented into teaching modules. Thus, in an airline reservation system, for example, the prime teaching modules might include such areas as:

- Seeking Availability
- Selling Space from Availability
- Creating a Basic Passenger Record
- Retrieving a Passenger Record from storage.

In a complex real time system, each of the prime teaching modules may contain many different entries and responses, and courses of action dependent on particular circumstances. Therefore, each prime module must be broken down into sub-teaching modules or lessons.

Each lesson and prime module should form a path of increasing complexity through the system facilities. However, the advanced modules should always utilise facilities learned in earlier modules. Normally, a real rime system which is transaction-based will have a certain number of entries and responses which form the backbone of the system. Such entries and responses must be constantly used throughout the training period, so that the operative

- treats them as second nature, thus speeding up his/her reaction time in an operational situation,
- has time to concentrate on the more complex facilities, thus gaining more efficient use of the system.

The integration of the CRT subsystem into the teaching environment must be given careful consideration. It is very unwise to use the subsystem as the prime teaching media. A careful balance between theoretical teaching and practical experience, using the subsystem, must be maintained.

The teaching system must operate in a conversational manner, simulating the operational system. A prompt to the student is displayed on the screen of the CRT, either instructing the student to take some action, or commenting on a previous action already taken. This is followed by a suitable entry being made by the student. If the entry is correct, then the relevant response is displayed, together with the next prompt. If the student had made an error, then a suitable error response must be given and perhaps an error prompt.

It is important that the responses, other than the prompts, given by the training system exactly match those of the operational system. The same holds for the inputs made by the student.

Each error made by the operative during a lesson should be logged by the system and output to a supervisor, not individual operatives, at the end of each training session. These, together with the results of theoretical sessions, can be used to closely monitor the progress of each operative and to direct him to more detailed study of any area in which he finds difficulty.

5. SYSTEM DESIGN

It is not the purpose of this paper to give a complete narration on systems design. However, to gain a deeper understanding of the operation of the system, it is worthwhile spending a few moments considering its structure. Figure 5.1 provides an overall picture.

This diagram is obviously very much simplified. It does not take into account such items as program and data loading, CRT control and so on. The major system areas are the prompts, input checking, error and normal response, output and the relationship between one step and the next within the lesson.

The prompts should be clear and unambiguous. They may appear on any part of the screen of the CRT, if desired, but since we are seeking to simulate the operational system, it would be more appropriate if they appeared on a part of the screen which would seldom be used by the system – say at the bottom. The prompts should lead the student step by step through the lesson, so that his confidence and knowledge is gradually built up at each step. Obviously, as the lessons increase in complexity, so will the prompts, but they should never lose their clarity.

The sytem under discussion was designed to compare an input made by an operative, in response to a prompt, with an expected input stored as part of the lesson data. The comparison method was to break the input down into fields and compare the input fields with those stored. For example, in an airline reservation system, the system input required to seek availability might be:

A 12MAY LHRJFK 1220

Where	A	Availability entry code
	12MAY	Desired date of travel
	LHRJFK	Board point and off point (London Heathrow and John F. Kennedy, New York)
	1200	Approximate departure time required by passenger.

In this case, the total entry breaks down into four fields. It is often the case that within one entry type, there may be alternative formats. These alternatives must be stored by the system. Therefore, each input must be analysed and details built up from which the computer data can be derived. For example, the above input could be analysed as shown in figure 5.2.

It should be remembered that the table shown in figure 5.2 is not only dependent on the input format for the entry, but also on the prompt which will cause the operative to make this entry.

Having defined the expected inputs in this manner, the error responses and error prompts can then be defined in a similar manner. The error responses should be the same as those which the operational system will output. The error prompts can supplement the error responses where necessary. Again, a table can be built up from which the training simulator data can be derived. Continuing with the example shown above, the error response table might formulate as shown in figure 5.3.

Following the error responses and error prompts, the response that would follow a correct entry and the next lesson step prompt can be defined. Again, the response must be as it will be in the operational system. A further table can be compiled as is shown in figure 5.4.

6. A BRIEF EXAMPLE OF THE SYSTEM IN ACTION

The example taken is for an airline reservations system and the first few steps required to create a passenger booking are shown. In order to clarify the situation, inputs from the operative are preceded by an asterisk.

The operative instructs the simulator that he wishes to start lesson 1.

```
*START 01
```

The system replies with the first prompt.

```
*START 01



HELLO. THIS LESSON DEALS WITH
AVAILABILITY. PLEASE SEEK AVAILABILI-
TY BETWEEN LONDON AND AMSTERDAM ON
01 JUNE DEPARTING AT ABOUT 1200.
```

The operative attempts an entry:

```
*A 01JUN LONAMS 1200



HELLO. THIS LESSON DEALS WITH
AVAILABILITY. PLEASE SEEK AVAILABI-
LITY BETWEEN LONDON AND AMSTERDAM
ON 01 JUNE DEPARTING AT ABOUT 1200.
```

The system checks the input. As it is correct, a response is displayed, together with a new prompt.

```
01 JUN
1 KL124 F4Y6 LHRAMS 1100 1155 DC9 TT 0
2 KL130 F2Y6 LHRAMS 1415 1510 DC9 LL 0
3 KL132 F0Y3 LHRAMS 1500 1600 DC9 TT 0


YES, YOU ARE RIGHT. NOW SELL SPACE ON THE
FLIGHT DEPARTING AT 1100. 2 FIRST CLASS
SEATS PLEASE.
```

The operative makes an entry:

```
01JUN
1 KL124 F4Y6 LHRAMS 1100 1155 DC9 TT 0
2 KL130 F2Y6 LHRAMS 1415 1510 DC9 LL 0
3 KL132 F0Y3 LHRAMS 1500 1600 DC9 TT 0
*S2F1



YES. YOU ARE RIGHT. NOW SELL SPACE ON
THE FLIGHT DEPARTING AT 1100. 2 FIRST
CLASS SEATS PLEASE.
```

This entry is not correct. The system displays the relevant error response and an error prompt.

```
01JUN
1 KL124 F4Y6 LHRAMS 1100 1155 DC9 TT 0
2 KL130 F2Y6 LHRAMS 1415 1510 DC9 LL 0
3 KL132 F0Y3 LHRAMS 1500 1600 DC9 TT 0
*S2F1
INVALID ENTRY


NO, NOT QUITE. REMEMBER THE ENTRY
CODE FOR BOOKING SPACE IS N FOR NEED
SEATS. TRY AGAIN.
```

The operative tries again:

```
01JUN
1 KL124 F4Y6 LHRAMS 1100 1155 DC9 TT 0
2 KL130 F2Y6 LHRAMS 1415 1510 DC9 LL 0
3 KL132 F0Y3 LHRAMS 1500 1600 DC9 TT 0
*S2F1
INVALID ENTRY
*N2F1

NO. NOT QUITE. REMEMBER THE ENTRY
CODE FOR BOOKING SPACE IS N FOR NEED
SEATS. TRY AGAIN
```

This time, the entry is accepted.

```
1 KL124 F 01JUN LHRAMS HS2 1100 1155



GOOD. YOU HAVE YOUR FIRST FLIGHT
SEGMENT. NOW WE MUST ENTER SOME
PASSENGER DETAILS. THEIR NAMES ARE
MR. AND MRS. B.J. SMITH. PLEASE ENTER
THESE NAMES.
```

So the lesson continues. It will be noted that the responses from the simulator are of a fairly convivial nature. This helps, particularly in the early stages of the training plan, to alleviate any fears the operative may have in conversing with a machine and to gain confidence.

7. CONCLUSIONS

This paper has briefly discussed the use of a CRT subsystem for adult education within a business environment. It was not intended to be a highly technical paper, rather to give some thoughts on the problems of computer education, borne out of some practical experience.

It is felt that such a system can provide the answers to many of the educational problems inherent in training operatives to use a real time computing system. However, it is felt that, particularly with the development in computer graphics, that this form of educational media could well be developed for other industrial and technical areas. For example, chemical engineering, structural engineering, mathematics and so on.

Where there is a degree of dynamicism in the working environment, the small computer, together with a cluster of CRTs or, special terminal devices related to a particular application, can provide a fast, efficient and flexible training media.

Fig. 5.1 SYSTEM BASIC BLOCK DIAGRAM

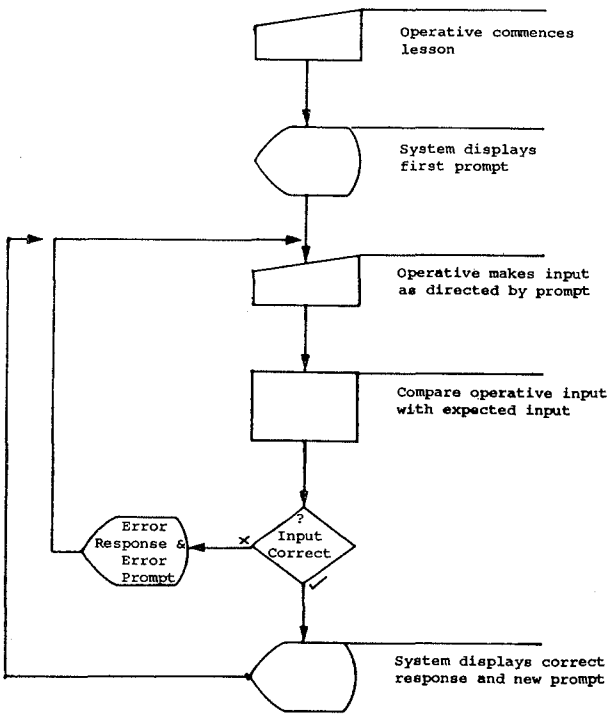


Fig. 5.2 INPUT TABLE

INPUT TABLE

ENTRY TYPE : SEEK AVAILABILITY

NUMBER OF FIELDS : 4

FIELD	NAME	NUMBER OF ALTERNATIVES	ALT 1	ALT 2	REMARKS
1	ENTRY CODE	1	A	–	–
2	DATE	1	12MAY	–	–
3	SEGMENT	2	LHRJFK	LONNYC	AIRPORT OR CITY CODES
4	TIME	1	1200	–	–

Fig. 5.3 ERROR RESPONSE TABLE

ERROR RESPONSE TABLE

ENTRY TYPE : SEEK AVAILABILITY

NUMBER OF FIELDS : 4

FIELD	ERROR RESPONSE	ERROR PROMPT
1	INVALID ENTRY	YOU WERE ASKED TO SEEK AVAILABILITY
2	INVALID DATE	TRY THE 12MAY PLEASE
3	INVALID CITY PAIR	AVAILABILITY IS REQUIRED BETWEEN LONDON AND NEW YORK
4	INVALID TIME	THE PASSENGER WANTS TO DEPART AT ABOUT 1200

Fig. 5.4 CORRECT RESPONSE TABLE

CORRECT RESPONSE & PROMPT TABLE

RESPONSE TYPE : AVAILABILITY DISPLAY

RESPONSE	12MAY 1 BA501 F2Y7 LHRJFK 1100 1340 V10 LL 0 2 BA505 F3Y7 LHRJFK 1300 1540 V10 LL 0 3 BA509 Y7 LHRJFK 1800 2040 707 DD 0
PROMPT	GOOD, THAT IS FINE. NOW BOOK 2 FIRST CLASS SEATS ON THE FLIGHT DEPARTING AT 1300.

'THE PRODUCTION AND USE OF TELEVISION COURSES ON 'LOGICAL DESIGN' AND 'PROGRAMMING IN ALGOL"

S.H. Lavington and J. S. Rohl

1. INTRODUCTION

The Department of Computer Science at the University of Manchester undertakes teaching in four main areas, namely:

(i) a three-year Honours Degree course in Computer Science,
(ii) a one-year postgraduate diploma/Master's course in association with the Department of Mathematics,
(iii) an undergraduate subsidiary course run for several other departments,
(iv) service-teaching for computer users.

It is in the last two areas that television has been found useful as an educational medium – primarily because it provides a means of dealing with the large numbers of students involved. Two 12-lecture television courses, plus accompanying booklets and work-schemes, have been prepared in conjunction with the University of Manchester Television Service, to meet this need.

Since the two courses differ in their design-philosophy, they will be described separately.

2. THE SYLLABUSES OF THE TWO TELEVISION SERIES

2.1 *'Logical Design'*

The syllabus for this course is based on two assumptions, namely:

(i) A familiarity with the general operation and potentiality of computers is becoming part of the basic education of all University science students.
(ii) Better programs are written by programmers who have a knowledge of computer configurations and device-performance.

To cover these two points the emphasis of the series is therefore on practical aspects of computer systems. The lectures are aimed at first-year undergraduates from several fields of study, and an elementary knowledge of physics – enough to know what is meant by 'voltage' and 'current' is the only course-requirement.

Wherever possible, photographs, models, and film-sequences of actual devices are used to enliven the theory, and performance and cost-figures are frequently mentioned. In this way, interest is maintained for the non-specialist. Many of the illustrative examples are taken from the three machines currently in use in the Department of Computer Science – namely, the Atlas, a 1905E, and a PDP8. A feature of the series is the concluding lectures, during which a complete small computer is designed. The detailed syllabus is as follows:

Lecture 1: Introduction; binary numbers and functions of binary variables.
Lecture 2: Electronic digit – representation; diode-resistor AND and OR gates; practical points.
Lecture 3: Logical arrangements for serial addition; performance of a particular adder; the need for parallel adders.
Lecture 4: Techniques for parallel addition – (synchronous; self-timing; carry look-ahead; others); comparative performance.
Lecture 5: Negative number representation, and subtraction.
Lecture 6: Slow and fast multipliers; division.
Lecture 7: Storage, part 1: definitions; principles of magnetic recording; tape, disc, and drum stores.
Lecture 8: Storage, part 2: operation, and organisation of core stores; other devices (thin films, flip-flops); complete memory-configuration for a typical computer.
Lecture 9: Design of a small machine, part 1: computers from a user's point of view: choice of word-length (form of an instruction, number-range etc.); how instructions are obeyed.
Lecture 10: Design of a small machine, part 2: function and address-decoding; choice of main store; function-list; design of lator and adder; its speed and cost, using TTL integrated-circuit modules.
Lecture 11: Design of a small machine, part 3: central machine rhytm, and timing of orders; complete arithmetic unit logical connections; input and output devices in general; choice of a particular I/O device; complete machine cost.
Lecture 12: Use of computers: a binary loader for the small machine. Possible design-improvements, including: address-modification; high-level languages; peripheral-handling facilities. General comments; historical notes; course-summary.

The accompanying course-booklet contains: an easy-to-read summary of each lecture; detailed notes and diagrams for each lecture; examples and course-work for each lecture; and bibliography and glossary of terms. There is a further booklet intended for the course-tutor which contains, amongst other things, answer to the exercises.

The 'Logical Design' course is self-contained, though at Manchester it is used in conjunction with the 'Algol' series as described in §2.2 of this paper. In other Universities it has been run, for instance, as part of an Electrical Engineering course, and also by itself as a General Science option.

2.2 *'Programming in Algol'*

This serie is really about problem and the expression of the solutions in Algol 60: there is very little numerical analysis.

The design of the course is based on the following principles:

(i) That students should be encouraged to write and test complete programs from the start and each of the early lectures contains at least one program. Once procedures are introduced emphasis moves to them.

(ii) That it is desirable to produce good programmers rather than just give them some idea of what programming is about. Consequently some stress is placed on procedures and one lecture is devoted to recursion. Another deals with program debugging and efficiency.

(iii) That languages of Algol 60's complexity are best understood by use of formal mechanisms such as BNF. These are used quite widely in the lecture, though it is clearly convenient sometimes to modify or condense definitions, and sometimes to avoid using them.

A detailed summary of the course follows:

Lecture 1: An introduction to the nature of computation; derivation of a computer model; a simple Algol program (to add 3 numbers)

Lecture 2: Transformations of the simple program of Lecture 1 arriving at one which will add any number of numbers;
introduction to BNF.

Lecture 3: Arithmetic – the definition of the <simple arithmetic expression> with examples; some input – output procedures.

Lecture 4: Conditions – the definition of the <simple Boolean> and the <conditional statement> with examples; flow-diagrams.

Lecture 5: Control – the definition of the <for statement> and the <go to statement> with examples.

Lecture 6: Arrays and blocks – the definition of the <block>, the <array declaration> and the <subscripted variable> with examples.

Lecture 7: Complete definition of the<expression>s; comment conventions; fuller discussion of input and output (for Atlas and the ICL 1900 series) including character procedures.

Lecture 8: Introduction to procedures; value and name parameters; body replacement.

Lecture 9: Function procedures; full list of <specifier>s; philosophical justification of procedures.

Lecture 10: Recursion both in recursive procedures and recursive calls: Jensen's device.

Lecture 11: Discussion of what makes a 'good' program; efficiency; debugging.

Lecture 12 A case study.

The associated course booklet is the static equivalent of the dynamic TV lecture. It contains exactly the same information and examples, but makes no reference to the televised lectures. It could, therefore, stand on its own.

Each lecture is followed by a number of short exercises which can be completed in a few minutes each from the information provided by the lecture. There is also a series of problems which can be programmed with the facilities discussed up to that point.

3. THE USE OF THE TWO TELEVISION SERIES

3.1 *Subsidiary Courses*

Subsidiary courses at Manchester run for the full academic year (24 weeks) and are examinable.

A Computer Science subsidiary could take an infinite variety of forms and as time goes on, different courses will evolve. At present we give only one course, which uses both TV series. A three hour period is required each week which is allocated as follows:

½ hour TV lecture
½ hour tutorial/discussion
2 hours programming practical

Programming can commence after 3 lectures (so in the first period of the year two Algol lectures are shown) and this continues throughout the year. The logical design lectures are shown during the second half of the year; there is no associated practical work, though each lecture has a set of questions for discussion.

Students are required to complete 10 Algol programs during the year according to a schedule which allows them a choice for all 10 programs. The programs are marked and this mark contributes towards the final mark.

Although two hours are set aside for writing and debugging program, and obtaining assistance from demonstrators (one member of staff or research student to 10-12 students), students are encouraged to make use of the turn-round from the punching service and from the computer to obtain daily runs if necessary.

3.2 *Service Courses*

Service courses arise in a number of ways, for example as crash courses for new research students, or for undergraduate students whose major course (such as numerical analysis) presupposes some knowledge of Algol. These are usually of very short duration (1 or 2 weeks) and are not examinable. Clearly, very little practical work is possible and these courses need to be followed by courses which enable programming experience to be gained.

4. AUDIENCE-REACTION

Questionaires and simple tests have been conducted both at Manchester and at other Universities, to assess the reaction of the students to the television courses. The main conclusions are summarised below.

4.1 *Feedback on the Television production techniques*

52% of all audiences tested stated that the diagrams and captions were of adequate legibility. The remaining 48% were about evenly distributed between thinking that the clarity was more than adequate, and thinking that it was rather inadequate. 64% of all audiences felt that each television lecture had attempted to cover about the right amount of ground. 26% thought that too much material was being included, and the remaining 10 felt that too little was being covered. There was also a tendency (31%) to state that the pace of the Television lectures was rather fast.

4.2 *Television as a teaching medium: students' viewpoint*

Most of the students whose comments have been sought had not experienced educational television before being exposed to these two Computer Science courses. It was found that the students' first reaction was to express approval of the number of photographs, diagrams, and

film-sequences which appeared during the series. This probably indicates a scarcity of such visual aids in conventional teaching, rather than any superfluity during the television series. The students had been warned that the pace of the lectures demanded their full attention, and although they had been advised to read over the printed lecture-notes before viewing a programme, only about 30% ever did so. Although the booklets were not intended to be read during the actual programme, 55% found it reassuring to have the notes open in front of them as viewed the screen. 69% of all students found the amount of course-documentation, examples, etc., just right, and the remaining 31% were almost evenly distributed between thinking that the course-documentation was more than adequate, or rather sparse.

4.3 *Television as a teaching medium: tutor's viewpoint*

Normally the course-tutor is not the same person as the original presenter. This has sometimes meant that the course-tutor is out of sympathy with the way certain points are described during the television programme. These cases of differing personel emphasis can however be exploited to advantage, with the course-tutor sometimes being able to give a new, and less formal viewpoint. The course-tutor is able to identify himself more with the student and his problems.

A single experiment was tried on a class of 60 students at Manchester to compare the degree of information-recall subsequent to a television programme with that obtaining for a normal blackboard-and-chalk lecture. Lecture 5 of the 'Logical Design' course was chosen for experiment, since this depended least of the series on complicated diagrams, films, apparatus, etc. 30 students were given lecture by the presenter using a normal blackboard-and-chalk technique, whilst the remainder watched the recorded television version. Each group was given a simple memory-test of 8 questions yielding a total maximum score of 15 points. The average percent recall-scores for the two groups immediately after the lecture and 3 weeks later, are given below.

	IMMEDIATE RECALL	3 WEEKS LATER
TELEVISION AUDIENCE	80%	56%
BLACKBOARD AUDIENCE	90%	63%

The blackboard-and-chalk lecture was not typical of its kind, in that it had been very adequately prepared, and it incorporated a visual aid. Thus the experiment tends to show that a good live lecture is preferable, for this kind of material, to a television presentation in that recall-scores are about 10% higher. Television does, of course, have other advantages such as organisational convenience.

It was interesting to note that in the above experiment the television programme presented in 29 minutes material which took 53 minutes to present by conventional blackboard-and-chalk techniques. The difference in time was mostly concerned with the mechanics of note-taking. On comparing the printed notes for Lecture 5 with the versions produced live by the 30 blackboard-and-chalk students, 20% of these students thought the printed notes superior to their own. The remaining 80% thought the two versions contained about the same amount of information.

5. ECONOMICS

The overall production-cost of each 12-lecture course was about £ 5,000. This includes television-staff salaries and studio overheads, but does not include booklet-printing or lecturer's salary. Each course took about a year to produce (overlapped with normal teaching and research duties), and each programme involved about 20 hours of the lecturer's time before rehearsal and shooting could begin. The University of Manchester Television Service is making available the video tapes and course-booklets to other educational institutions. The selling-price of the Logical Design tapes plus 50 copies of the booklet is £ 312.10.0. The booklets cost 12/6d each. A similar price-structure is being arranged for the 'Algol' course, though here the booklets will probably cost 17/6d to £ 1 since they contain much more material.

6. CONCLUSION

The production of these two Television courses has enabled the Department of Computer Science's subsidiary and service teaching to be expanded to meet a growing demand, whilst freeing skilled lecturer effort for other tasks.

7. ACKNOWLEDGEMENTS

The authors wish to thank Professor T. Kilburn of the Department of Computer Science for supporting this project, and the Director and staff of the University of Manchester Television Service for the production of the programmes.

THE USE OF AN INTERACTIVE COMPUTER SYSTEM AND APL IN TEACHING PROBABILITY AND STATISTICS

U.M. von Maydell

1. INTRODUCTION

The teaching of probability and statistics without a digital computer has the obvious disadvantage that the student is not able to perform large scale calculations. The use of a computer alleviates this calculation problem and enables the student to analyse simply large samples of observations. This particular asset of using a computer is not emphasized here. The problem considered here is how a computer can be used to motivate the student to discover by himself some of the theory in probability and statistics. In § §2.1 - 2.3 three examples are given to illustrate the method of stimulating the student's imagination and giving him more insight of both theory and problems. A broad outline of the course (Computing Science 312) is given in §2.

An IBM360/67 computer system running with OS/APL was used. Eighteen typewriter terminals (IBM 2741's) out of sixtyfour were available for students. In this particular course a student was able to use the interactive computer system for one hour per week. Good supervision of these computer-terminal sessions was found to be essential. Even though the students had not been exposed to the APL language before, they were able to program APL functions after three terminal sessions and became quite proficient in APL after about ten terminal sessions.

The two main assets of APL as far as statistics is concerned are that it is an array-oriented language allowing for the easy manipulation of arrays in an arbitrary number of dimensions, and that the query operator ? (see Lewis **et al.** (1969)) provides a simple means of generating random integers from a uniform distribution (see §2.1). The ease of manipulating arrays gives incentive to the student to try various combinations of operators and explore arrays of more than two dimensions. One of the disadvantages of the APL system for computing science students is the fact that APL functions may take an undue amount of storage or execution time. Since the main purpose of the use of computers in this course is to encourage students to design algorithms and to experiment with them, this disadvantage is considered to be of minor importance.

The definitions of the APL operators are given in the Falkoff and Iverson (1968) manual and are reduced in tables 1 and 2. An introduction to the APL language with some illustrations of its use in elementary statistical analysis is given in the Smillie (1970) manual.

2. TEACHING METHOD

The course for which this particular method was developed is a first course in probability and statistical inference for second year undergraduate university students in science. The required prerequisites are linear algebra and calculus. No knowledge of programming was assumed. The course deals with the concepts of probability, discrete and continuous random variables and their distributions, some sampling theory, estimation, and hypothesis testing, (see von Maydell (1970)). Thus the course content is comparable to most of the topics in Freeman (1963) and Freund (1962). Distributions of various random variables are emphasized in order to assure a strong background for sampling distributions and test statistics.

As mentioned in §1 the prime aim of this teaching method is to motivate the student to discover by himself some of the theory, some of the difficulties in solving practical problems, and some of the necessary methods. In the following §§2.1 - 2.3 three different ways are described in which an interactive computer system, particularly with APL, can assist the student to discover various aspects of the theory of probability and statistical inference.

2.1 *Probability*

One of the most difficult tasks in teaching the concepts of probability is to convey to the student the empirical basis of the subject. If a random experiment is repeated many times under the same circumstances the relative frequency of each of its outcomes tends to a stable value, which in the hypothetical case of an infinite number of repetitions can be defined as probability. With an interactive computer system the student can simulate an experiment and repeat it as often as he likes. The ? operator in APL greatly simplifies the simulation of repeated experiments and the student can easily observe the 'long-run stability' of relative frequencies.

As an example let us consider a thumbtack tossing experiment, in which five identical thumbtacks are tossed and the outcome is the number of upward-pointing tacks. The operation ?5 results in one number out of the set (1,2,3,4,5), where each one of these numbers has the same chance of being selected. To simulate the tossing of one thumbtack such that it has an arbitrarily chosen chance of 20%, say, of pointing upward, we are interested only in the occurrence or non-occurrence of one of the five integers. Hence,

```
1 = ?5
```

or

```
0.
```

where 1 corresponds to the thumbtack pointing upward and 0 to the thumbtack lying sideways. A toss of five thumbtacks is achieved by

```
1 + ?5⍴5
```

which, for example, may result in

```
1 1 0 0 1 .
```

Then the number of upward points on one toss of five thumbtacks may result in

```
+/1 = ?5⍴5  ↔  3.
```

To simulate M repetitions of this experiment, the expression

```
+/1 = ?(M,5)⍴5
```

yields a vector of M elements, which are numbers from the set (0,1,2,3,4,5)=(number of upward points). For M=10

```
+/1 = ?(10,5)⍴5
```

may result in

```
1 0 0 2 1 1 3 0 1 1 .
```

The frequencies corresponding to 0,1,2,3,4,5 points upward in M repetitions of the experiment are given by the vector

```
V ← +/(0,⍳5)∘.=+/1 = ?(M,5)⍴5.
```

for example,

```
V ↔ 3 5 1 1 0.
```

Finally, the frequency and relative frequency corresponding to the values 0,1,2,3,4,5 can be expressed in a 3 x 6 matrix by the following APL function:

```
     ∇ F ← THUMBTAC M;V
[1]    V ← +/(0,⍳5)∘.=+/1=?(M,5)⍴5                    (2.1)
[2]    F ← ⍉(3,6)⍴(0,⍳5),V,V÷M
       ∇
```

For instance,

```
THUMBTAC 10  ↔  0   3   0.3
                1   5   0.5
                2   1   0.1
                3   1   0.1
                4   0   0
                5   0   0
```

The algorithm (2.1) can be expanded to handle large values of M and to include output headings as shown in algorithm (A.1) in the appendix.

Relatively frequencies from M=50 and 10000 repetitions are given in the Appendix (A.2 and A.3). Thus having designed the algorithm himself the student knows exactly what is meant by repeating an experiment many times.

In order to study the stabilizing effect as M increases, the variation in relative frequencies for a given number of upward points and a given M must be considered. Suppose a set of M repetitions of the experiment is repeated N times, then the range of the N relative frequencies indicates how widely these relative frequencies vary for a given number of upward points. Let D denote a matrix of relative frequencies such that the first column contains the number of upward thumbtacks, 0,1,2,3,4,5 and the rest of the elements in each row are the corresponding N relative frequencies for a given M value. Then the maximum relative frequency for each of the six values can be expressed as the vector.

```
⌈/D[;1+⍳N]
```

and, similarly, the vector of minimum relative frequency for each of the six values is

```
⌊/D[;1+⍳N].
```

The range of relative frequencies for each of the values is simply the difference of these two vectors. An APL function giving a matrix of the six values and the corresponding relative frequency ranges is

```
     ∇ R ← RELFRANGE D;N
[1]    N ← (⍴D)[2]-1
[2]    R ← ⍉(2,6)⍴,D[;1],(⌈/D[;1+⍳N])-⌊/D[;1+⍳N]
       ∇
```

For N=10 repeated sets of M=50, and 10000 repetitions of the experiment the relative frequency ranges were found to be

number of upward points	rel. freq. ranges for M=50	rel. freq. ranges for M=10000
0	0.10	0.0110
1	0.20	0.0108
2	0.12	0.0053
3	0.10	0.0088
4	0.04	0.0037
5	0	0.0008

Hence, the student can conclude that in the long-run the relative frequencies for different values tend to stabilize to certain constants, that these constants lie in the interval (0,1), and that they all add up to one (see output (A.3)). Thus not only has the student illustrated the empirical basis of probability, but he also has discovered some of the probability axioms.

2.2 *Random variables*

In §2.1 the numerical results helped the student to visualize part of probability theory. In this section we indicate how the actual design of the algorithm helps to clarify some of the theory of random variables.

Let us consider the addition of two independent, discrete random variables. Then the distribution of this sum poses two problems, that of finding the value set of the new random variable, and that of determining the corresponding probabilities. To illustrate the algorithm with a simple example, let the random variable X have the value set (x)=(−1,1) with P(X=x)=½ and Y have the value set (y)=(−2,0,2) with P(Y=y_j)=$^1/_3$. The value of the random variable T=X+Y are formed by considering the cartesian product of (x) and (y) and adding the values in each pair. Thus for

```
X ← ¯1  1
Y ← ¯2  0  2
```

the matrix of all possible values is obtained from the outer product

```
X∘.+Y  ↔  ¯3  ¯1  1
          ¯1   1  3 .
```

Forming a vector of these values, we obtain

```
V ← , X∘.+Y ↔ ¯3 ¯1  1 ¯1  1  3 .          (2.2)
```

which in ascending order is

```
V[⍋V]  ↔  ¯3 ¯1 ¯1  1  1  3                (2.3)
```

Then

```
1,(1↓V[⍋V])≠ ¯1 ↓ V[⍋V]  ↔  1  1  0  1  0  1
```

where the zeros indicate the positions of duplicate values in the vector (2.3). Thus

```
1  1  0  1  0  1/V[⍋V] ↔  ¯3 ¯1  1  3          (2.4)
```

gives the set of distinct values of (2.2), which is the value set of T. The complete APL function, VALSET, for finding value sets of the sum of two random variables is given in (A.4). Finally, to obtain the corresponding probabilities for T, let

```
PX ← 2⍴ .5
PY ← 3⍴ 1÷3 .
```

In order to specify the duplicate values in the vector (2.2) and add the corresponding probabilities, the vector

```
,PX∘.×PY ↔ .1667  .1667  .1667  .1667  .1667  .1667
```

should be multiplied by the matrix

```
B ← (,X∘.+Y)∘.=(X VALSET Y)

B ↔ 1  0  0  0
    0  1  0  0
    0  0  1  0
    0  1  0  0
    0  0  1  0
    0  0  0  1
```

Thus the vector

```
(,PX∘.×PY)+.×B ↔ .1667  .3333  .3333  .1667
```

gives the corresponding probabilities of the value of T in (2.4). The complete algorithm, SUMRV, for the distribution of T is given in (A.5) and the corresponding output in (A.6).

In order to design this algorithm the student must define precisely the addition operation of two random variables. Thus, he is well aware that this operation is not an arithmetic sum. He also can see from this example that the sum of two uniform random variables does not have a uniform distribution. This method can easily be extended to add more than two random variables and, hence, enable the student to determine that the sum of binomial (or Poisson) random variables is itself a binomial (or Poisson) random variable. If the random variables have the same distribution, the student also can discover the central limit theorem on his own. This algorithmic approach of adding several discrete random variables seemed to facilitate a better visualization of the convolution theory in adding continuous random variables.

2.3 *Statistical inference*

Another topic with which students often have difficulties is sampling distributions. Without a computer the selection of large samples, their analysis and repeated sampling is very difficult. The interactive computer system not only enables the student to easily generate samples, set up frequency tables and calculate moments, but the immediate output also encourages him to try various sample sizes and observe the variation in sample moments. He then can test whether or not the sample statistics confirm a hypothesized population distribution. Repeated sampling from a given distribution can of course be used to illustrate the normal behaviour of the sample means. On the other hand, the different sized samples will show the student how the amount of information for more reliable results increases with larger samples.

In order for a student to determine the distribution of an unknown population from sample observations, APL functions, simulating different distributions from which samples can be generated were 'locked' in an APL library from where they can be used but not seen. Thus samples of different sizes could be obtained without disclosing the values of the distribution parameters. Consider the APL function in (A.7),

```
DISTN3  N,
```

which generates N values of a random variable with unknown distribution. If N=10, then the vector

```
R ← DISTN3  10
```

may have the values

```
R ↔ 6  5  3  4  7  2  4  5  5  6.
```

Immediately, the student will class the random variable as a discrete random variable with integer values. For a sample of size N=500, then

```
R ← DISTN3  500
```

The maximum and minimum observed values could be

```
⌈/R ↔ 13
⌊/R ↔ 0 ,
```

respectively. With the APL function (A.8) a frequency table can be set up for this example

```
¯.5  1  14  FREQ  R

¯0.5      0.5      0      4
 0.5      1.5      1     16
 1.5      2.5      2     36
 2.5      3.5      3     87
 3.5      4.5      4     78
 4.5      5.5      5     81          (2.5)
 5.5      6.5      6     64
 6.5      7.5      7     59
 7.5      8.5      8     40
 8.5      9.5      9     16
 9.5     10.5     10      8
10.5     11.5     11      5
11.5     12.5     12      3
12.5     13.5     13      3
```

The sample mean and variance were found to be

```
SAMEAN R ↔ 5.074
SAVAR  R ↔ 5.477
```

At first the student will probably class this distribution as binomial. Upon further sampling he may realize that the mean and variance are nearly equal, and, hence, conjecture that the population has a Poisson distribution with parameter value of about five. At this stage, in order to check his conclusion, the student is encouraged to invent some test of his own to compare the conjectured distribution with observations from another sample. This motivation can be used to introduce the chi-square test. Thus this example also has introduced the student to the theory of estimation and hypothesis testing.

3. CONCLUSION

This teaching method was used here at the university for the last three years in which time the number of students taking the course increased from 30 to 130. With this approach the student seems to have a much better understanding of, for example, random sampling, the addition operator in sums of random variables, the empirical basis and axioms of probability, sampling distributions for various sample sizes, estimation, and hypothesis testing. The students, themselves, have commented that the laboratory problems and terminal sessions clarify the theory and also point out strategies for solving practical problems. Some students have said: 'this is great, now I can really see what's going on.'

As indicated in §2 the approach is intended to be open ended, so that an inquisitive student can explore various given topics in as much detail as he likes, or even develop some theorems or methods on his own. In general the mere fact that the student deals with actual numbers and sees the immediate results in the typewriter output makes the theory and problems more meaningful and less abstract. The ease of programming in APL is a great asset in developing involved and difficult algorithms. In writing APL functions the APL-operator definitions, themselves, assist the student in obtaining a general perspective of the problems and hence promote the design of algorithms to solve classes of problems.

This teaching method may be classed under 'computer-assisted-instruction' (CAI) in the broad sense, but clearly does not belong to the set of software packages designed to teach students by the 'question-and-answer method' usually referred to as CAI. The method introduced here is not as restricted as the usual CAI method and hence is more likely to provide the student with incentive to learn.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the assistance of Mr. G. Links, Mr. D. Lougheed and Mr. G. McCalla for designing many of the algorithms and functions used in this course.

Table 1 : Definitions of APL Operators
(Primitive Scalar Functions).

Monadic form fB		f	Dyadic form AfB	
Definition or example	Name		Name	Definition or example
$+B \leftrightarrow 0+B$	Plus	+	Plus	$2+3.2 \leftrightarrow 5.2$
$-B \leftrightarrow 0-B$	Negative	-	Minus	$2-3.2 \leftrightarrow \bar{}1.2$
$\times B \leftrightarrow (B>0)-(B<0)$	Signum	×	Times	$2\times 3.2 \leftrightarrow 6.4$
$\div B \leftrightarrow 1\div B$	Reciprocal	÷	Divide	$2\div 3.2 \leftrightarrow 0.625$
B | $\lceil B$ | $\lfloor B$ 3.14 | 4 | 3 $\bar{}$3.14 | $\bar{}$3 | $\bar{}$4	Ceiling Floor	⌈ ⌊	Maximum Minimum	$3\lceil 7 \leftrightarrow 7$ $3\lfloor 7 \leftrightarrow 3$
$*B \leftrightarrow (2.71828..)*B$	Exponential	*	Power	$2*3 \leftrightarrow 8$
$\circledast N \leftrightarrow N \leftrightarrow *\circledast N$	Natural logarithm	⍟	Logarithm	$A\circledast B \leftrightarrow$ Log B base A $A\circledast B \leftrightarrow (\circledast B)\div\circledast A$
$\lvert\bar{}3.14 \leftrightarrow 3.14$	Magnitude	|	Residue	Case | $A\lvert B$ $A\neq 0$ | $B-(\lvert A)\times\lfloor B\div\lvert A$ $A=0, B\geq 0$ | B $A=0, B<0$ | Domain error
$!0 \leftrightarrow 1$ $!B \leftrightarrow B\times !B-1$ or $!B \leftrightarrow$ Gamma$(B+1)$	Factorial	!	Binomial coefficient	$A!B \leftrightarrow (!B)\div(!A)\times !B-A$ $2!5 \leftrightarrow 10$ $3!5 \leftrightarrow 10$
$?B \leftrightarrow$ Random choice from ιB	Roll	?	Deal	A Mixed Function (See Table 2).
$\circ B \leftrightarrow B\times 3.14159...$	Pi times	○	Circular	See Table at left
$\sim 1 \leftrightarrow 0$ $\sim 0 \leftrightarrow 1$	Not	~		
		∧	And	
		∨	Or	
		⍲	Nand	
		⍱	Nor	
		<	Less	
		≤	Not greater	
		=	Equal	
		≥	Not less	
		>	Greater	
		≠		

A	B	$A\wedge B$	$A\vee B$	$A\barwedge B$	$A\barvee B$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	0

Relations
Result is 1 if the relation holds, if it does not:
$3\leq 7 \leftrightarrow 1$
$7\leq 3 \leftrightarrow 0$

$(-A)\circ B$	A	$A\circ B$
$(1-B*2)*.5$	0	$(1-B*2)*.5$
Arcsin B	1	Sine B
Arccos B	2	Cosine B
Arctan B	3	Tangent B
$(\bar{}1+B*2)*.5$	4	$(1+B*2)*.5$
Arcsinh B	5	Sinh B
Arccosh B	6	Cosh B
Arctanh B	7	Tanh B

Table of Dyadic ○ Functions

Reprinted by permissi~n :RSON [1968].

Table 2 : Definitions of APL Operators (Primitive Mixed Functions).

Name	Sign[1]	Definition or example[2]
Size	ρA	$\rho P \leftrightarrow 4 \quad \rho E \leftrightarrow 3\ 4 \quad \rho 5 \leftrightarrow \iota 0$
Reshape	$V\rho A$	Reshape A to dimension V $\quad 3\ 4\rho\iota 12 \leftrightarrow E$ $\quad 12\rho E \leftrightarrow \iota 12 \quad 0\rho E \leftrightarrow \iota 0$
Ravel	$,A$	$,A \leftrightarrow (\times/\rho A)\rho A \quad ,E \leftrightarrow \iota 12 \quad \rho,5 \leftrightarrow 1$
Catenate	V,V	$P,\iota 2 \leftrightarrow 2\ 3\ 5\ 7\ 1\ 2 \quad 'T','HIS' \leftrightarrow 'THIS'$
Index[3 4]	$V[A]$ $M[A;A]$ $A[A;..;A]$	$P[2] \leftrightarrow 3 \quad P[4\ 3\ 2\ 1] \leftrightarrow 7\ 5\ 3\ 2$ $E[1\ 3;3\ 2\ 1] \leftrightarrow \begin{matrix} 3 & 2 & 1 \\ 11 & 10 & 9 \end{matrix}$ $E[1;] \leftrightarrow 1\ 2\ 3\ 4 \quad E[;1] \leftrightarrow 1\ 5\ 9 \quad 'ABCDEFGHIJKL'[E] \leftrightarrow \begin{matrix} ABCD \\ EFGH \\ IJKL \end{matrix}$
Index generator[3]	ιS	First S integers $\quad \iota 4 \leftrightarrow 1\ 2\ 3\ 4$ $\iota 0 \leftrightarrow$ an empty vector
Index of[3]	$V\iota A$	Least index of A in V, or $1+\rho V$ $\quad P\iota 3 \leftrightarrow 2 \quad P\iota E \leftrightarrow \begin{matrix} 5 & 1 & 2 & 5 \\ 3 & 5 & 4 & 5 \end{matrix} \quad 4\ 4\iota 4 \leftrightarrow 1 \quad \begin{matrix} 5 & 5 & 5 & 5 \end{matrix}$
Take Drop	$V\uparrow A$ $V\downarrow A$	Take (drop) $\lvert V[I]$ first elements on coordinate I. (Last if $V[I]<0$) $\quad 2\ 3\uparrow X \leftrightarrow \begin{matrix} ABC \\ DEF \end{matrix} \quad {}^{-}2\downarrow P \leftrightarrow 5\ 7$
Grade up[5] Grade down[5]	$\grave{A}$ $\acute{A}$	The permutation which would order A (ascending or descending) $\quad \grave{}3\ 5\ 3\ 2 \leftrightarrow 4\ 1\ 3\ 2 \quad \acute{}3\ 5\ 3\ 2 \leftrightarrow 2\ 1\ 3\ 4$
Compress[5]	V/A	$1\ 0\ 1\ 0/P \leftrightarrow 2\ 5 \quad 1\ 0\ 1\ 0/E \leftrightarrow \begin{matrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{matrix}$ $1\ 0\ 1/[1]E \leftrightarrow \begin{matrix} 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \end{matrix} \quad 1\ 0\ 1\not E$
Expand[5]	$V\backslash A$	$1\ 0\ 1\backslash\iota 2 \leftrightarrow 1\ 0\ 2 \quad 1\ 0\ 1\ 1\ 1\backslash X \leftrightarrow \begin{matrix} A & BCD \\ E & FGH \\ I & JKL \end{matrix}$
Reverse[5]	ϕA	$\phi X \leftrightarrow \begin{matrix} DCBA \\ HGFE \\ LKJI \end{matrix} \quad \phi P \leftrightarrow 7\ 5\ 3\ 2 \quad \phi[1]X \leftrightarrow \ominus X \leftrightarrow \begin{matrix} IJKL \\ EFGH \\ ABCD \end{matrix}$
Rotate[5]	$A\phi A$	$3\phi P \leftrightarrow 7\ 2\ 3\ 5 \leftrightarrow {}^{-}1\phi P \quad 1\ 0\ {}^{-}1\phi X \leftrightarrow \begin{matrix} BCDA \\ EFGH \\ LIJK \end{matrix}$
Transpose	$V\mathbin{⍉}A$ $\mathbin{⍉}A$	Coordinate I of A becomes coordinate $V[I]$ of result $\quad 2\ 1\mathbin{⍉}X \leftrightarrow \begin{matrix} AEI \\ BFJ \\ CGK \\ DHL \end{matrix} \quad 1\ 1\mathbin{⍉}E \leftrightarrow 1\ 6\ 11$ Transpose last two coordinates $\quad \mathbin{⍉}E \leftrightarrow 2\ 1\mathbin{⍉}E$
Membership	$A\epsilon A$	$\rho W\epsilon Y \leftrightarrow \rho W \quad E\epsilon P \leftrightarrow \begin{matrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{matrix}$ $P\epsilon\iota 4 \leftrightarrow 1\ 1\ 0\ 0 \quad \begin{matrix} 0 & 0 & 0 & 0 \end{matrix}$
Decode	$V\perp V$	$10\perp 1\ 7\ 7\ 6 \leftrightarrow 1776 \quad 24\ 60\ 60\perp 1\ 2\ 3 \leftrightarrow 3723$
Encode	$V\top S$	$24\ 60\ 60\top 3723 \leftrightarrow 1\ 2\ 3 \quad 60\ 60\top 3723 \leftrightarrow 2\ 3$
Deal[3]	$S?S$	$W?Y \leftrightarrow$ Random deal of W elements from ιY

Reprinted by permission from FALSOFF and IVERSON [1968]

APPENDIX

The following APL functions and output were referred to in §2.

```
    ∇ TH←THUMBTACK N;TH1;TH2;NO;N1;J
[1]   NO←N
[2]   TH1←5
[3]   TH2←(TH1+1)ρ0
[4]   N1←N
[5]   J←0
[6]   →(N≤1400)/9
[7]   N←N-N1←1400                                          (A.1)
[8]   J←1
[9]   TH2←TH2++/(0,⍳TH1)∘.=+/1=?(N1,TH1)ρTH1
[10]  →(J=1)/4
[11]  'NO. OF TRIALS OF TOSSING ';TH1;' THUMBTACKS IS: ';NO
[12]  'THE PROBABILITY OF A THUMBTACK POINTING UP IS: ';÷TH1
[13]  'POSSIBLE      FREQ. OF     RELATIVE'
[14]  'OUTCOMES      OUTCOMES     FREQUENCY'
[15]  TH←⍉(3,TH1+1)ρ(0,⍳TH1),TH2,TH2÷NO
    ∇
```

Statement (9) in (A.1) contains the main algorithm discussed in §2.1.

```
      THUMBTACK 50
NO. OF TRIALS OF TOSSING 5 THUMBTACKS IS: 50
THE PROBABILITY OF A THUMBTACK POINTING UP IS: 0.2
POSSIBLE      FREQ. OF     RELATIVE
OUTCOMES      OUTCOMES     FREQUENCY

   0            17           0.34
   1            17           0.34
   2            15           0.3
   3             1           0.02                          (A.2)
   4             0           0
   5             0           0
```

```
      THUMBTACK 10000
NO. OF TRIALS OF TOSSING 5 THUMBTACKS IS: 10000
THE PROBABILITY OF A THUMBTACK POINTING UP IS: 0.2
POSSIBLE      FREQ. OF     RELATIVE
OUTCOMES      OUTCOMES     FREQUENCY                       (A.3)

 0.000000E0    3.289000E3   3.289000E¯1
 1.000000E0    4.106000E3   4.106000E¯1
 2.000000E0    2.030000E3   2.030000E¯1
 3.000000E0    4.930000E2   4.930000E¯2
 4.000000E0    7.200000E1   7.200000E¯3
 5.000000E0    4.000000E0   4.000000E¯4
```

```
    ∇ F←X VALSET Y;V;A
[1]   F←(1,(1+A)*¯1+A)/A←V[⍋V←,X∘.+Y]                      (A.4)
    ∇
```

```
    ∇ F←S SUMRV T;V
[1]   F←⍉(2,ρV)ρV,..((V←S[1;] VALSET T[1;])∘.=,S[1;]∘.+
      T[1;])+.×,S[2;]∘.×T[2;]                              (A.5)
    ∇
```

For

```
      S←(2,2)ρ¯1.1,.5,.5

      T←(2,3)ρ¯2,0.2,3ρ÷3

      S SUMRV T                                            (A.6)

 ¯3        0.1667
 ¯1        0.3333
  1        0.3333
  3        0.1667
```

```
    ∇ R←DISTN3 N                                           (A.7)
[1]   R←+/((?(N⍴100000))÷100000)∘.≥+/((17 17)ρ((
      5÷0,⍳16)*¯5)÷!0,⍳16)×(⍳17)∘.≥17
    ∇
```

The algorithm (A.7) generates N values of a Poisson random variable with parameter value equal to five. The inverse of the distribution function is used and probabilities of values greater than 16 are considered to be zero.

```
    ∇ T←P FREQ X
[1]   T←(P[3],4)ρ0                                         (A.8)
[2]   T[;1]←P[1]+P[2]×(⍳P[3])-1
[3]   T[;2]←T[;1]+P[2]
[4]   T[;3]←T[;1]+P[2]÷2
[5]   T[;4]←+/(⍳P[3])∘.=(X-P[1])÷P[2]
    ∇
```

In the algorithm (A.8) the parameter X represents the vector of values to be grouped into a frequency table and the parameter P is a vector of order three. The elements of P are the starting point of the frequency table, the interval size and the number of intervals, respectively. The output T is a four-column matrix with columns 1,2 and 3 giving the starting point, the end point and the midpoint of the intervals, and column 4 representing the frequency of values occuring in each interval (see (2.5)).

```
    ∇ R←SAMEAN X
[1]   R←(+/X)÷ρX                                           (A.9)
    ∇
```

```
    ∇ R←SAVAR X
[1]   R←((+/X*2)÷ρX)-(SAMEAN X)*2                          (A.10)
    ∇
```

REFERENCES

Falkoff, A.D., and K.E. Iverson, APL/360 User's Manual, IBM Corporation, Yorktown Heights, N.Y., 1968.

Freeman, H., 1963, Introduction to Statistical Inference, (Addison-Wesley, Reading, Mass.)

Freund, J.E., 1962, Mathematical Statistics (Prentice-Hall, Englewood Cliffs, N.J.)

Lewis, P.A.W., A.S. Goodman, and J.M. Miller, A Pseudo-Random Number generater for the System/360, IBM System Journal, 8, No. 2 (1969), pp. 136-146.

von Maydell, U.M., An Introduction to Probability and Statistics Using APL, Department of Computing Science, Publication No. 21, University of Alberta, Edmonton, Alberta, 1970.

Smillie, K.W., An Introduction to APL 360 with Some Statistical Applications, Department of Computing Science, Publication No. 19, University of Alberta, Edmonton, Alberta, 1970.

AN OPTIMAL STRATEGY FOR COMPUTER EDUCATION

T.T. Moskova and A.T. Moskov

1. INTRODUCTION

The application of computers for solving complicated problems, optimizations, and for increasing the accuracy of solutions, spans more and more areas and, with the development of modern mathematical methods, has particularly increased. This has resulted in the need for well-trained specialists for computer centres (CC) and at least an acquaintance for the remaining specialists such as engineers, economists, etc., with computer science.

The general formulation of the problem, the topic of this paper, is to propose an optimal strategy for computer education (E), showing the way to assure specialists necessary for implementation of (CC) what kind of procedure has to be undertaken by the others in order to obtain maximum efficiency of education.

The solution to this problem can be reached on the basis of a stochastic self-justifying system, with the automatic control object, the education system (E).

2. PROBLEM DESCRIPTION

Let us suppose that the structure of the system E is as shown in Fig. 2.1. The meaning of the symbols is as follows:

- E – Computer education system
- E_F – Facultative education on computers, using courses
- E_I – Education by insertion of special subjects into the school syllabus
- E_S – Education by specialized schools on computers
- $E^{(m)}, E^{(h)}$ – Education of medium to highly qualified specialists
- $E^{(h)}_{F,C}$ – Facultative education of higher specialists for CC
- $E^{(h)}_{F,n_j}$ – Facultative education of higher specialists of the j-th subject not closely connected with the CC, for j=1,2,...,k.
- $E^{(h)}_{I,j}$ – Education of the type E_I of higher specialists for the j-th subject,(j=1,2,...,k).

In order to establish the corresponding mathematical model, it is assumed that each form of computer education has, as a main parameter, the number of the students n, with appropriate subscript (resp. superscript) at each standard.

Because of the changes of the computer system, the system of education E has to be flexible enough in order to react on time to these changes.

In this paper, the optimal control of E is carried out by the help of a stochastic, self-justifying, discrete, non-interrupted system, with a time-interval of one year. The corresponding scheme is shown in Fig. 2.2.

As is clear from the scheme, Y-controlling block, $\bar{v}_t = \left(n_F^{(m)}, n_{F,C}^{(h)}, n_{F,n_1}^{(h)}, \ldots, n_{F,n_k}^{(h)}, n_I^{(m)}, n_{I,1}^{(h)}, \ldots, n_{I,k}^{(h)}, n_S^{(m)}, n_S^{(h)}\right)$ control vector for a certain value of the discrete parameter t (years), E the object of the control, the system of education, (see Fig. 2.1), characterized by so-called „pure delay", reflecting the period of the study, $\bar{\tau}$ -the vector of the „pure delay" for the different forms of education, $\bar{y}_{t+\tau} = \left(n_F^{(m)}, n_{F,C}^{(h)}, n_{F,n_1}^{(h)}, \ldots, n_{F,n_k}^{(h)}, n_I^{(m)}, n_{I,1}^{(h)}, \ldots, n_{I,k}^{(h)}, n_S^{(m)}, n_S^{(h)}\right)$ the output vector of the object, whose components form the number of graduated students, $\bar{\gamma}, \bar{\delta}$ the vectors of the random disturbing effects, respectively, on E and Y; $\bar{x}_t$ the input action characterizing the aim of the control.

The choice of the optimal strategy for computer education is reduced to define the control vector $\bar{v}_t$ optimal in a certain sense. As an optimal criterion is supposed to be the maximum of the efficiency function $H(\bar{v}_t)$, it can be defined in the following way:

$$H(\bar{v}_t) = W(\bar{v}_t) - L(\bar{v}_t)$$

where $W(\bar{v}_t)$ is the optimal profit for the state of certain periods as a result of the machine-solving of the problems, instead of manually, $L(\bar{v}_t)$ the total expenses for the same period are on the basis of the chosen strategy V_t.

3. PROBLEM SOLVING

3.1 *The function of the total profit*

It is supposed that the total profit W is coming mainly from the CC. Mathematically, this is:

$$W(\bar{v}_t) = \sum_{i=1}^{N_c} W_i(\bar{v}_t) \tag{3.1}$$

where W_i is the profit from the i-th CC, and N_c is the total number of the CC's. Let us assume that the vector of the „pure delay" consists of different value components τ, corresponding to the different forms of education, and $\tau_{max} = \sup\tau$. In this case, the yearly profit functions $W_i(s)$ will depend on the control action $\bar{V}_t$ for the period $t+1 \leqslant S \leqslant t+\tau_{max}+1$. This results in that each of the functions W_i is defined on the basis of the entire period, or:

$$W_i(\bar{v}_t) = \sum_{s=t+1}^{t+\tau_{max}+1} W_i(\bar{v}_t, s) \tag{3.2}$$

The defining of the yearly profit function $W_i(\bar{v}_t, s)$ can be obtained as a result of the following considerations:

Let us suppose the problems to be carried out in the i-th CC are considered as separated realizations A_{ij} of the random event A_i which can be defined, for instance, by the following parameters: W_i- the profit for the State at a problem, solved by the computers, instead of manually, t_{ai} the time necessary for the algorithm to be prepared to make the programs, etc., t_{mi} the machine-time for problem solving.

So the random event A_i is characterized by the density of the probability $f_i(w_i, t_{ai}, t_{mi})$. In the timetable for loading the computer, the problems A_{ij} are considered as not equally probable. A special group, say, the group for analysis in every CC makes a selection of the problems depending, for instance, on W_i and t_{ai}/t_{mi}. Consequently,

the analysis group appears to be a kind of functional transformer for the density f_i of a form:

$$f_{ci}(w_i,t_{ai},t_{mi}) = k_i.\xi_i(t_{ai}/t_{mi},w_i).f_i(w_i,t_{ai},t_{mi}) \quad (3.3)$$

In this case, $\xi_i(w_i,t_{ai}/t_{mi})$ is a positive function of the weight for the problems selection and has a value between 0 and 1, k_i is the coefficient, which can be defined so:

$$k_i = \frac{1}{\int_{\Omega_i} \xi_i(w_i,t_{ai}/t_{mi}).f_i(w_i,t_{ai},t_{mi})d\Omega_i} \quad (3.4)$$

where $\Omega i=\Omega_i(t_{ai},t_{mi},W_i)$ is the set of the feasible meanings of m_i and t_{ai},t_{mi}.

The yearly profit realized, W_i by the i-th CC, if the realizations are independent (as a restrictive condition) can be defined by the expression:

$$W_i = \sum_{j=1}^{m_i} w_{ij} = m_i M[w_i] =$$

$$= m_i \int_{\Omega_i} w_i.f_{ci}(w_i,t_{ai},t_{mi})d\Omega_i \quad , \qquad 0 < m_i \le m_{i\,max} \quad (3.5)$$

where $M[w_i]$ is the mathematical expectation of W_i,m_i is the number of the problems solved yearly, m_{imax} the capacity of the computer in the i-th CC.

It is necessary to notice that because of the comparatively big values of m_i, the approximation introduced in (3.5) is completely permitted, PUGATCHOV 1960 (Ref. 3).

The variable m_i is stochastically dependent on the number of specialists $\hat{n}_{ci}$, working in CC_i and on the set of the moments $\bar{u}_i$ of the random variables W_i,t_{ai},t_{mi} as well. This relation is a hyperplan:

$$m_i = m_i(\hat{n}_{ci},\bar{u}_i) \quad (3.6)$$

which can be described by the help of an equation of a non-linear regression. In special cases $\bar{u}_i$ can consist of mathematical expectations only, that is: $M[t_{ai}]$ and $M[t_{mi}]$.

The maximal value of m_{imax} can be defined by the yearly loading-up T_e at a specialist and $M[t_{ai}]$ as well, according to the expression:

$$m_{i\,max} = (\hat{n}_{ci}.T_s)/M[t_{ai}] \le T_{im}/M[t_{im}] \quad (3.7)$$

where T_{im} is the total maximum yearly usage (in hours) of the computer in certain CC. Fig. 3.1 and Fig. 3.2, the curves $\xi_i=\xi_i(t_{ai}/t_{mj})$ for $W_i=C^{te}$ and $W_i=W_i(\hat{n}_{ci})$ for certain sets $\bar{u}_i$ are shown. Here, $W_{imax}=M_{jmax}$, $M[W_i]$.

To define the distribution of the number of specialists necessary say, $\hat{n}_{ci}$ to work in the CC's, between high and middle grade $\hat{n}_{ci}^{(h)}$ and $\hat{n}_{ci}^{(m)}$ on one hand, and those who are not directly working with the computers and who have to be educated on them somehow $\hat{n}_n^{(h)}$ on the other, this can be done on the basis of the density of probability $f_i(t_{ai}, t_{mi}, W_i)$. This influence is shown in Fig. 3.3, where the curves are $f_i(t_{ai}, t_{mi}, W_i)$ for $t_{mj}=C^{te}$ and $W=C^{te}$

The curve 1 on the figure is built up for $\hat{n}_n^{(h)}=\hat{n}_{n(1)}^{(h)}$ and the curve 2 represent the relation for $\hat{n}_n^{(h)}=\hat{n}_{n(2)}^{(h)}>\hat{n}_{n(1)}^{(h)}$.

Similar are the curves of the density f_i for $\alpha_i=\alpha_{i,1}$ and $\alpha_i=\alpha_{i,2}>\alpha_{i,1}$ where $\alpha_i=\hat{n}_{ci}^{(h)}/\hat{n}_{ci}^{(m)}$. Mathematically this could be expressed by the conditional density of the probability:

$$f_{i_1\,con}(t_{ai},t_{mi},w_i) = f_i\left[t_{ai},t_{mi},w_i \middle| \alpha_i,\left(\hat{n}_{n1}^{(h)}\right)_i,\ldots,\left(\hat{n}_{nk}^{(h)}\right)_i,\left(\hat{n}_{n1_\Sigma}^{(h)}\right)_i,\ldots,\left(\hat{n}_{nk_\Sigma}^{(h)}\right)_i\right]$$

In this expression $\left(\hat{n}_{n1_\Sigma}^{(h)}\right)_i,\ldots,\left(\hat{n}_{nk_\Sigma}^{(h)}\right)_i$ (3.8)

is the number of specialists on different subjects whose problems are being solved on computers in the CC's, and

$$\left(\hat{n}_{n1}^{(h)}\right)_i,\ldots,\left(\hat{n}_{nk}^{(h)}\right)_i$$

the number of specialists, who have been educated on computers previously.

Practically, there will be problems to be solved on computers in the CC's and they are carried out by limited numbers of specialists. That is the reason why t_{ai},t_{mi} and W_i appear to be dependent only on the part of the parameters in the expression of the conditional density. In this case, (3.8) can be presented in the following form:

$$f_{i_1\,con}(t_{ai},t_{mi},w_i) = f_i(t_{ai},t_{mi},w_i \middle| \alpha_i,\hat{\bar{s}}_i,\hat{\bar{s}}_{i,\Sigma}) \quad (3.9)$$

$\hat{\bar{s}}_i$ and $\hat{\bar{s}}_{i\Sigma}$ are partial sets from the total set of the parameters $\left(\hat{n}_n^{(h)}\right)_i$ and $\left(\hat{n}_{n_\Sigma}^{(h)}\right)_i$ from (3.8).

As a result of the present analysis, we expressed the profit for the certain year S as a function of the numbers of specialists on computers where $t+1\leqslant S\leqslant t+1$. Taking into consideration that the year S appears to be a future one with respect to the year t, the a priori information which is necessary to define the function of the profit can be obtained on the basis of short and long period forecasting.

Each of the numbers $\hat{n}_{i,s}$ of specialists available for the S-th year, (for the sake of simplicity we will skip the subscripts) can be expressed on the basis of specialists $q_i=q_i(\bar{v}_{t-1},\gamma,s)$ who graduated until the S-th year, as a result of the previous controls $\bar{V}_{t-1}=(\bar{V}_{t-1},\bar{V}_{t-2},\ldots)$ and the specialists n_i, as a result of V_t, which has to be optimized:

$$\hat{n}_{i,s} = q_i.\delta+n_i(n_{F,i},n_{I,i},n_{S,i},\bar{\gamma},\bar{\tau},\bar{\zeta},s) \quad (3.10)$$

In this case, $n_i = n_i(n_{Fi},n_{Ii},n_{si},\bar{\gamma},\bar{\tau},\bar{\zeta},s)$ expresses the relation between n_i and the number of specialists educated by the different forms of education, taking into consideration the random factors. The variable $\bar{\gamma}$ is a vector, which defines the connection between $\bar{y}_{t+\tau}$ and $\bar{v}_t$. Its components represent the percentage of failed students and of those not working with the subject they have studied, $\bar{\tau}$ is the vector of the „pure delay"; $\bar{\zeta}_i$ is a vector whose components are weight-coefficients reflecting the preferences to the different forms of education of specialists for CC's; $\delta = \bar{\delta}$ is a coefficient for staff changing in the CC's.

Because of the relatively large period of the control-one year, it is not possible to define a realistic density of the probability of the random variables $\bar{\delta}$ and $\bar{\gamma}$; that is why in (3.10) their forecasted values were introduced with the help of corresponding extrapolating operators.

Finally, for the function of the profit $W(\bar{V}_t)$, the following expression was obtained:

$$W(\bar{V}_t) = \sum_{i=1}^{N_c} \sum_{s=t+1}^{t+\tau_{max}+1} \left[m_i(\hat{n}_{ci}(s),\bar{u}_i).\int_{\Omega_i} w_i.k_i.\xi_i \times \right.$$

$$\left. \times f_{i\,con}(t_{ai},t_{mi},w_i \middle| \alpha_i(s),\hat{\bar{s}}_i(s),\hat{\bar{s}}_{i\Sigma}(s))d\Omega_i \right] \quad (3.11)$$

The number of needed specialists to be educated, according to the structural scheme for computer education (see Fig. 2.1) and which appear to be an element of the control $\bar{V}_t$, can be defined in the following way:

$$n_c = \sum_{i=1}^{N_c} n_{ci} \quad (3.12)$$

for the specialists, working in the CC's and

$$n_n = b \sum_{i=1}^{N_c} n_{ni} \quad (3.13)$$

for the specialists not directly connected with the CC's, where b is the coefficient of simultanety.

3.2 The function of the expenses

This function is defined in the observed interval of time $t+1 \leqslant S \leqslant t+\tau_{max}+1$ using the expression of the form:

$$L(\bar{V}_t) = \tau_{max} \cdot \left[l_1(\bar{V}_{t-1}) + L_1(\bar{V}_t) k_{eff} \right] + \sum_{i=1}^{N_c} \sum_{s=t+1}^{t+\tau_{max}+1} L_2[\hat{n}_{ci}(s)]$$

where $l_1(\bar{V}_{t-1})$ are the yearly returns on the expenses for education connected with the control $\bar{V}_{t-1}$, and $L_1(\bar{V}_t)$ are the expenses on the basis of the strategy chosen, $\bar{V}_t$, k_{eff} a standard coefficient for the efficiency of expenses, $L_2[\hat{n}_{ci}(s)]$ the yearly expenses for the staff (wages, etc.), $\hat{n}_{ci}(s)$ of the i-th CC, τ_{max} are considered as an integer(years).

With the help of the defined function of the profit $W(\bar{V}_t)$ and also that for expenses $L(\bar{V}_t)$, can be defined, according to (2.1) the efficiency function we are seeking, $H(\bar{V}_t)$, whose maximum corresponds with the optimal strategy $\bar{V}_t$ for computer education for a given moment t.

Fig. 3.4 represents the function mentioned above with regard to the number $n_c^{(h)}$.

The definition of the maximum of $H(\bar{V}_t)$ can be carried out on the basis of the well-known methods for optimization, FELDBAUM 1968, HASEN 1968.

4. CONCLUSION

1. As is clear from this paper, the problem for computer education can be described mathematically, which gives us a possibility of defining its optimal strategy.
2. Using a stochastical, self-adjusting model, we assure the flexibility necessary for the system of computer education with the aim of obtaining maximum efficiency.
3. The method we offer here for the optimal control of computer education could be applied successfully for a similar system for computer education again, but would be much more complicated.
4. The investigations we have shown here, do not exhaust completely the complicated problem for a selection of optimal strategy for computer education, but they can be used as a fundamental basis for further investigations into this question.

REFERENCES

1. FELDBAUM A., 1968, The fundaments of the optimal automatical systems theory, („Nauka"), Russian.
2. HASEN E.M., 1968, Methods for optimal statistical decisions and optimal-control problems, („Sovetskoe radio"), Russian.
3. PUGATCHOV V.S., 1960, The theory of the random functions (Fizmatgiz), Russian.
4. The papers of the East European Symposium on Computer Education, (Hungary, Sept., 1969).

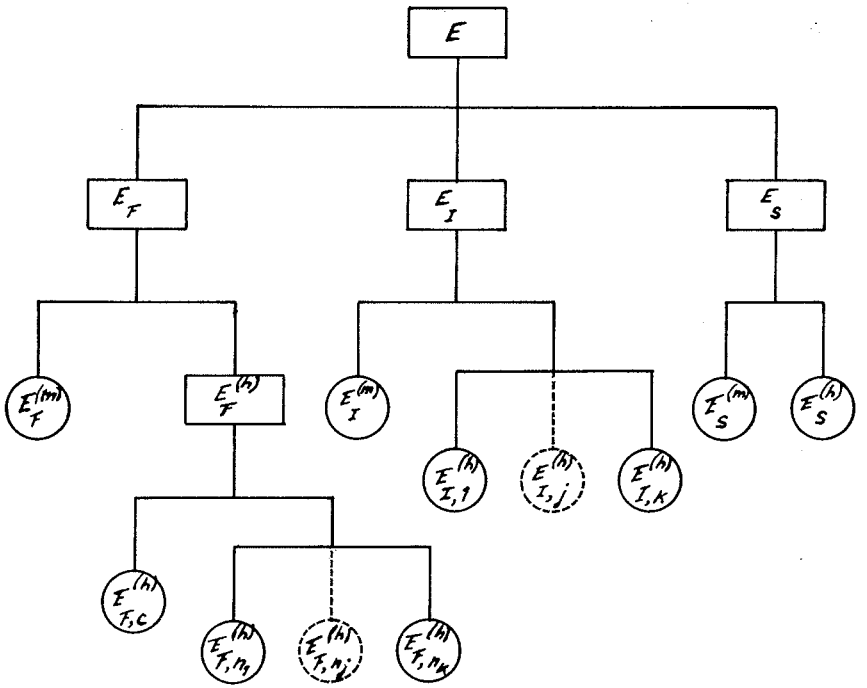

E
E_T
E_I
E_S
E_T^(m)
E_T^(h)
E_I^(m)
E_S^(m)
E_S^(h)
E_I,1^(h)
E_I,j^(h)
E_I,K^(h)
E_T,c^(h)
E_T,n_1^(h)
E_T,n_j^(h)
E_T,n_K^(h)

Fig. 2.1.

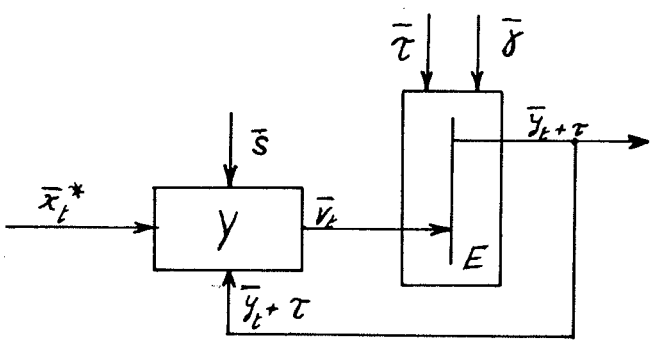

τ̄
γ̄
s̄
ȳ_t + τ
x̄_t*
y
v̄_t
E
ȳ_t + τ

Fig. 2.2

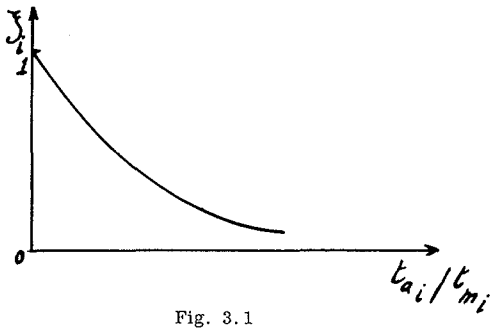


Fig. 3.1

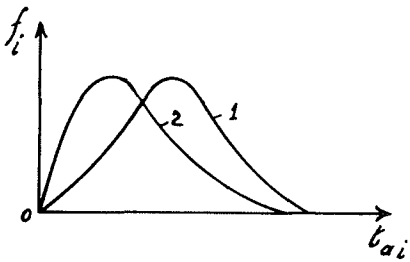


Fig. 3.3

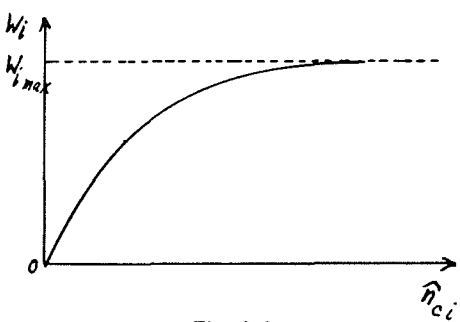


Fig. 3.2

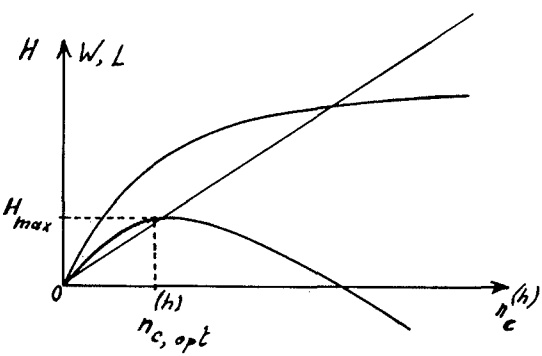


Fig. 3.4

L. E. A. R. N.:
LOCAL EDUCATIONAL AUTOMATED REPORTING NETWORK

Howard L. Sampson

1. HISTORICAL BACKGROUND OF LEARN

The impetus for ADP (Automated Data Processing) in the Madison Public Schools came out of the tedious task of hand tabulating the annual school census. The need for the automation of this process was apparent and in the fall of 1962, conversion of the school census records into punched cards was started. From the census project, automation moved into school enrollment procedures because of the direct relationship to the school census.

A short time later, a data processing department was created within the Business Services Division. At its beginning, the department consisted of three staff members and basic unit record equipment.

The 1963-64 school year saw additional utilization of the data processing equipment. A payroll system was designed in the fall of 1963. This system went into production on January 1, 1964.

A budgetary accounting system was designed and implemented during the 1964-65 school year. This system automated all accounting procedures effective January 1, 1965.

Concurrently, additional projects utilizing ADP equipment were proposed. One of the projects, class scheduling, was planned and implemented in the junior high schools during the 1965-66 school year. Grade reporting for senior high schools was also implemented that year.

At this time, new requirements for services were not possible on the unit record equipment, thus, an IBM 1401 computer configuration was placed on order.

In addition, due to the increasing complexity and demands of data processing, a separate division for data processing services was created in the organizational structure of the Madison Public Schools. The new division called Management Information Services, became a functional unit with the installation of the IBM 1401 computer configuration in August, 1966.

In 1966-67, the first year as a division, all existing unit record operations were converted to the new computer system. Grade reporting and scheduling were extended to all junior and senior high schools.

During the next two school years, 1967-68 and 1968-69, the use of data processing in the school system expanded rapidly and became an integral part of the administration of Madison Public Schools.

To coordinate the future growth of data processing, a new concept, LEARN (Local Educational Automated Reporting Network) will be used. It breaks the present and proposed reporting systems into nine major areas. The nine major systems presently contemplated are Financial, Library, Payroll and Personnel, Physical Facilities, Pupil Accounting, Purchasing, Instructional Applications, Educational Research and Administrative.

These reporting systems will be designed to allow for general expansion or growth as well as the introduction of new data which may be determined at a later date, to be significant. It is entirely possible that the proposed reporting systems might be merged or new ones created as experience provides a clearer picture of the total requirements of data processing in Madison Public Schools.

2. DATA PROCESSING CENTER

The data processing center is located on the basement level of the central administration building and is divided into the computer room and office space. An IBM 1401 computer configuration, unit record equipment and keypunch equipment are located in the computer room. Adjacent to the computer room is office space that is shared with Financial Services. In addition, two MIS (Management Information Services) staff members occupy office space elsewhere in the central administration building. The situation has been further compounded with the creation of new positions in MIS.

To rectify the lack of office space and to provide for a new computer configuration, remodeling of the entire data processing center is necessary. The plans have been completed and the actual remodeling will begin early in 1970. At that time, Financial Services will move to a new complex of offices on the second floor of the central administration building.

The existing office space will be divided into a keypunch area, storage area, and offices. The area for the keypunch machines will be sound-proofed because of the high noise level of the equipment. The storage area will be designated for storage of data prosessing supplies. All remaining space will be provided for offices.

The present computer room must be revamped to accomodate a new IBM System/360 computer configuration to be installed in June, 1970. Due to the critical temperature and humidity control requirements of the new computer configuration, a separate air conditioning facility will be installed. Remodeling of the data processing center is scheduled for completion by no later than May 1970.

3. PROPOSED AND IMPLEMENTED SYSTEMS

SECTION A
FINANCIAL

The current financial system encompasses budget preparation and budgetary accounting and reporting. Salaries make up a large part of the total budget and arriving at a realistic salary estimate is an important segment of budget preparation. Using the salary placement information contained in the payroll-personnel master file, salaries are projected for each budget year. The salaries are

first projected by position within the fourteen functional salary areas of the budget. Then the projected salaries and positions are summarized by track and level. Using the two projected salary lists, an accurate salary estimate for the budget year is obtainable. This process has an additional use during salary negotiations. Each salary proposal can be projected to arrive at a cost factor and a realistic evaluation can be made of it.

The remaining portion of the automated budget preparation process is used to maintain a current budget during the modification and review stage. Once the final budget is approved, the budget preparation file is used to prepare the required chart of accounts. Finally, accounts are loaded with annual budget appropriations.

Next, budgetary accounting and status reporting is initiated on a regular basis. The file is maintained in punch cards with each financial transaction updating the appropriate accounts.

Monthly reports are prepared to show budget activity. A monthly detail ledger shows all income and expenditure activity for the month. The accounting status (detail) and the budget status (summary) reports are prepared to show what occurred at each location of budget responsibility. These are used by management to keep abreast of their budget status. An operating statement for the entire system is prepared to show the budget activity in summary.

Other reports include the encumbrance listing, cash receipts, interim bills and accounts payable (board bills). An analysis of operational and maintenance accounts is prepared for Building Services to evaluate their accounts. The detailed ledger is also prepared both on a calendar year basis and a fiscal year basis. These are summarized with an annual operating statement.

In the near future, a new budgetary accounting system will be designed for the IBM S/360 computer configuration scheduled to be installed in June, 1970. Since the program budgeting concept is under serious consideration at the present time, the new financial system will include provisions for it.

The most significant departures from the present system will be the elimination of punched cards from the accounting process and the utilization of telecommunication terminals for remote update and inquiry. The master accounting file will be maintained on a computer accessible file. All transactions affecting the file will either be entered via terminal or in the case of automatic purchase order writing, machine generated.

It is comtemplated when a purchase order is written, the computer system will encumber the correct account and create an accounts payable record. Invoices entered on the terminal will cause the de-encumbering and liquidation of the account and put the accounts payable record into the next board bills process. New appropriations, and transfers in and out of accounts will be entered via terminal. Also, the system will be developed to provide for the de-encumbrance of partial payments. The entire function will be controlled by the Director of Business Services or his delegated agent.

With the implementation of the new budgetary accounting system, meaningful budget status reports will be provided. Account transactions could be shown in addition to the normal summary totals now shown and periodic budget status reports would reflect an accurate up to date picture of all accounts.

Over a period of years, a continuing analysis of present and past budgets as well as forecasting future budgets would be possible. It is felt that statistical analysis of budget performance would be more effectively obtained than has been possible in the past.

SECTION B
LIBRARY

Automation of book ordering procedures and the printing of catalog cards was the first data processing application developed relating to the library reporting system.

The information necessary for book ordering and cataloging was converted to a computer accessible file. From this file, a Madison Instructional Materials (MIM) list is produced annually. The MIM list contains all books alphabetically listed by author that can be ordered through the system. Periodically, new titles are added to the file and supplement MIM lists are produced.

At the same time the MIM list is printed, order cards for each book title are produced. The school librarian uses the order cards to specify the number of books and the number of catalog card sets desired. The completed card set is then used to activate purchase orders printed on the computer and then receipt cards are produced. All output is sent to the Cataloging Department for processing. When books on order have been received, the receipt cards are returned to Management Information Services to produce the catalog card sets and encumber the appropriate budget accounts. Lastly, catalog card sets and encumber the appropriate budget accounts. Lastly, catalog cards are returned to the Cataloging Department to be combined with the books and sent to the school librarian.

This process of book ordering will be expanded from selected library books to all instructional materials, library books, periodicals, reference books and textbooks. The first phase toward this goal will be the conversion of all cataloging information obtained through the Library of Congress. Once this plan is fully implemented, a complete inventory of all books will be established.

A circulation control system will be developed after all books are on the main file. Overdue notices and follow-up overdue notices will be prepared and sent to the home of the student when necessary. In addition to the control function, this system will generate valuable reader preference information. It will show what specific books and what general category of books are in demand and with this information, selective book ordering will be possible.

With a central computer accessible file, a central card catalog system using telecommunications will be developed. A student will have the ability to determine whether a book is in their school library of if it is available in any other school library by means of a remote terminal device. Book sharing between schools will be possible and it could make feasible the purchase of books either too costly or in not high enough demand for one school.

The ordering process of new books will be simplified, because all titles would be on the central file with a unique identification number. The school would code the identification number of the book and the total number of books desired to an optical scanning sheet. The sheets would be processed causing the purchase orders to be written and the correct accounts encumbered. Upon receipt of the books, they would become part of the circulation control system and available for utilization.

SECTION C
PAYROLL-PERSONNEL

A payroll-personnel master file has been developed for ease in maintaining payroll and personnel information and because the Personnel Division and the Financial Services Department have been duplicating their efforts in many areas of respective record keeping. This effort began in August, 1967 with the preparation of he Teacher Education and Experience Record. In addition, applications implemented in the personnel area include the State Teachers Report, the Staff Directory, salary projection and contract printing.

The creation of the payroll-personnel master file also made it possible to process all payrolls from one master file. In addition to checks, other payroll reports include check registers, deduction registers, payroll preprints, and payroll distribution. Other reporting requirements include the Wisconsin Teacher Retirement (monthly), Social Security (quarterly), Wisconsin Retirement Fund (quarterly) and annual employee earnings (W-2).

The combined information for both payroll and personnel on one file provides the ability to collect, sort, process, store, retrieve, analyze and report information on all employees of the Madison Public Schools. The system simply involves the process of pulling together many disparate activities into a logical, meaningful, integrated whole to accomplish a given objective. Here the computer acts on masses of information in a much more efficient and orderly manner than previous methods allowed.

In the next several years, the major change in the payroll-personnel system will be the conversion to a new computer configuration. The new configuration will make possible the direct access to the payroll-personnel master file using telecommunications terminals. This development would enable some additional applications in the system. A potential application would be the creation of an applicant file. If a principal wishes to look at the applicants for a specific teaching position such as English, a code could be entered through the terminal and retrieve a video display of all English applicants. When an applicant is put under contract, the applicant's record would be transferred to the payroll-personnel master file.

In conjunction with the applicant file and the possibility of program budgeting, a position control file would be created. The position control file will contain all the certificated personnel allocations in the school system. At any time, the number and nature of vacant positions would be available for recruiting purposes. Statistical analysis of the position control file would give valuable budget information by disciplines, turnover rates, average years of experience and other information as required.

These future applications only represent a sampling of the true potential of this reporting system and the benefits which will be derived.

SECTION D
PHYSICAL FACILITIES

Although the computer is not utilized in the present physical facilities system, many excellent potential subsystems do exist. One of these sub-systems could be an equipment inventory control system. Items to be included in this system would be furnishings (chairs, desks, etc.) and moveable equipment (typewriters, movie projectors, etc.) maintained through the use of the computer.

An identification number would be a unique control number assigned to each item and permanently attached to each item in the Madison Public Schools. Transactions effecting the file, (additions, deletions, etc.), would be updated as required. Various reports for analysis of equipment utilization, abnormal rates of repair of obsolescence, and surplus items could be produced either by location or for the entire sytem.

Annually, a complete equipment inventory for each physical site would be produced. The school administrator would mark each item as accounted for, missing, in need of replacement or surplus. After the file is updated, reports of missing items, work orders for the repair of items, bid specifications, and purchase orders for the replacement of items could be produced.

The equipment inventory control sub-system will provide control of equipment not now possible. It is believed that accurate cost figures and analysis will be available, and maximum utilization of existing equipment will be reached.

Another sub-system could be a physical facilities inventory system. This sub-system would maintain a complete inventory of all buildings in the Madison Public School System with a situs file containing pertinent information on each building. Typical information contained in the file, by building, would be site size and cost, site improvements and cost, size of the building by floor, building cost by functional category, size and function of rooms, number of students and teachers by classroom and other related data.

With the planning and construction of schools spanning several years, CPM (Critical Path Methods) and PERT (Program Evaluation and Review Technique) could be used to good advantage for developing timetables through use of the computer. The results would provide each person, responsible for certain tasks, ample notification of when completion should be accomplished.

SECTION E
PUPIL ACCOUNTING

The pupil accounting system maintains student information for census, enrollment, the student cumulative record, scheduling, grade reporting and attendance.

Census is the starting point for the accumulation of information on a student. Each spring, census enumerators take a census of the Madison Public School District. The census identifies all children from birth through the age of twenty by name, address, age, sex, and school of attendance. The major uses of this census data are to project future school needs and to establish school boundaries. Once this information is captured and put into machine readable from, it is used to produce several reports. Two primary reports are the Census Report and the Child Census Distribution. The Census Report lists all the census information on a child and the Child Census Distribution summarizes the information by school district.

Another service provided to Pupil Services is the maintaining of the student enrollment file. When a child enters school for the first time, an enrollment card is prepared and it becomes a component part of the enrollment file. Each fall, these cards are prepared and sent to each elementary school. A master enrollment file for the entire school district is maintained in Pupil Services.

A pilot study to automate the cumulative record

keeping function at the secondary level was made at LaFollette High School during the 1968-69 school year. This is the first phase in the development of an on-line pupil information system. Each student record was maintained on a computer file and was automatically updated from scheduling, grade reporting, attendance, census, and standard test scores. A current cumulative record for use at the school was printed after each grade period. Other features of this system provide the ability to print student transcripts, calculate class standing and grade point average, and in general terms, maintain each student record on a current basis.

Class scheduling is simply meeting demand (student requests) with supply (teachers and classrooms). The purpose of computer scheduling is to help school administrators assign students to class sections rapidly and efficiently as possible. Each student submits an election sheet for the classes they desire to attend. Election sheets are then processed by the computer and the result is a total request summary for each class. Next, the school administrator prepares a master schedule (teachers, class times and rooms) using the summary as a guide. The computer then matches the requests against the master schedule assigning students to a class schedule.

Upon completion of the scheduling process, class lists, homeroom lists and student schedules are produced. Currently, secondary schools are utilizing the computer in their class scheduling process. Scheduling information is then used to prepare grade and attendance cards. Quarterly, the attendance cards are sent to MIS (Management Information Services) with the quarterly grade sheets for processing. The attendance summary cards and grade sheets are used to produce student report cards and teacher graded class lists. Cumulative labels containing student grades and attendance records are produced as required by the computer and sent to the appropriate school to be affixed to the student cumulative record.

In addition, MIS (Management Information Services) is providing scheduling and grade reporting services to four Madison suburban high schools with another school contemplated for the 1970-71 school year.

A pilot attendance system was performed at East High School during the 1968-69 school year. Daily, each teacher recorded attendance on the student's card for that class. At the end of the week, all attendance cards were processed by the computer with a total summary of all absences for the school and per student calculated. If a student had two or more class absences per week, a notice indicating the absences was produced and mailed to the parent. A duplicate copy of the notice was provided to the student's guidance counselor. At this time, the feasibility of expanding this attendance system throughout Madison Public Schools is being evaluated as to its merits.

A new innovation planned for the pupil accounting system will be the combination of all the student files into a single computer master file. The single master file plus the use of telecommunications terminals will greatly facilitate the tedious and time consuming record keeping function. Maintenance will be made to a single file rather than the present six or seven files to keep student records accurate and up to date.

Test scoring of standardized tests and teacher made tests is now possible due to the installation of an Optical Scanner. The initial phase in a program for scoring standardized tests has been implemented. This phase involves scoring of all G.E.D. tests on a weekly basis. Expansion of this program is currently in the systems and design stage. A pilot program for scoring teacher made tests is in development stage. Schools participating in the pilot program will be West Senior High, West Junior High and Leopold Elementary. The curriculum discipline for this program has not been selected at this time.

SECTION F
PURCHASING

Currently, Management Information Services provides a monthly report showing all stock transactions by departments or schools and annually, a complete stock inventory report for the Purchasing Department.

In the future, the entire stock inventory could be maintained on a computer master file. Using advanced programming techniques, optimum in stock quantities, reorder points and reorder quantities would be calculated. This would be a continual process with any change in usage, reorder lead time or the many other variables effecting the calculation of an optimum stock inventory. In addition, a new inventory system for Food Services could be developed on a similar basis. It would keep an accurate perpetual inventory of cafeteria food supplies.

Initially, purchase orders will be produced by the computer to replenish the inventory items with the appropriate accounts being charged. Soon after, purchase order writing will be expanded to include non-inventory items as well as inventory items. All vendor information for purchase order writing will be stored on a vendor master file and be available as needed.

In summary, is should be noted that the purchasing function in the Madison Public Schools is one of a very complex nature. As a result, a great deal of human judgment must be exercised to maintain year-end inventories with surplus limitations. Therefore, only further detailed systems analysis will determine whether or not the above thoughts are feasible for implementation.

SECTION G
INSTRUCTIONAL APPLICATIONS

A pilot study of the instructional use of a computer was undertaken at two Madison senior high schools during the 1968-69 school year. This was Madison Public Schools first exposure to utilization of the computer as an aid to teaching.

The exposure started when a telecommunications terminal was connected to a computer in Chicago in September, 1968. Two months were used setting up the pilot and it was implemented in November, 1968.

The initial pilot covered only the problem solving use of a computer. Two general math classes, one experimental group and one control group, were chosen for the project. The experimental group used the terminal to enhance the learning of properties of the real numbers, geometry, operations with rational numbers and probability and statistics. The control group proceeded as a normal general math class. Comparisons were made between the two groups periodically throughout the year.

An evaluation at the termination of the pilot program in May, 1969, showed the experimental group were higher achievers than the control group in motivation and learned concepts.

Another instructional application area involved the use of the Management Information Services computer center to demonstrate computer concepts. This was accomplished by two methods.

The first method was a demonstration and familiarization of the data processing equipment. The second method involved students developing their own programs for the IBM 1401 computer. These programs were written in class and tested on the computer by the students. The „Hands On" concept has proved beneficial in learning the operations of the computer.

As a result of the 1968-69 pilot study using the computer as an instructional aid, the following plan to pursue the use of CAI (Computer Assisted Instruction) and CMI (Computer Managed Instruction) has been approved:

Develop a design team which should include the Directors of Secondary Education, Elementary Education and Management Information Services, an Educational Research Analyst, and teachers who are presently involved in this effort to date to perform the following fuction.

1. Analyze the current system for purpose and need.
2. Specify educational objectives of computer use in instruction at senior, middle and elementary levels.
3. Define everyday limiting constraints which any proposed system must satisfy.
4. Adopt alternative approaches to implementing such a program.
5. Analysis and selection of best alternative approach.
6. Development and pilot implementation of selected approaches at the senior, middle and elementary levels of education.
7. Evaluation to determine effectiveness of the system on meeting stated objectives.
8. Based on experimental and real results, feedback the required modifications and continue this cycle until objectives have been attained.

It is believed this approach will facilitate communications among members of the design team, speed up design, development and production and increase satisfaction of participants with the ultimate product. In addition, it is believed this approach has a particular applicability because it places much emphasis on problems of implementation, evaluation, feedback and revision – an emphasis which is an integral part of the educational process.

Definitions

CMI – The computer analyzes student performance and directs them to materials such as audio-visual aids or programmed instructional segments stored elsewhere.

CAI – The computer is utilized in four different modes as presented below:

a. **Problem Solving** is a technique where the computer is used as a tool in solving problems in math, science, business education, etc. In this mode, the student has direct communication with the computer via a terminal device.
b. **Simulation and Gaming** is a technique where the student has the ability to carry on a dialog with the computer via a terminal device. This technique provides the ability to extend classroom experiments with the computer's capability to act on large volumes of data. In addition, it provides the ability to sample and simulate selected data for purpose of arriving alternative solutions to problems.
c. **Drill or Practice** is a technique that allows the teacher to author lesson plans to be presented and maintained on a terminal device, by the computer to students. The teacher then follows with discussion of subject matter. The prime benefit to this mode is standardizing presentation of subject matter to students.
d. **Tutorial** is a technique that tests and judges the understanding of students on subject matter. As in the Drill or Practice mode, this mode is teacher developed and is presented on a terminal device by the computer on a question-answer basis to the student. If the student answers a question correctly, the same question reappears to give the student another opportunity to advance into the lesson. After a predetermined number of unsuccessful efforts, the computer will relay remedial action to the student via the terminal device.

SECTION H
EDUCATIONAL RESEARCH

Projects not falling within the systems defined previously may be assigned to the Educational Research Analyst. This area of responsibility is described as a „what's needed" function, in that the analyst works with the user to further define necessary procedures for processing the particular project.

These procedures include such services as:

A. Basic explanation of data processing.
B. Development of card formats.
C. Development of coding.
D. Selection of programs.
E. Writing of programs.
F. Preparation of control cards and submission of data to computer.
G. Explanation of computer output.

The user submits a written description of the project to the Director of Management Information Services. This should include the purpose of the study, the data to be used, the method of data collection, and the output required. Preferably, this request should precede the actual data collection, especially in a survey-type study, since the format of the questionnaire of survey from used has a great effect on the efficiency of the processing.

Specifically lacking from this list is interpretation of results and statistical consulting. Although in some instances, discussion of the various statistical procedures available may occur, determination of the statistics to be used remains the responsibility of the user.

Priorities for completion of projects are determined by their order of receipt and the processing requirements. Any employee of the Board of Education may submit a project request; the Director of Management Information Services reserves the prerogative of acceptance or rejection, and of determining a projected completion data of projects accepted.

Examples of the educational research projects completed during the 1968-69 school year are shown as follows:

A. Analysis of the questionnaire sent to all 1967 graduates of Madison School District high schools. The hypotheses developed by the committee of guidance counselors directing the study were tested by the use of the chi-square statistic.
B. Comparison of different teaching methods used in elementary physical education classes and in reading classes.

C. Calculations for PhD. dissertations in educational administration.

D. Analysis of questionnaires designed to test the effect of special education programs.

SECTION I
ADMINISTRATIVE SYSTEMS

Administrative systems are on-going applications that are too small to be considered as a separate reporting system and do not fit into any of the previously defined reporting systems. However, they are still very important in the total LEARN program.

An example of an administrative system would be the sports registration system under development at the present time. This system will provide for a registration and printing of rosters for all teams participating in various sports sponsored by the Sports Office of the Madison Public Schools.

Presently, the Sport Office oversees nineteen different sports. The registration and printing of the team rosters is a cumbersome manual task. In the softball program alone, there are about 3,000 participants.

The registration card has been revised to a new format from which information will be keypunched into an IBM card. These cards will be sorted by league and team and the rosters will be printed for distribution. Periodically, MIS (Management Information Services) will print updates of the rosters.

Another administrative system is the Independent Study Reporting System at James Madison Memorial High School. Under the I.S. (Independent Study) program, a student was allowed to take course work in addition to his regular class load. A computer system was designed to facilitate the administrative task of record keeping for the I.S. program.

The system operates very simply with each student being evaluated on number of conferences, number of projects, initiative, originality and performance three times a year. The teacher records the evaluation on mark sense cards and sends them to Management Information Services for processing and production of the following reports. The most important is the I.S. Progress Report, in essence a report card on the students independent study, which is sent to the parents. Three other reports list the student evaluations by teacher, by course, and then in summary by teacher and course.

A third example of an administrative system is the printing of student identification cards. Each year, current student identification information from the student file is printed on the I.D. form. The forms are distributed to the appropriate school which in turn affixes the student photo and laminates the card before giving to the student.

SECTION J
TENTATIVE 3 YEAR SYSTEM IMPLEMENTATION SCHEDULE*

(*BASED ON EXISTING STAFF)

4. EDUCATION, ORGANIZATION AND POSITION SPECIFICATIONS

This part describes the education, organization and position specifications of the Management Information Services Division.

An in-service training program is being developed to

TENTATIVE 3 YEAR SYSTEM IMPLEMENTATION SCHEDULE *

System LEARN — INSTALLATION PLANNING SCHEDULE

CODES
S-Systems Design
P-Programming
I -Implementation and Follow-Up

REPORTING SYSTEM	1969 October	November	December	1970 January	February	March	April	May	June	July	August	September	October	November	December	1971 January	February	March	April	May	June	July	August	September	October	November	December	1972 January	February	March	April	May	June	July	August	September
A FINANCIAL BUDGETARY ACCOUNTING MODIFICATION EXISTING SYSTEM NEW ON-LINE SYSTEM	S	P	P	I	S	S	S	S	S	S	S	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	I	I	I						
B LIBRARY CATALOGING & PURCHASING MODIF. TO INCLUDE TEXTBOOKS & SUPPLEMENTS AND IMC	S	S	S	S	S	P	P	P	P	P	P	P	P	P	P	I	I	I																		
C PAYROLL-PERSONNEL NEW ON-LINE SYSTEM			S	S	S	S	S	S	P	P	P	P	P	P	P	P	P	I	I	I																
D PHYSICAL FACILITIES																			S	S	S	S	S	P	P	P	P	P	I	I	I				
E PUPIL ACCOUNTING LAFOLATTE PILOT OTHER SCHOOLS	S	S	S	P	P	P	P	I	I										I	I	I	o														
F PURCHASING (Not in Three Year Plan)																																				
G INSTRUCTIONAL CAI (SPECIFIC CURRICULUM DIS. NOT DETERMINED AT THIS DATE	S	S	S	S	S	S	S	S	P	P	P	P	P	P	P	P	P	I	I	I																
H EDUCATIONAL RESEARCH	S	P	I	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	S	P	I
I ADMINISTRATIVE SYSTEMS	S	P	I	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	S	P	I

* BASED ON EXISTING STAFF

introduce data processing to all staff members of the Madison Public Schools. Periodically, seminars will be held either at the central administration building for central staff or at each school for the school staff.

The course outline is below:

I. Brief history of computer development.
II. Basic computer concepts.
  A. Computer system elements.
  B. The stored program concept.
  C. Computer organization.
  D. Stored media.
III. Survey of computer equipment.
  A. The card reader.
  B. The card punch.
  C. The high speed printer.
  D. The magnetic tape system.
  E. Random access devices.
  F. Central processing unit.
IV. Stages of system development.
  A. Definition of the problem.
  B. Design of the system.
  C. Program development.
  D. New system implementation.
V. General discussion period.

An organization chart (Exhibit I) and position descriptions (Exhibit II) are shown on the following pages.

5. EDP PROCEDURES GUIDE FOR USERS

The EDP Procedures Guide for Users containing procedures to be followed to secure data processing assistance in any school or department is under development and will be issued as a supplement to the LEARN Program Report.

ACKNOWLEDGEMENT

The LEARN Program Report was the product of thought and diligent work of the entire staff of the Management Information Services Division.

EXHIBIT I

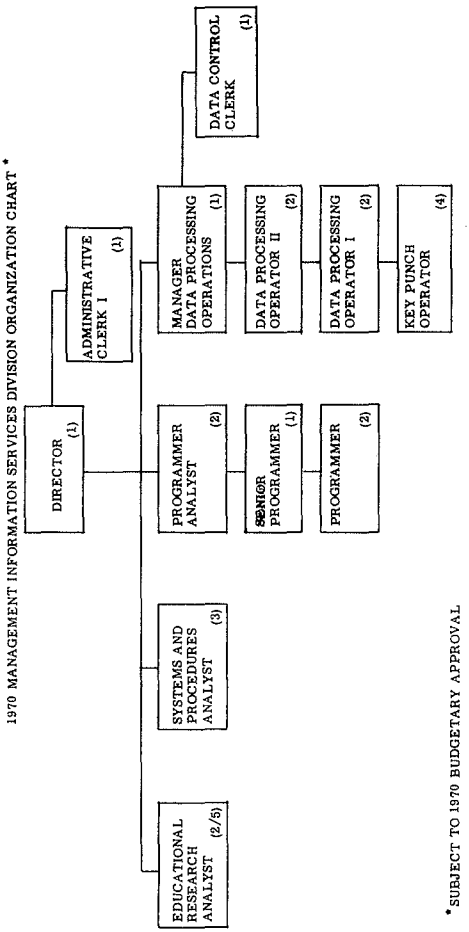


EXHIBIT II
MIS POSITION DESCRIPTIONS

A. DIRECTOR
Administrative and technical director of all data processing activities for Madison Public Schools.

B. SYSTEMS ANALYST
Deals with management throughout the school system to develop information requirements, describe procedural and operating improvements and define data processing support.

C. EDUCATIONAL RESEARCH ANALYST
Deals with management throughout the school system to develop research requirements, describe procedural and operating improvements, and define data processing support.

D. PROGRAMMER ANALYST
Assists in analysis of systems problems, development of solutions and programming of computer applications.

E. SENIOR PROGRAMMER
Plan, develop, test, document and modify computer programs.

F. PROGRAMMER
Assists in planning, developing, testing, documenting and modifying computer programs.

G. MANAGER, DATA PROCESSING OPERATIONS
Schedules and supervises operation of data processing equipment.

H. DATA PROCESSING OPERATOR II
Lead operator on data processing equipment.

I. DATA PROCESSING OPERATOR I
Assists in operation of data processing equipment.

J. KEYPUNCH OPERATOR
Keypunches data into computer input compatability..

K. DATA CONTROL CLERK
Controls input to and output from the data processing center.

L. ADMINISTRATIVE CLERK I
Provides the secretarial and clerical services for the Management Information Services Division.

This paper is published with the permission of Her Majesty's Stationery Office, Great Britain.

THE DEVELOPMENT OF A COMPUTER MANAGED TRAINING SYSTEM IN THE ROYAL NAVY

J.W. Scorer & J. Foster

1. INTRODUCTION

1.1. This paper describes a project which will extend work on programmed learning in HMS COLLINGWOOD. HMS COLLINGWOOD is the Weapon Electrical School of the Royal Navy and trains all members of the Navy's Weapon Electrical Branch from newly entered Mechanics to Senior Technicians and Officers. Programmed instruction has been used here in some form or other since 1963. The early experiments have led to accepted practice, but experimentation continues as more programmed instruction is developed. The Mechanic training scheme and its progress towards 'Individually Paced Training', 1963 to December 1968, is described in the Pergamon Training Systems in Industry series Volume 1 Number 3 – Integrated Programmed Instruction in HMS COLLINGWOOD. A major disadvantage remaining in the present system is that learning is still not individually paced throughout and there are no tangible goals for the trainees, by diligent work, to become trained men more quickly. Individual pacing occurs in parts
However, the course as a whole remains tied to the class system and there is little incentive to the man to try harder since he reverts to his class for the unprogrammed parts. These classes start every two weeks and it is possible for men to jump classes but few actually do it. An objective of the individually paced wholly programmed course is then to permit men to go through the system more quickly by offering them seniority and hence cash for expeditious work on course. We attach great importance to this. First, because we think it will make men more objective minded and it will give them a better sense of purpose. Second the Services have the prospect of shorter engagement time being imposed on them which will bring into very much sharper focus the need for task specific training and objective minded men – (Huggett 1969).

1.2. It was these disadvantages which identified the need for Individually Paced Training. The work done by the US Naval Training Research Laboratory and reported by Johnson (1968) indicated that the potential advantages to trainees of programmed methods could be more fully realised by attempting to hold student quality constant while allowing the training time to vary. Training time can be reduced by any or all of the following factors in an individualised training system:

(1) Alternative paths through the instruction.
(2) Elimination of unnecessary instruction.
(3) Avoiding repetition of what some students already know.
(4) Taking advantage of tutorial and self instruction material.

1.3. At the beginning of 1968 we decided to design and implement an individualised version of the Radio Electrical Mechanics Course. The course is 19 weeks long with instruction in 4 categories:

(1) Basic hand skills.
(2) Electronic fundamentals.
(3) Communication principles and equipment.
(4) Radar principles and equipment.

The target population averages 11 years education, but can vary in age between 16 and 26 years, and averages 18 years. Tests revealed great variation in student ability and the need for individualised training.

1.4. From the very beginning of the project we studied the possible use of a computer to help individualise training by processing tests to provide diagnostic and prescriptive reporting on the course material and individual man; whilst still ensuring the optimum use of resources by:

(1) organising the programmed instruction modules making up the course;
(2) Distributing trainees and tutors to maximum advantage, (there are 60-130 men in the system at any one time);
(3) organising the use of course facilities (some 300 pieces of equipment are included for practical work).

And the computer was used further to aid the production of programmed instruction modules (using text editing, video graphics and hard copy output) to reduce the course revision time cycle. We were encouraged by Brudner's (1968) paper on Computer Managed Instruction published in Science, Volume 162, but took careful note of the Schutz (1969) letter in Science, Volume 163, which stated that some of the problem areas identified by Brudner were far from trivial and that CMI technology required a range of capabilities that necessitates the setting-up of a multi-discipline team. For this project the team had to be capable of:

(1) Producing the course behavioural objectives;
(2) Constructing criterion tests;
(3) Preparing the programmed learning materials;
(4) Devising manageable logistical procedures for administering the course;
(5) Producing computer programs to mark the tests, maintain files on trainee performance, analyses test results and file contents and prepare diagnostic and prescriptive reports on the learning materials and trainees;
(6) Developing training for the instructor to ensure competent use of the system.

The multi-discipline team set-up included:
(1) programmed learning material writers, plus the service of a Learning Systems Limited consultant;
(2) subject area specialists, in this case engineering training staff;
(3) computer programmers and a system analyst trained in the Naval digital weapon field;
(4) and, when required, the assistance of behavioural scientists from the Royal Naval Applied Psychology Unit.

1.5. To date the course objectives and criterion tests have been produced, the programmed learning materials have been prepared and are currently being validated. The remainder of this paper is concerned with the devising of manageable logistical procedures for administering the course and the production of the necessary computer programs.

2. THE ROLE OF THE COMPUTER

2.1. In defining the role the computer was to play our intention was to use the computer to manage the learning process and individualise the Mechanics training. Although extensive claims have been made that a computer, using the students past history, can adjust the work presented on a lesson to lesson basis we accepted the views of D'Aracy et al (1969) that there was no evidence that we knew enough about individual differences to effect any advantage, or determine what the benefit would be. But a degree of individualisation was possible:

(1) Allowing the time for the achievement of an objective to vary.
(2) Providing learning materials of different degrees of difficulty.
(3) Employing different modes of presentation.
whilst still ensuring the optimum use of resources, particularly capital equipment and tutor time, by scheduling each trainees learning.

2.2. To achieve this degree of individualisation in the Mechanics learning process we have designed the course to consist of a number of learning material modules all of which must be taken by each trainee, although not necessarily in the same order. The trainee must successfully complete a criterion test before he can proceed to the next module. He can take as much time as he requires, within reasonable limits, to complete each module. The following design aims for the computer management of the instruction were specified:

(1) To provide the optimum utilisation of resources; there are 300 separate pieces of equipment of 20 different types, 7 laboratories with 100 work spaces and 12 tutors;
(2) to ensure that the trainees are subject to minimum waiting times as they move from module to module;
(3) to relieve the tutors, as far as is possible, of the routine tasks of marking;
(4) to maintain dynamic files on trainee performance that can be interrogated by the tutors to provide information on training progress;
(5) to evaluate continuously the learning materials by the analysis of the trainee success/failure scores in the criterion tests, the trainee answers to the questions in any particular test, and the trainee responses as they study the programmed learning materials.

It is this last evaluation which will help to eliminate unnecessary instruction, avoid repetition of what some trainees already know, classify learning materials into different degrees of difficulty and assess the efficiency of different modes of presentation. By doing so we aim to further individualise the Mechanics training having first established the trainee quality which can be obtained from a standard set of programmed learning modules with alternative paths through the instruction.

3. PROJECT DESCRIPTION

3.1 *Training Modules*

Each trainee on entering the Computer Managed Training System will be given a training module on which to start work. It will consist of programmed learning materials largely in the form of printed booklets, but audio programmed learning materials will also be used. In addition he will require certain equipments and other materials as the theoretical and practical work are closely integrated. He can take as long as he likes, within reasonable limits, to complete the module.

3.2 *Criterion tests*

Before he may proceed to the next module the trainee must succesfully complete a criterion test. This can take many forms, varying from a test consisting of multiple-choice questions to a test that requires the carrying out of a routine to set-up a complex piece of electronic equipment.

3.3 *Test Marking*

Computer marking imposes limitations on the mode of the questions that can be set unless considerable computer programming effort and expertise is available. Objective questions – multiple choice, multiple completion, multiple response, true/false, matching and classification, lend themselves to computer marking as shown by the work of Maddren et al (1969) on the computer marking of multiple-choice questions. Questions requiring an alpha-numeric answer selected from a number of alternatives can be computer marked, even to the extent of allowing for spelling mistakes as indicated by Gilligan (1969). Freely-structured answers can be handled by syntax analysers but the greater the freedom the trainee has in forming his answer the more difficult become the computer programming problems.

We have the additional problem of the delay the trainee can experience in obtaining knowledge of his test results from computer marking, because insufficient on-line computer terminals can be provided in such a diverse, self-pacing Training System. Reducing the marking load on the tutors would be to the mutual benefit of both tutors and trainees. The tutor would have a less routine and more creative job as he endeavours to assist and guide the trainees with their difficulties and to improve the quality of the learning materials. The trainee would enjoy more individual attention and an improved learning experience. Despite the benefits resulting from computer marking our problems are such that some marking in the Training System will still be done by the tutors. This applies particularly to those practical tests such as the setting-up of a piece of electronic equipment.

3.4 *Tutorials*

Those trainees who do not reach the required standard in a criterion test will be given a tutorial. This means that those trainees who are having most difficulty will have the largest contact with the tutors. Where the criterion test has been marked by the computer the tutor will have a Tutor's Report to assist him in conducting the tutorial. It will contain such details as his mark, the questions he got wrong, the answers he gave to those questions, etc. As a result of the tutorial the trainee may be allowed to proceed to the next module or be required to repeat the module.

3.5 *Prescriptive reports*

This report, which is produced by the computer, prescribes the next module that the trainee is to start work upon It is designed to:

provide optimum utilization of resources;
whilst ensuring that the trainees have minimum waiting time between modules.

The resources available include:

(1) 300 separate equipments, consisting of 20 different equipments ranging from complex communications and radar equipment to relatively simple test equipment.
(2) 7 laboratories with over 100 work spaces.
(3) 12 instructors.

The number of trainees in the Training System at any one time will be about 70. They will be, at 70 different points, proceeding at 70 different learning rates along many different paths through the modules. By using the computer to process this data on resources and trainees we aim to obtain:

greater utilization of equipment, facilities and tutors required;
minimum waiting times for the trainees between modules.

3.6 *Trainee history file*

The Trainee History File is a dynamic file containing information on trainee performance. It will obtain:

(1) Trainee success/failure in the criterion tests.
(2) Trainee answers to the questions in the tests.
(3) Trainee responses to the programmed learning materials.
(4) Times to complete each module by the trainees.

The interrogation of this file will provide:

Trainee Progress Reports;
Learning Materials Reports.

The trainee Progress Reports will indicate those trainees who are doing well, those that are doing badly, those who are deviating from expected performance, etc. They will provide the tutors and others with reports on a trainee's progress that will be useful in conducting tutorials, detecting a run of poor results at an early stage, act as a final report on completion of course, etc. The Learning Materials Reports based on analysis of data held in the Trainee History File will indicate – tests that are proving too difficult, poorly worded test questions, learning materials that are too difficult or too easy, etc. By analysing student responses using the type of statistical methods used by Greenway (1965) and Nicoud (1969) an attempt will be made to improve the programmed learning materials.

3.7 *Computer Access*

The computer management of this project is done using an IBM 1130, accessed once an hour to provide:

(1) Marking of trainee tests.
(2) Production of Tutor's Reports.
(3) Production of Prescriptive Reports.

and used each evening to provide:

(4) Updating of the Trainee History File.
(5) Production of the Trainee Progress Reports.
(6) Production of the Learning Materials Reports.

We restricted the computer access to once an hour during the training day because the computer is used for other training activities. This rate of access is sufficient to manage the Training System without delaying any one trainee for an unacceptable time. Ideally this project should be on-line but within our training establishment this would not have been the most cost effective use of our computer and not justified to obtain the management results we require.

4. SUMMARY

4.1 Our work with programmed learning and our progress towards Individually Paced Training, 1963 to 1968, has lead us into Computer Managed Instruction. It was the realisation that the potential benefits to be obtained from programmed learning methods were eluding us because we did not have a complete individually paced course that first started us along the path to an individualised learning system. In attempting to individualise the learning system for Mechanics we became aware of a resource utilisation and management problem eminently suitable for computer application. Using a computer to manage the course inevitably raised the question of managing the learning process and this we are about to attempt. At this stage we do not know how individual the learning process can be and must therefore use an empirical approach to establish those factors which effect the learning process within our own Naval training environment.

Initially we intend only to attempt a system which permits alternative paths through the instruction so that the training quality is held constant while the individual training time varies. Eventually we hope to establish other parameters in our learning process which can be varied to permit further individualisation of the training. We would emphasise the fact that without a computer to manage the utilisation of resources and the evaluation of the results we would have found it extremely difficult to have attempted even this initial stage in our advance towards an individualised training system. Within our particular training environment we see the computer playing a major role in the future management of training, including the actual learning process.

REFERENCES

1. Brudner, H.J., 1968, Science, vol. 162 pp. 970 - 976.
2. D'Aracy, J., P.M. Ash and P. McLaren, 1969, The Computer as an Educational Tool. In: Study Team of the National Council of Educational Technology, Computer Based Training Systems: A programme for research and development (National Council of Educational Technology, London) p. 15.
3. Gilligan, J., 1969, CAI Crawley, International Computers Limited, Research Department, Loudwater House, Loudwater, Nr. High Wycombe, Bucks., England.
4. Greenway, A., 1965, A Systematic Proccdure for Internal Evaluation of Branching Instructional Programmes, SP(N) Report 2/65, Division of The Senior Psychologist (Naval), Manpower Department, Ministry of Defence, U.K.
5. Huggett, A., 1969, The Development of a Free-Running Training System in HMS COLLINGWOOD. In: Mann, A.P., and C.K. Brunstrom eds., Aspects of Educational Technology III (Pitman, London), pp. 158 - 164.
6. Johnson, K.A., 1968, The Advantages of a More Individualised Training System (U.S. Navy Training Research Laboratory).
7. Maddren, W., and D.R. Williams, 1969, Education in Chemistry, vol. 6 No. 3, pp. 81 - 83 and p. 87.
8. Nicoud, J.D., 1969, Computer-Aided Validation of Skip-branching Programmes. In: Mann, A.P., and C.K. Brunstrom, eds., Aspects of Educational Technology III (Pitman, London), pp. 367 - 370.
9. Schutz, R.E., 1969, Science, vol. 163, pp. 1009 - 1010.

Pergamon Training Systems in Industry Volume 1 Number 3 – Integrated Programmed Instruction in HMS COLLINGWOOD is also referred to.

SOME DESIGN OBJECTIVES OF THE LEEDS EDUCATIONAL NETWORK

Derek H. Sleeman and A.J. Cole

1. INTRODUCTION

The notion of multi-access working is now common in a number of complex situations and this paper is concerned primarily with the hardware and software problems of implementing a multi-access system for computer-based learning. In this particular context, a potentially large number of students interact with a computing system through remote terminal devices and the computer controls the material presented to the students.
New terminals devices and new educational approaches are, however, being rapidly developed and it is necessary to design hardware and software systems in a flexible way. New terminals can then be added to the system easily and new teaching strategies accomodated. The concept of modularity in design is therefore stressed.

The paper briefly reviews early experience gained in computer-based learning on a uni-access Elliott 903 machine and then develops in general terms the requirements for multi-access systems. The particular case of computer-based learning is then examined and the basic components of a general operating system and their functions are reviewed. The hardware characteristics of a Modular One computer are then assessed and the implementation of the operating system is discussed. Further, some of the special hardware and software problems arising in interfacing different types of peripheral device are examined. Some assessment is then given of early experience with the system and the desirable additional developments are discussed.

2. EXPERIENCE OF C.A.L. ON A UNI-ACCESS SYSTEM

This section briefly summarises some work carried out over a period of eighteen months on an 8K (later 16K) Elliott 903 computer with a single teletype console. The educational areas examined included junior arithmetic, English as a second language, elementary science and clinical diagnosis. At the same time a flexible author language was developed (CBLP INTERNAL REPORT (1970)) which allows authors to teach any suitably structured subject. (Fox et al. (1970) and de Dombal et al. (1969) have reported use of this system in the teaching of English as a second language and clinical diagnosis respectively.) Further, as the aim of a C.A.L. system is to individualize instruction such that the system is able, as far as possible, to adapt to the needs of the learner we have investigated adaptive systems where details of performance are used to control the generation of teaching material from basic elements. Recent experiments carried out (Woods et al. (1970)) using such a system to teach junior school arithmetic have shown very considerable gains in the rate of student learning.

A sub-routine facility has been added to the Author Language which allows the student to gain access to facilities (such as calculation facilities) to help him with his work: similarly the author is afforded the facility for instance to monitor the students' progress or to create material "on-line".
On the Elliott 903 system these facilities are provided as sub-routines of the processor but it is expected that on the Modular One these will be segments called from within the operating system.

At an early stage it was apparent that the teletype imposed serious limitations and that, in many areas, the presentation would be greatly enhanced by some form of audio-visual presentation. Therefore a prototype was developed and attached to the system. This device enabled any of a set of pre-recorded messages to be played and any of a bank of about 50 slides to be presented.
The audio and visual components can be controlled separately from the computer and by using a form of relative addressing, the particular "frame" specified by the teacher can be accessed.

3. DESIGN CRITERIA FOR A MULTI-ACCESS SYSTEM

As a result of early experience, we have concluded that the following are the key features for a multi-access system for computer-based learning.

a) Modular computer hardware such that the basic computing system and the number and type of peripherals may be readily expanded.
b) Modular construction of software so that new teaching approaches, additional facilities and sub-routines to service new types of peripherals can be accommodated without causing a major revision of the systems software.
c) Modular construction of the teaching station. Here it is suggested that the system should not have a number of fixed teaching stations comprising of, for example, a graphical display unit, a teletype keyboard and a taperecorder, but rather that there should be a number of different types of terminals grouped and re-grouped to accommodate the needs of any particular teaching situation.
d) Finally, the efficiency of the system is greatly affected by the extent to which the input/output transfers are autonomous.

In a later section we will discuss some of the details of the computer hardware and of the proposed network of terminals. However, the emphasis of this particular paper is on the design of the software to control a multi-access system and the structure of the various segments required to provide the calculating facilities, to carry out the various

teaching strategies and to service the terminals. (Here we are using the segment to mean part, or the whole, of a compiled machine-code program which uses base control (see § 4.2) for service functions such as input/output.) In fact, we suggest that the requirements set out above are those for a very considerable number of systems including process control, on-line patient monitoring, as well as C.B.L. systems. There are necessarily considerable differences in the nature of the input/output peripherals attached to the various systems and also in the algorithms used to control the systems. However, we suggest that the following general structure is able to accommodate all the above applications:

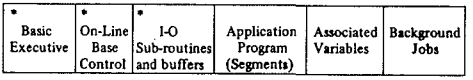


Figure 1 - *Proposed Layout of the Systems Software*
(*Indicates permanently core-resident)

A multi-access system having a very similar layout to the above has already been implemented on the KDF9 (see Wells et al. (1970)).

We shall now analyse the various components of such a systems layout; where it is necessary to be specific we will discuss the C.B.L. application.

4. ELEMENTS OF THE OPERATING SYSTEM

4.1 *Basic Kernell Executive*

The executive should be modularly designed so that it is possible to 'tailor make' an executive to fit the configuration to be serviced and to provide the desired operator-machine and programmer-machine interface. Given a configuration with suitable backing store, then it would be desirable to segment the executive and keep the code for the less frequently accessed routines on backing store. The most basic form of the executive is one which services interrupts.

4.2 *On-line Base Control*

This is the program which effectively controls the whole of the on-line system. The system we have designed in Leeds performs the following functions:

a) Records the number, type and grouping of all the peripherals in the system.
b) Records the status (i.e. inputting, outputting, etc.) of each peripheral.
c) Controls and services all the peripherals.
d) Records the status of each segment which has been called by a job (typical status will be active, suspended, suspended and on disc).
e) Controls the segment overlaying facility.
f) Maintains a queue of jobs to be serviced.

4.3 *Input/output Sub-Routines*

The number and type of these routines will depend on the nature of the peripherals attached to the configuration. It is desirable to have a buffer for each active device resident in core; however, the optimum size of the buffers will depend on the speed of the device, the size of the main store and disc access times. (Although logically separate there is very considerable interaction between base control and the peripheral servicing routines and in practice these sub-routines form part of base control).

Details of the available peripheral configuration are given to the system during the initial set up phase, and so the system knows how much buffer space to allocate and which peripheral service routine calls are permissible.

4.4 *Application Programs (Segments) and Associated Variables*

The system has been designed such that each task processor is written as a segment or series of segments which interface with the base control in a defined way. A large task will, in fact, be a series of segments which will call each other and which will interface with themselves in a pre-arranged manner. Thus we have the concept of a job which has a set of variables associated with it – some of these variables give information about the status of the associated devices, some about the segments which are being used and some which are used by the various segments themselves. Jobs frequently need to be suspended, for example waiting for i/o transfers to complete, and, if there are further entries in the job queue then base control writes the variables associated with the suspended segment to the disc, reads the next segment and its associated variables into core and initiates the job. (See § 4.2).

Thus at any one time only a very small fraction of the existing systems software is in core; the rest is disc-resident.

In a C.B.L. system the application segments seem to fall into four distinct parts:

a) Teaching segments (using both pre-stored and adaptive teaching (see 2).
b) On-line calculating facilities which can be called from within the C.B.L. system.
c) System Control. There must be some facility by which a 'controller' can determine how the system is being used and some facility which gives him control to rearrange teaching stations, to terminate lessons, etc.
d) Systems software such as file amenders (for modifying computer and educational program texts), assemblers, teaching material pre-processors, macro-generators, etc.

In various segments of the system it will be necessary to collect information about the allocation of systems resources. This information will be used to provide an insight into the operational efficiency of the system: it is conceivable that some form of accounting may also be provided.

Most of the application segments will need to manipulate information which, because of its bulk, will have to be disc resident and so it is necessary to design and implement a disc based filing system.

The disc system has been designed such that it can be used both by the multi-access systems and by background jobs. Both binary and text information is kept in chained files and a common systems index is held at a known block on the disc. This system also allows for disc blocks to be allocated to the multi-access system for use as scratch store (e.g. for suspended segments). In fact, the layout of the disc is such that it could co-exist with a second filing system if this were necessary.

The executive has to contain sub-routines which allow reading and writing of blocks of information. A further useful facility, still to be implemented, is the ability for one program to inhibit transfers from all others until it gives the

corresponding 'all clear' instruction. These two executive commands are required when one is updating information accessible to a large number of users.

4.5 *Background Programs*

Whenever possible background programs should be run simultaneously with the on-line system in order to utilize the c.p.u. efficiently. Obviously, off-line job should run at a lower priority than the on-line system. The sort of jobs which might be run as off-line jobs in a C.B.L. system are the processing of student records, compilation of statistics and perhaps the off-line generation of teaching material using macro-generation techniques.

5. HARDWARE FOR THE C.B.L. NETWORK

The Leeds Educational Network, L.E.N., is in fact being based around a Modular One computing system (COMPUTER TECHNOLOGY LIMITED (1969)). The design of this machine is such that the hardware can be readily expanded and that additional and new types of peripherals can be fairly easily interfaced to the system. The very important feature of autonomous transfers (noted in 3) for the peripherals is achieved in this system by means of the communications multiplexer (fig. 2) which transfers information to and from core and only interrupts the central processor unit when such a transfer has occurred. This special purpose piece of hardware handles information in serial form and is able to cope with a number of lines using simultaneously a variety of speeds from 200 to 4800 baud.

6. THE DESIGN OF THE TERMINAL NETWORK

Recently, a thorough survey of peripherals required for the different teaching areas has been made (Sleeman (1970)): for the present we plan to attach 8 teletypes and 4 audio-visual devices to L.E.N.

The audio-visual device in fact works on a parallel signal and therefore a serial to parallel/serial converter has to be interposed between the device and the communications multiplexer. Remote devices would normally be attached via modem links to the communications multiplexer. As both the teletype and the audio-visual device are relatively slow, it seemed sensible to attempt to drive both of these over a single GPO line. In order to do this a Character Recognition and Line Switching device (CRLS) has been developed. This spots certain control characters and then switches to the appropriate terminal device. (It is possible to control 4 "serial" devices,i.e. TT and 4 "parallel" devices with the CRLS.) The CRLS device has been designed so that it also carries out serial/parallel conversion.

(On our configuration, peripherals which are in the central laboratory can either be attached directly to one of the spare input/output channels or into one of the channels from the communications multiplexer.)

The computer will not only be able to send messages to the audio-visual device but it is proposed that the device will contain a status word which the computer will read. From this the controlling teaching program will be able to infer the status of the device (e.g. rewinding forwards, rewinding backwards, error condition etc.) The following would be the sequence of commands where an audio-visual presentation is required to accompany a message on the TT and to precede a response from the TTY:

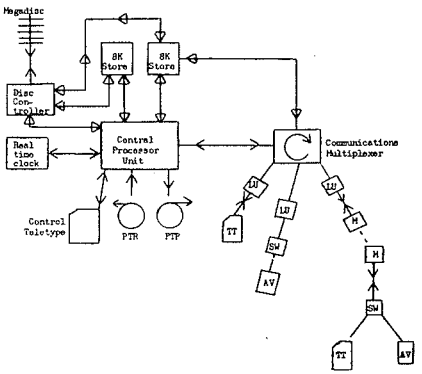


Figure 2

LEEDS MODULAR ONE CONFIGURATION

Code: AV Audio Visual device
LU Line Unit
M Modem
SW Character Recognition and Line Switching device
TT Teletype
↔ Direct Link
⇔ High-speed Link

SWITCH LINE TO A-V DEVICE
SEND COMMAND TO A-V DEVICE
READ STATUS (Check message received – retransmit if necessary)
SWITCH LINE TO TTY
SEND MESSAGE TO TTY
SWITCH LINE TO A-V DEVICE
READ STATUS (Loop until frames displayed)
SWITCH LINE TO TTY
READ RESPONSE FROM TTY

Because of the modular design of the communications multiplexer and the features of the CRLS device it will be a fairly straight forward matter to interface further devices into the system as they become available.

7. SYSTEMS RELIABILITY

As the system will be servicing a large number of users, many of them situated at some distance from the system, it would be very desirable for the system to have some fail safe or at least fail soft capabilities.
The National Physical Laboratory (see DAVIES (1968)) have recently considered such systems in detail and have suggested possible configurations for remote access computer networks.

As our present configuration has only one c.p.u. it will not be possible for us to provide such a capability. However, we feel that any considerable expansion of the system should include an additional processor and that the systems software should be amended to provide a fail soft capability.

8. CURRENT STATE OF THE SYSTEM

At the time of writing we have only taken delivery of the most basic parts of the system and we still await the disc, disc controller, communications multiplexer and the real time clock. The design for the whole system is now complete and the preliminary version of much of the system described above has been written and partially tested using simulation techniques. The initial system will use a version of C.T.L.'s multi-programming executive modified to provide disc read/write facilities and the ability to control the communications multiplexer (see COMPUTER TECHNOLOGY LIMITED (1969)).
In order to increase the performance of the C.B.L. system we envisage that it will be necessary to rewrite the executive along the lines indicated above.

As usual with disc systems, both because of systems back-up and systems maintenance considerations, we have also provided a suite of disc maintenance programs. In our case because of the lack of suitable auxiliary storage we have been forced to use paper tape as a back-up medium. Naturally, we would welcome the opportunity to change to a more convenient auxiliary storage device such as magnetic tape or removable discs.

9. CONCLUSIONS

In the above paragraphs we have set out what we intend to implement and have given some indication of what has been achieved at the time of writing. At the Conference we propose to report experience which we subsequently will have gained from the working system. This, and subsequent experience will undoubtedly point to the shortcomings in the system design but, hopefully, the basic design is such that the system can subsequently be easily improved.

ACKNOWLEDGEMENTS

The multi-access system proposed for Modular One has been greatly influenced by the experience gained with the ELDON system, and it is with great pleasure that we acknowledge our debt to colleagues in the Electronic Computing Laboratory and the Department of Computational Science.
Further, we are grateful to Mr. R.J. Ross (School of English) for permission to discuss details of the random-access audio-visual device, and Computer Technology Limited for various helpful discussions.

REFERENCES

Computer Based Learning Project, The University of Leeds, Internal Document AO1: "The Leeds Modular Author Language", 1970

Computer Technology Limited: "Modular One User Manual" 1969

DAVIES, D.W., 1968, Proceedings of IFIP Congress, p.D11

de DOMBAL, F.T., J.R. HARTLEY and D.H. SLEEMAN, 1969, Lancet, p.145

FOX, J.D., R.J. ROSS, J.R. HARTLEY and D.H. SLEEMAN, "Computer Assisted Language Learning: a Report on Work in Progress at Leeds University". In "Aspects of Educational Technology", vol. 3, 1970, (Pitman) p.347

SLEEMAN, D.H. "The Design and Function of Terminals in a Computer Based Learning Network". To be published in the proceedings of Computer Graphics 70 Congress at Brunel University, April 1970

WELLS, M.D. HOLDSWORTH, A.P. McCANN, A.A. HOCK, C.V. HARMAN, D.K. JOHNSON. "Eldon 2 Manual", Electronic Computing Laboratory, The University of Leeds, 1970

WOODS, Patricia, J.R. HARTLEY, K. LOVELL and D.H. SLEEMAN, "The Teaching of Arithmetic Using a Computer Based System". In "Aspects of Educational Technology", vol. 3, 1970, (Pitman) p.378.

THE USE OF DECISION TABLES IN PREPARING COURSE MATERIALS FOR CAI

R. Clay Sprowls

1. INTRODUCTION

Many ways are used to prepare and present course materials to serve the different needs of CAI instruction. Certainly authors who wish to inform others about their courses need an overall verbal description. Authors who wish to review another course would like a general statement about the flow of materials, especially the strategy used in the instruction. Finally, any serious committment to a CAI implementation demands details about individual text statements, possible answers, and the author's method of handling the student interaction.

The preparation of such materials has been a bottleneck to the more widespread use of CAI as a teaching vehicle. One reason for this has been that authors have generally needed a knowledge of the computer and of some programming language. Unless one wished to become heavily committed to learning about computers and programming, he could not actively participate in a CAI development. Only after having learned about computers and a programming language is the author prepared to do battle with the material itself. This means that he is ready to decide upon his text presentations, the possible set of alternative answers, the strategy used in handling these answers, the branches to reinforcing lessons, etc. At this point, not only does overall logic become complicated but it must all be organized with respect to the specific computer and author programming language at his disposal.

Programming languages for CAI now number in the dozens and more seem to be developing with passing time. Since standardization on one language seems highly unlikely, some means in needed to simplify the author's task in writing his course without tedious involvement with the computer. This means should also prevent the obsolescence that comes from having to redo the basic presentation of his course merely because of a computer or a language change. Demands exist for use of CAI instructional materials that have already been developed. Unless the computer system is compatible and the programming language the same, such exchange is prohibited. Authors working with different equipment and languages are able to trade materials only with difficulty when the course presentation is tied to the only available CAI system. As a result, many potential CAI anthors are discouraged before they begin.

One compelling author need is a mode of preparing course materials that:

(1) will express complex decision logic;
(2) is independent of computer equipment and programming language facilities;
(3) will 'speak for itself', that is, is a good documentation system to present details of what is done in a particular lesson or lesson segment.

The purpose of this paper is to present Decision Tables as a potential methodology for preparing CAI course materials because they inherently meet these criteria.

The decision table is a very useful technique in the design of management information system. As a management tool it facilitates description of the complex logic of the information and processing flows, and the myriad of decision points and paths that branch from them. Business managers have learned to work with them independently from computers and programming languages to describe the information system logic. System analysts and managers are able to state the various alternative conditions that may be present and the actions that are to be taken with each set of conditions.

Because they focus on the overall logical construct of the problem and not on computer systems or computer programming they have the same potential usefulness in the design of CAI courses. They can provide authors who desire to devote their energies to preparing their course materials rather than to studying the computer, an efficient means to express the logic of the teaching strategy.

2. DECISION TABLE FORMAT

The general format of a decision table and the four basic elements which constitute its structure are shown in Figure 1. The double horizontal and vertical lines separate the four basic elements. The **condition** area is **above** the horizontal; the **action** area is to the right. These relative positions remain constant in any discussion of decision tables.

Conditions and actions reflect an 'IF... THEN' relationship. No conditions may appear in the action area; no actions may appear in the condition area. The entry is used to show, in each case, whether the particular condition is true or 'Yes' (Y), false or 'No' (N), or not pertinent (blank), and whether a particular action is to be taken (X). This notation is used in Figure 3.

The basis of most CAI lessons is an author presentation of some material to the student and an analysis of the subject response. The interchange or interaction between the author and subject is an IF ... THEN relationship. If the response is such-and-such, THEN such-and-such action is to be taken. The simplest case is a True-False question. IF the subject response is 'correct' THEN this is recorded, and the student may go on to the next question. IF the response is 'incorrect', THEN this is recorded and the student may go on to the next question. In the T-F situation, the strategy may be more complicated. Some encouraging comment may follow a 'correct' answer. Some direction to reread the question and try again may be given after an 'incorrect' answer. Even in this simple case, the overall teaching method may become quite complex and a verbal description tedious to follow.

Decision Tables permit an author to record his strategy in tabular form. The author lists the set of alternative

conditions in the condition stub and the set of alternative actions in the action stub. He then uses the entry sections of the table to record the actions that are to follow the conditions when they are present.

3. EXAMPLE OF A CAI QUESTION

Figure 2 is a flow diagram for one CAI question taken from a geography course reported by Gordon Fielding (1968) of the University of California, Irvine. The flow chart uses symbols adopted as more or less standard in the computer field: rectangle, circle, and diamond, (Flow charting is widely used as a method of presenting instructional material.) Figure 2 also has terminology from the Coursewriter programming language. PR, QU, CA, WA, WB, Un and TY are actually Coursewriter commands for PResent, QUestion, Correct Answer, etc., and as such they become explicit directions to a programmer. Fielding's flow chart illustrates not only the teaching logic and the material to be presented, but also the actual language commands used in his implementation.

The same instruction logic is shown in Decision Table form in Figure 3. This is an example of a limited entry form because all of the conditions and actions are in the stub and the entry is limited to showing for each case whether the particular condition is true, false, or not pertinent, and whether a particular action should be taken.

The table shows at a glance that there are five conditions and nine possible actions which may be taken in combination. Six rules will satisfy all of the IF ... THEN relationships. The rules are interpreted as follows:

(1) IF 'next question' is true, THEN 'text' and 'question' are presented. The quote marks are used here to signify that something is to be typed or displayed in some fashion.

(2) IF the answer by the student is 'discrete', THEN the 'correctmessage' is written and the next question is called for.

(3) IF the answer is 'contingent', THEN the 'no-message' is written followed by 'try again'.

(4) IF the answer is 'continuous', THEN the 'incorrect-message' is written followed by 'try again'.

(5 and 6) IF the answer is unspecified, that is, not one of the three key words given in the text, THEN the first time the student is asked to sign off if he has not read the material; the second time he is to reread the question. The parenthesized numbers (1) and (2) are used here to designate the order that is taken into account in the lesson.

Decision table vocabulary and grammar are left to the discretion of the user. Just as management adapts decision tables to the subject matter of management information systems, so may CAI authors adapt to their own subject matter and the conventions that surround it. One suggestion has already been made: that of numbering the preferential order in which certain conditions are to be taken into account. Another is an extension of the decision table format on the left (the stub) to include room for text in quotation marks. A revised decision table that incorporates both suggestions is shown in Figure 4. The decision table is now complete with respect to the overall logic of the question treatment and also its specific content.

4. CAI LESSON AND COURSE

The decision table presentation is computer and programming language. independent. How the geography question illustrated in Figures 3 and 4 is translated into a particular programming language for a given computer system is up to the programmer. The author has presented his teaching approach for the subject without regard for either a computer or a programming language. Another geography teacher may view this question and decide whether the wishes to use it intact or modify it in someway. A modification can be made on the decision table form.

The example given here is for only one question. A complete lesson becomes a series of decision tables, each one labeled for reference.

Since the need exists to present an overview of the instructional process at a less detailed level than individual questions, a decision table approach can be used to show the general logic of the course. Individual lesson decision tables become imbedded in the course decision table; individual questions become imbedded in the lessons.

Implied in the foregoing discussion is a claim that the decision table form is simpler to prepare and understand than the flow chart. It is simpler to prepare because a standard recording form is used for each table and an author need merely write in the stubs his list of conditions and actions, and under each rule check off the set of conditions and resulting actions. It is simpler to understand because the tedious tracing of flow chart paths through various routes and symbols is made unnecessary.

5. ANOTHER EXAMPLE

A second example is derived from a CAI Physics course now being implemented by Bork (University of California, Irvine and Sherman (University of Michigan). They are preparing the same course on two different computers in two different programming languages. They have recognized the need for a chart showing the general flow of materials in a section on Engergy Conversation and for a standard set of symbols.

Their notation is T for text statement; U for user response; M for match of user answer to text question. Subscripts are attached to the symbols to reference different statements, responses and matches in the instructional sequence. The details of the text statements and the alternative matches are not given in their general discussion. Since they are numbered, they are written in a separate dictionary from which they are taken when the instruction is programmed for the computer.

The heart of their presentation is summarized in Figure 5 where the matching rules are shown. The first two lesson segments T_1 and T_2 are shown in the flow charts of Figure 6. These are relatively uncomplicated as compared with some later segments.

The same two lesson segments are shown translated into decision table form in Figures 7 and 8.

Figure 8 shows a more complicated decision logic in the condition entry. IF $M_{2\,1}$ and $M_{2\,2}$ are both true, THEN one action is taken. IF $M_{2\,1}$ and $M_{2\,2}$ is false, THEN another action is taken. The fact that the rules are to be checked in order is written under the rule heading rather than to have an entry and a rule for the N alternative to each Y and the Y alternative to each N in the condition entry. This is a departure from the usual convention that only one rule in a decision table may be carried out each time a table is examined. It is a departure which further shows the flexibility of decision tables and one that will not cause any difficulty.

Some of their flow charts cover more than one page and embrace as many as four levels of matching with some levels having as many as five matching alternatives. (Only the two cases are shown here because the intent is to illustrate decision tables and not present an exhaustive treatment.)

6. SUMMARY AND CONCLUSION

Examples of the decision table approach to the design of CAI instructional materials have been given using two lesson segments from actual CAI courses. Decision tables that are so useful in the desigr. of management information systems appear to have a potential for solving one of the bottlenecks of author committment to CAI: the need for a simple means to prepare materials. This is because they.

(1) will express the complex instructional logic;
(2) are computer and language independent;
(3) are self-documenting;
(4) can be adapted to an author's grammar for text and response
according to his subject matter needs.
(5) are prepared in a standardized form that is easy to read and interpret;
(6) are adaptable to course, lesson and individual question presentations.
(7) will facilitate interchange among authors working with different equipment and languages.

This is a long list of claims. It behooves authors of CAI materials to investigate this recognized managment tool.

IF ... Condition Stub — Condition Entry
THEN ... Action Stub — Action Entry

IF ...	Condition Stub	Condition Entry
THEN ...	Action Stub	Action Entry

Figure 1 Decision Table Format and Elements

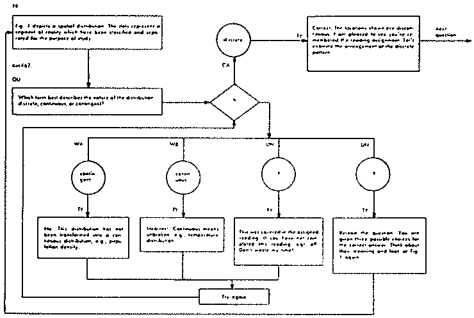


Figure 2 Flow Chart for One Geography Question

Rule No.	(1)	(2)	(3)	(4)	(5)	(6)
Next Question	Y					
'Discrete'		Y				
'Contingent'			Y			
'Continuous'				Y		
Unspecified(1)					Y	
Unspecified(2)						Y
'Text'	X					X
'Question'	X					X
'Correct-message'		X				
'No-message'			X			
'Incorrect-message'				X		
'Assigned Reading'					X	
'Reread-message'						X
'Try Again'			X	X	X	
Go To Next Question		X				

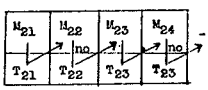
Figure 3. Decision Table for Geography Question

Rule No.	(1)	(2)	(3)	(4)	(5)	(6)
Next Question	Y					
'discrete'		Y				
'continuous'			Y			
'continuous'				Y		
'discrete(1)'					Y	
'discrete(2)'						Y
'Fig. 1 depicts a spatial distribution. The data represent a segment of reality which have been classified and separated for the purpose of study.'	X					X
'Which term best describes the nature of the distribution: discrete, continuous or contingency?'	X					X
'Correct. The locations shown are discontinuous. I am pleased to see you've remembered this reading assignment. Let's examine the arrangement of the discrete pattern.'		X				
'No. This distribution has not been transformed into a continuous distribution, e.g., population density.'			X			
'Incorrect. Continuous means unbroken, e.g., temperature distribution.'				X		
'This was covered in the assigned reading. If you have not completed this reading, sign off. Don't waste my time.'					X	
'Correct the question. You are given three possible choices for the correct answer. Think about their meaning and look at Fig. 1 again.'						X
'Try again'			X	X	X	
Go To Next Question		X				

Figure 4 Suggested Decision Table Format for CAI

M_{11}	M_{22}	M_{1n}

- three possible matches the user may answer a question with. Search from left to right.



- If M_{21} appears, the computer is to respond with statement T_{21} and go on to check for the other responses. If M_{22} does not appear, go directly to M_{22} and check for it. The grounds for acceptability may be changes as with M_{22} and M_{24}.

Figure 5 Notation for Matching Rules

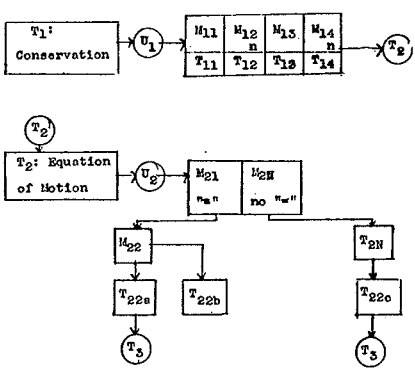


Figure 6 Lesson Segments T_1 and T_2

Rule No. (Checks in Order)	(1)	(2)	(3)	(4)	(5)
Enter Section T_1	Y				
M_{11}		Y			
M_{12}			N		
M_{13}				Y	
M_{14}					N
T_1 : conservation	X				
T_{11}		X			
T_{12}			X		
T_{13}				X	
T_{14}					X
Check M_{11}	X				
M_{12}		X			
M_{13}			X		
M_{14}				X	
A to T_2					X

Figure 7 Decision Table for T_1

Rule No. (Checks in Order)	(1)	(2)	(3)	(4)
Enter Section T_2	Y			
M_{21} "="		Y	Y	
M_{22}		Y	N	
M_{2N}				Y
T_2 : Eq of Motion	X			
Check M_{21}	X			
Check M_{22}				
T_{22A}		X		
T_{22B}			X	
T_{2M}				X
T_{22C}			X	X
Go to T_3		X	X	X

Figure 8 Decision Table for T_2

DEVELOPMENT OF AN AUTOMATED DIAGNOSTIC COLLEGE PLACEMENT TEST IN MATHEMATICS

Anita S. West

1. PROVIDING APPROPRIATE OPTIONS

1.1 *Placement data*

The increased number of students entering institutions of higher education, and the concomitant increase in their diversity of academic preparation, have resulted in additional burdens for departmental administrators responsible for the placement of freshman students into their academic programs. Selecting the most useful and beneficial educational programs for these freshmen without wasteful repetition, demoralizing failure, or compromise of standards can only be achieved with any real rationale when there is good and sufficient data for such administrative decision making. Departmental administrators express the need for assistance in administering and evaluating placement examinations and in designing a freshman curriculum to meet the varied and changing needs of the entering college population. Innovative programs utilizing electronic digital computers, video tapes and numerous combination audio-visual devices are among the developing methods for improving a variety of school and college services and may be of special value in college placement.

1.2 *The role of testing*

Computer testing and diagnosis are investigated as the first steps in a sequence of testing, computer remediation and retesting for incoming college students. The purpose of a diagnostic test is to isolate deficiency areas so that a student may be directed into a short program of individualized instruction before starting his freshman sequence courses. The purpose of the development of a computer terminal administered test is to provide an efficient method of diagnosis and reporting, to relieve the university of this task and to provide the opportunity for future students to be tested before they come to college so that they might repair their deficiencies individually if they so choose. The test designed and evaluated in this study is being used by the University of Denver, Department of Mathematics, in place of the customary placement test, but it is expected that its value to the student and to the institution is considerably more than as a placement instrument. The results of this test are not for the end purpose of passing or failing a student, but as the first step in his continuing education in mathematics. The automated diagnostic mathematics examination is designed to indicate to the department and to the high school senior, immediately upon completion of the test, those topics in which he evidences need for some further study before continuing his mathematics education with Calculus at the university level. Further, the programmed test recommends texts and chapters to the student for such self study if he chooses this form of remediation. Computer assisted instructional units in mathematics are being developed now at the University of Denver and will eventually be available to the students as an alternative method for repairing deficiencies. Research on the reporting of results to students shows the advantages of immediate feedback to him. He is apt to take more initiative and responsibility for his own learning and becomes more self-directive, LEE (1966).

There are implications in the development of this kind of instrument for teachers, administrators, and those who develop teaching materials, but these go beyond individual student progress. The emphasis here, is on the student and the role the test can play in assisting him to reach his highest level of achievement, given a particular educational milieu.

2. CONSTRUCTING A DIAGNOSTIC MATHEMATICS TEST

2.1 *Selection and description of diagnostic areas*

There are several somewhat compartmentalized bodies of knowledge with which a student needs a degree of familiarity and facility in order to expect that he can succeed in college Calculus. It might be that even with these skills he will not do well, but it is almost certain that without them he cannot succeed. These areas of subject matter are included in courses normally taught in high school Algebra and Trigonometry in this country. The National Longitudinal Study of Mathematical Abilities (NLSMA) which had as its main purpose the evaluation of mathematics achievement throughout the nation, isolates subject matter normally covered during two years of algebra and one semester of trigonometry, WILSON et al (1968). In addition to the NLSMA study findings, standardized tests, texts and discussions with college and high school instructors provided the sources that were valuable in the selection of areas for diagnostic analysis. The derivation of differentiation formulas is found near the beginning of most Calculus texts. An examination of the presentations of these derivations point out the necessity for a well developed capability for algebraic manipulation so that the concepts of differential caluculus, developed with the use of algebra, can be clearly seen and understood by the student. Concepts of the structure of the real number system, logic and set theory, are not omitted as special topics because they are considered unimportant, but because they are considered rather as an aid to the student for a facilitated development of algebraic achievement of the nature necessary for beginning calculus, and students who are considering studying mathematics beyond calculus are offered a more rigorous course in mathematical analysis. A test of techniques is considered to be sufficient and appropriate at this time.

The results of these inquiries and studies was the identification of the five following competence areas for diagnostic analysis as being significant in the determination of the mathematical background necessary to complete a college course in Calculus.

1. Simultaneous equations, powers and roots.
2. Factoring and quadratic equations.
3. Absolute values, inequalities and solution sets.
4. Graphs of functions, solution of right triangles and radian measure.
5. Logarithms, change of base, arbitrary base.

2.2 *Selection of items and test format*

Standard item selection procedures were employed for choosing test questions. A pool of items was constructed after studying several college placement tests, standardized tests and faculty recommendations. Preliminary administrations of the test and item analyses resulted in the selection of 25 test items, five items in each content area, considered a minimum for diagnosing competency. Therefore, the difficulty of the question rather than the number of questions, associates the importance of the area with the ability to begin Calculus. The test utilizes a multiple choice format, thought to be most suitable for computer administered examinations.

3. COMPUTERIZING A DIAGNOSTIC MATHEMATICS TEST

3.1 *Computer software*

The University of Denver program library includes an ALGOL symbolic language version of a Coursewriter program developed at I.B.M. for the purpose of facilitating automated course instruction. William Eichelberger, Manager of the University Computer Center, wrote the ALGOL version and entitled it, TEACHER/DU. TEACHER/DU provides the structure for writing computer assisted course instruction with almost no knowledge of computer programming. This program for the Burroughs B5500 provided basis for a new program written for this study with the programming assistance of Mr. John Skelton. This modification, known as DU/TEST, uses all the instructions and constructs available in TEACHER/DU and includes additional options required for the diagnostic test. It is useful for diagnostic course instruction as well as testing in mathematics and a variety of other subjects.

3.1.1 DU/TEST Program Description

DU/TEST structures provision for typed messages and questions to students. It also provides the teacher-programmer with the ability to re-teach, continue, or reinforce a concept depending upon correct anticipated answers, alternate forms of correct anticipated answers (e.g., Your answer is correct, but you misspelled Mississippi,) anticipated incorrect responses, and instructions for unanticipated responses. The teacher-programmer need not have any real knowledge of programming to use these options, but follows a set of instructions clearly outlining to him the manner in which to write such a course. Branching in the IBM version is limited to options following question responses. This does not provide the flexibility required for a diagnostic test or course. DU/TEST permits GO TO (label) IF statements which alter the order of the program by shifting to any labeled instruction, depending upon the results of the IF test. These can be used before as well as after questions. No provision for tallying responses is included in the previous program. DU/TEST defines variables for use in tallying correct responses to questions in any given area, so that after a question is answered correctly, the variable which scores that sub-test can be incremented by an add instruction, also new to this version of the program. A further modification has to do with the use of another defined variable that is used as a random number generator. In order to protect the secrecy of a test available to any teletype user with a valid charge number at the computer center, it was considered necessary to write a pool of items to be available for each of the twenty-five questions. For this study, five items were written for every question, and are selected at random as questions are put to the students. The program can multi-process and correctly score any number of students taking the test at different terminals simultaneously, the only limitation being imposed by the hardware.

3.1.2 Math Test

MATH test is the automated version of the written diagnostic examination. The test itself is input as data into the DU/TEST program. It selects at random one of a number of items for each question, tallies responses, and gives additional sub-test questions until an assessment of achievement in that area can be deduced. At the conclusion of the test, it prints out messages to the student advising him of any achievement deficiencies he might have before continuing mathematics with Calculus.

The variables, VI, V2, V3, V4, and V5 represent the subtest score tallies. After a question in one of these areas, i, is answered successfully, the variable tallying that area score is incremented by a command, AD 1 to Vi. The test is now being scored by an examination of that variable for the value of zero (fail) or three (pass) once three questions are asked in the area. The fourth and fifth questions are asked only as necessary. There is no need for questions in any area to follow one another consecutively and they are scattered troughout the examination. Fig. 3.1 illustrates the input required for DU/TEST with an excerpt from MATH. The variable, VO, is a random number between zero and nine.

4. ADMINISTRATION AND VALIDATION OF AN AUTOMATED TEST

4.1 *Problems and limitations of a computer administered test*

The most severe restriction of the automated form of the test is the limited character set available at the teletype. This is probably at least as severe a limitation for a mathematics test as for any other department test application. Radical signs, superscripts for exponents, subscripts for logarithm bases, fractional representations, etc., must be represented to students in forms other than as they appear in textbooks and on most examinations. Answers must be given by means of the teletype keyboard, which means the examinee must type the letter corresponding to the correct answer, and then another special character which allows the computer to process this response. For students unfamiliar with a typewriter or teletype keyboard it was possible that this mode of response might have some adverse affects on test outcomes. A third problem had to do with the inflexibility of a computerized examination. Test questions that are difficult

cannot be skipped and returned to at a later time as on a written examination. Answers cannot be reconsidered and changed once the process key has been typed. Additionally, teletypes in use at the University and in the participating high schools are fairly noisy while they are running. As a comparison they are considerably louder than are electric typewriters.

These were the conditions that presented potential problems in the feasibility investigation. An instruction sheet was given to participating students several days before the examination. This was written and distributed for the purpose of acquainting students with the kind of an examination they would be taking, and generating a positive attitude toward it. As indicated on this sheet, fig. 4.1, instructions are summarized again for the student when he is seated at the teletype. A sample question utilizing the asterisk construct for exponentiation follows the automated instructions and must be answered correctly before the program proceeds to the first test question. In the event the question is answered incorrectly, anticipated responses are used to direct messages re-explaining the test to the student. These instructions and the sample question are shown from an actual teletype output in fig. 4.2. A questionnaire aimed at assessing the extent and determining the cause of any problems connected with the automating of the test was given to students following the teletype test. The form is shown in fig. 4.3. The results of the questionnaire indicated a neutral to favorable attitude toward the automation of the test.

4.2 *Administration*

The experimental test was administered in Denver, Colorado during the months of April and May, 1969.

4.2.1 Population Sample

Over 700 high school juniors and seniors from four Denver and suburban high schools participated in the development and evaluation of the automated test. The student population for the computerized version of the test was chosen similar in every way to a sample of 580 students who took a written version of this same examination. No attempt was made to control the proportion of juniors and seniors nor the proportion of boys ans girls, neither for the written nor for the automated version. The schools from which the samples were drawn were not classified or weighted according to their urban or suburban characteristics, school size, etc. Each school used in the study, however, had a sufficient number of students to offer a three track program in mathematics, and all the students were selected from the college bound mathematics program. All the schools used in the testing program were in the greater Denver area, except for a sample of 97 students from Bedford, New York. As further tests of freshman students are given, sample differences should be noted in order to make recommendations for revisions of the diagnostic goals of the examination.

4.2.2 Test Conditions

Tests were administered in two different settings after the multi-process (time-sharing) capability of the software and hardware had been checked out by members of the staff who ran the test program simultaneously at four terminals of the University. Twenty student volunteers with the same general background as those from the actual sample were invited to the University to take a preliminary test using the D.U. teletypes. Their comments regarding the testing conditions, the sample question, instructions, and the format of the questionnaire resulted in some minor changes. These students were brought in to record the kinds of questions asked by a student with a monitor present to respond to them and to time the tests. No method for controlling question-response time has been instrumented to date. The time required for printing out instructions and questions is about eight minutes. This was added to the 45 minute administration time and an additional seven minutes was allowed for adaptation to teletype characters, etc., giving a one-hour administration time, which proved to be more than adequate.

On May 10, 1969 forty students from Lakewood High School came to D. U. four at a time, at 50 minute intervals from 8:00 in the morning The high school was visited during the week prior to the test date and students were given the instruction sheet shown in fig. 4.1. Because of the telephone line limitation of the B5500 at that time, only four teletypes could be used simultaneously, although there are many more available on campus. With the help of graduate assistants these tests were also monitored. Although all phases of software and hardware had undergone debugging, there were some problems connected with teletype performance that required the assistance of monitors. Altogether four tests were lost because of hardware difficulties. On the following Saturday, twenty-six students from Thomas Jefferson High School came to the University, and the tests were administered again. Scheduling was prompt and there were no difficulties. Students filled out the questionnaires and made verbal comments to the investigator, which together with the results of the test, indicated that the examination was satisfactory for continued testing, without monitors, as it was originally designed to be administered. A computer charge number for off campus testing was then assigned, and high school students were instructed in the procedure for calling the University Computer Center and logging in on the teletypes located in their own schools. The procudere was the same in both schools. The student came to the school Mathematics Study Room or the Mathematics Faculty Room where the teletypes are located. He went there during his free period or mathematics period, and took the terminal-administered test. Students were not monitored during the test, but they had only 50 minutes (one period) to complete the test and would not have had time to refer to textbooks or notes even if they had shown a desire to do so. Classroom teachers seemed confident that their students would accept the responsibility for self-monitoring and daily visits by the investigator to the schools indicated that they were correct in their assessment of their students. The messages to the student regarding his demonstrated abilities were masked during these tests. Students were told that because the test was still under development it would be necessary for their teachers to go over the results before making recommendations to them. However, variables that normally control the selection of the messages were printed out with their magnitude and this information was sufficient to give the instructors a full summary of results to report to their students. Fig. 4.4 shows the final portion of a sample test.

4.3 *Validation*

After a written form of the diagnostic placement test was demonstrated to be reliable and valid, it was

administered to 580 high school juniors and seniors. The results of the automated version, described in §2 were then tabulated. The null hypothesis was then proposed: there is no difference between the results of the written and automated test. The chi square coefficient was considered to be most suitable for comparing pass-fail decisions for the total test as well as for each of the five sub-tests. The statistic, chi square, is useful in determining whether discrepancies in grouped data are as great or greater than might have resulted merely from fluctuations in the sample, WERT et al (1954). Two way classifications of automated and written administrations with pass and fail groupings formed the rows and columns of a four celled contingency table. Table 4.1 shows this information for the total test results. The chi squares calculated from sub test and total test tables are shown in Table 4.2. In order to reject the hypothesis that there was no difference between the automated and written examinations for the one degree of freedom analysis, chi square would have to be greater than 3.841 at the 5 per cent confidence level. The largest chi square calculated was 0.044. There was no evidence to reject the hypothesis and it was accepted.

Table 4.1

2 x 2 Contingency Table for Pass-Fail With Written and Automated Test

	Passed	Failed one or more units	Total
Written	244	216	460
Automated	51	46	97
Totals	295	262	557

Table 4.2

Tabulation of Chi Square Coefficient for Five Sub-Tests and Total Test with Written and Automated Samples

	Chi Square
Sub-Test 1	0.044
Sub-Test 2	0.00015
Sub-Test 3	0.00066
Sub-Test 4	0.0045
Sub-Test 5	0.0075
Total Test	0.0070

To reject hypothesis at the 5 percent level, one degree of freedom, chi square must be equal to or greater than 3.841.

5. CONCLUSIONS AND IMPLICATIONS

5.1 *Conclusions*

The investigation has given evidence that it is indeed feasible to automate a diagnostic placement examination. This has been demonstrated with the correlation between a reliable and validated diagnostic instrument in its two forms – written and automated. The results of this study are evidence that college departmental placement testing can safely and effectively be left to computerized methods. The success of such a program, however, has been shown to depend upon an investigation of appropriate content for diagnosis of difficulties, a study of the literature to select items which isolate content area and are interpreted by students in the way they were intended, and a testing program that anticipates negative reactions and prepares to compensate for them. The program software for constructing such a test is now available for a variety of teaching and testing purposes and provides the programming flexibility required for diagnostic teaching and testing. In the fall of 1969 the University of Denver administered both its existing placement examination and a computer scored diagnostic test to its 300 entering mathematics students. Placement decisions were made on the basis of the results of the existing test, but follow-up studies of these students and especially the six students whose scores differed enough to indicate a different decision regarding their placement, are being performed and will be available in June, 1970. The major obstacle for utilizing an all computer administered placement examination is the lack of sufficient hardware to process the students.

5.2 *Recommendations*

In addition to a constant reconsideration of the subject matter tested for diagnostic analysis, the following recommendations are made.

5.2.1 Providing Alternatives

The automated diagnostic test results point to individual deficiencies and weaknesses. In order to make full use of the information gleaned from such a test, it would be well to consider the development of systematic methods for allowing students to repair such deficiencies. The development of computer assisted instructional (CAI) units for students who fail to pass only one or two of the sub-tests seems a logical step in the development of individualized alternatives. A CAI program of remediation which could be given during the summer months or concurrently with the Calculus, could eliminate many weeks of non-credit classes for the student who requires only a small portion of its content. At the same time the CAI alternative would free Calculus instructors from reviewing algebraic and trigonometric concepts, which they must do now for the minimally prepared students who were allowed to register for Calculus simply because of the lack of other reasonable choices for instruction.

5.2.2 Modifications and revision of items.

The diagnostic test requires a systematic program for evaluation and revision, as does any good testing program. The ease with which items can be altered or changed using the MATH test limits the problem of time and cost for the revision and updating of the test, making it more likely that such revisions will be made as often as necessary. In the event that a question or a number of questions is to be modified or replaced, the cards on which the questions appear are pulled from the card deck, repunched, and inserted back into the proper place. The card deck is then taken back to the Computer Center and is reloaded onto the disk, replacing the old edition without any publication costs or delays. If disk modification is available the matter is further simplified.

5.3 *Implications*

The implications of the success of this test go beyond its use as a placement instrument. Diagnostic teaching, long

an ideal, becomes a practical procedure with the advent of the computer test. In the area of college counseling, automated diagnostic tests can provide, quickly and accurately, the data that is needed for helping a student choose a relevant and suitable sequence of courses for him. As colleges tend toward more liberal and flexible degree programs, the need for diagnosis of individual needs persists and grows. The feedback from those tests to teachers and those engaged in curriculum studies should be of extraordinary value in evaluating instruction, materials, and content, by providing the hard data to add to personal evaluation. Such data would not be intended to replace knowledgeable judgments, but to broaden the input from which decisions are made. It appears that student populations will continue to grow more numerous and more varied as social and economic conditions continue to change. The university which seeks to prepare for its changing role as pivotal institution in a changing society will attempt to provide the varied educational experiences necessary for its future population. Computer testing is one way to gather more information about the needs of individual students. It is only the first step toward filling those needs.

REFERENCES

1. Lee, Dorris May, 1966. Diagnostic Teaching (National Education Association) Library of Congress Catalog Number 66-30196, p. 15.
2. Wert, J.E., E.O. Neidt, and J.S. Ahman, 1954, Statistical Methods in Educational and Psychological Research. (New York: Appleton-Century Crofts, Inc.) p. 146.
3. Wilson, J.W., L. Cahen and E. Beagle (editors), 1968. NLSMA Reports, (Stanford: School Mathematics Study Group).


```
   RESPONSE FOLLOWED BY A LEFT POINTING ARROW.
UM TRY ONCE MORE.
UM YOU SEEM TO BE HAVING TROUBLE. WE ARE CONCLUDING THIS TEST.
BM EXIT
LA Q160
QU IF THE SYSTEM OF EQUATIONS   X = 3K
      .  .  .  .  .  .  .  .  Y = 2X = K
      .  .  .  .  .  .  .  .  .  .  Y = 4 = K
   IS CONSISTENT, THEN K =
   A -2
   B 1/2
   C 2/3
   D  3
   E  4
CA B
AD 1 TO V1
WA A
BR Q17
WB C
BR Q17
WB D
BR Q17
WB E
BR Q17
UM TYPE ONLY THE LETTER A,B,C,D, OR E FOLLOWED BY THE ARROW.
UM TYPE ONLY THE LETTER CORRESPONDING TO THE CORRECT
   RESPONSE FOLLOWED BY A LEFT POINTING ARROW.
UM TRY ONCE MORE.
UM YOU SEEM TO BE HAVING TROUBLE. WE ARE CONCLUDING THIS TEST.
BM EXIT
LA Q17
QO Q18 IF V2 EQ 3
GO Q18 IF V2 EQ 0
RS V0
GO Q17A IF V0 LT 2
GO Q17B IF V0 LT 4
GO Q17C IF V0 LT 6
GO Q17D IF V0 LT 8
QU IF THE TWO SOLUTIONS OF X=2 +6X +C= 0 ARE REAL AND
   UNEQUAL THEN WHICH DESCRIBES ALL POSSIBLE VALUES OF C
   A  C = 0
   B  C LESS THAN 9
   C  C GREATER THAN 1
   D  C NOT EQUAL TO 0
   E  C GREATER THAN 9
CA B
AD 1 TO V2
BR Q18
WA A
BR Q18
WB C
BR Q18
WB D
BR Q18
WB E
```

Fig. 3.1

Sample On-Line Printout Sheet for MATH Test

UNIVERSITY OF DENVER AUTOMATED PLACEMENT EXAMINATION

Department of Mathematics

Instructions to the Student

The mathematics test you are about to take is an automated multiple choice examination. You will be seated at a remote terminal of the Burroughs Computer at the University of Denver. The terminal looks much like an ordinary typewriter, but you do not need to know how to type in order to use the teletype. You will, however, need to locate the keys, A, B, C, D and E and one other special key called the "left-pointing arrow." These will be pointed out to you when you take the examination. After each question is typed for you, five responses labeled A, B, C, D and E will be listed for you. After you work the problem (you may use paper and pencil) you are to type the letter corresponding to the correct answer. After you are sure you typed the letter you intended to, you type the left pointing arrow. Should you change your mind before you type the arrow, type the answer you wish and then the arrow. Once the arrow is typed your answer cannot be changed.

In this test the asterisk denotes exponentiation. That is, X*2 is to be interpreted, x squared or x raised to the power 2. Fractions are written as they are in some textbooks, with a slash. Thus, one half is written, 1/2, and x divided by the expression x squared plus 3 would appear, X/(X*2 + 3). The test is not a speed test and you will have ample time to become accustomed to this notation should it seem a little strange to you at first.

The test is designed to take between 45 minutes and one hour. You will not be penalized for using a full hour. If a question seems too difficult, make the most careful guess you can and go on.

These instructions will be repeated for you in summary when you are seated at the terminal. At the conclusion of the test you will receive five messages indicating your success with problems normally covered in Algebra and Trigonometry classes. The test is designed to help you by assessing your individual capabilities and needs. It is our hope that you will find this test an interesting and educational experience.

Fig. 4.1

Instruction Sheet for Computer Examination

```
PLEASE TYPE YOUR FULL NAME AND SCHOOL FOLLOWED BY A LEFT
POINTING ARROW.  IF YOU MAKE A TYPING ERROR JUST RETYPE
THE INCORRECT NAME CORRECTLY BEFORE TYPING THE ARROW.

A- STUDENT-
THIS IS A MULTIPLE CHOICE EXAMINATION. YOU WILL BE GIVEN A
QUESTION WHICH WILL BE FOLLOWED BY FIVE RESPONSES LABELED
A,B,C,D AND E. WHEN YOU DECIDE UPON THE CORRECT RESPONSE,
TYPE THE LETTER CORRESPONDING TO THAT ANSWER. IF YOU ARE
SURE YOU TYPED THE LETTER YOU INTENDED TO TYPE A LEFT POINT-
ING ARROW. IF YOU DID NOT TYPE THE KEY YOU INTENDED TO DO
NOT TYPE THE ARROW. MERELY TYPE THE CORRECT LETTER AND
FOLLOW IT BY THE LEFT POINTING ARROW. IN THIS TEST THE
ASTERISK DENOTES EXPONENTIATION.  THAT IS, X*2 IS TO BE
INTERPRETED, X SQUARED, OR X RAISED TO THE POWER 2.
MAKE SURE THAT YOU HAVE PAPER AND PENCIL.  IF NOT, TYPE
STOP FOLLOWED BY THE ARROW. IF YOU ARE READY TO START TYPE
ONLY THE LEFT POINTING ARROW.


HERE IS A SAMPLE QUESTION. TYPE THE LETTER CORRESPONDING
TO THE CORRECT ANSWER AFTER RESPONSES A,B,C,D AND E ARE
PRINTED. DONT FORGET THE LEFT POINTING ARROW FOLLOWING
YOUR RESPONSE.

  IF X*2 = 9 THEN X =
A  0
B  4.5
C  7
D  PLUS OR MINUS 3
E  11
4.5-
TYPE THE LETTER A,B,C,D OR E CORRESPONDING TO THE CORRECT
SOLUTION FOLLOWED BY A LEFT POINTING ARROW.
D-
TRY ONCE MORE. REMEMBER THE ASTERISK DENOTES EXPONENTIATION.
D-
YOU HAVE THE IDEA. HERE IS YOUR FIRST QUESTION.


THE LOWEST COMMON DENOMINATOR OF THE FRACTIONS
 1/(A*2 +A-20)  AND 1/(A*2-7A+12) IS
A  (A+5)(A-4)(A-3)
B  (A+5)(A+4)(A-4)(A-3)
C  (A+5)(A-5)(A+4)(A-4)
```

Fig. 4.2

Teletype Instruction Output

Date _____

NAME _____

SCHOOL _____

Questionnaire for Participants

1. Have you ever used a teletype computer terminal previous to this experience? . . . . . yes ____ no ____

   If yes, please indicate . . Frequently _____ Occasionally _____ Once or Twice _____

2. How do you feel, in general, about the automation of college placement tests? . . . . . Approve _____ No strong opinion _____ Disapprove _____

3. How do you feel the use of the teletype affected your test performance? . . . . . Adversely _____ No measurable effects _____ Beneficially _____

4. Would you like to take the test again? . . . yes ____ no ____

5. Do you plan to go to college? . . . . . yes ____ no ____

6. If you answered "yes" to #5, what do you think your college major will be? . . . . . _____

7. Do you plan to take Calculus in college? . . . . yes ____ no ____

8. If a math placement test were not given at the college of your choice, would you elect to take any pre-Calculus mathematics? yes ____ no ____

9. Do you type? . . . . . yes ____ no ____

10. Additional comments _____

Fig. 4.3

Questionnaire for Participants

```
IN RIGHT TRIANGLE RST THE HYPOTENUSE RS IS  15 AND THE
ANGLE TRS = ARCTAN 3/4. FIND THE AREA OF TRIANGLE RST.
A 12
B 18
C 27
D 36
E 54
E-


IF LOG 20 = P AND LOG 2 = Q THEN LOG 40 =
A P9
B P+Q
C 2(P+Q)
D 10(P+Q)
E NONE OF THESE
B-


WHAT ARE ALL THE REAL NUMBERS X FOR WHICH THE PRODUCT
(X+3)(X-1) IS GREATER THAN ZERO.
A  X GREATER THAN 1
B  X LESS THAN 3
C  X GREATER THAN -1 AND LESS THAN 3
D  X GREATER THAN -3 AND LESS THAN 1
E  X GREATER THAN 1 OR LESS THAN -3
E-


FIVE SCORES WERE RECORDED FOR YOU BASED UPON YOUR RESPONSES
TO THESE QUESTIONS.  IF THIS HAD BEEN AN ACTUAL PLACEMENT
EXAMINATION THIS MESSAGE WOULD HAVE GIVEN YOU SOME PERSONAL
SUGGESTIONS FOR PRE CALCULUS STUDY IF SUCH HAD BEEN
INDICATED.  BECAUSE THIS TEST IS STILL UNDER DEVELOPMENT
YOUR INSTRUCTOR IS BEING ASKED TO CONSIDER THESE MESSAGES
BEFORE THEY ARE REPORTED TO YOU.  AT THE CONCLUSION OF THIS
MESSAGE PLEASE WAIT A FEW SECONDS AND THEN ROLL THE PAPER
AHEAD A FEW TURNS, TEAR OFF AND HAND TO THE FACULTY MEMBER
OR ASSISTANT MONITORING THIS TEST.  YOUR WRITTEN COMMENTS
TO THE MONITOR REGARDING THE AUTOMATED TEST ARE REQUESTED
AND WILL BE APPRECIATED BY THE UNIVERSITY STAFF.  GOODBYE.

YOU ARE LEAVING THIS COURSE.
        V0= 4
        V1= 3
        V2= 3
        V3= 3
        V4= 3
        V5= 3
WOULD YOU LIKE TO SELECT ANOTHER COURSE.
```

Fig. 4.4

Teletype Messages During Test Development

'LEARN-IN':
EXPERIENCES WITH A METHOD FOR GROUP STUDY

J.S. de Vries
W. van der Klooster

1. ORIGIN

Business undertakings are experiencing an increasing need to extend, or keep up-to-date, the proficiency of their staff by means of full-time day courses which can often extend over a number of weeks.

Participation in one of these courses means a return to the schoolroom atmosphere for employees who may already hold important positions within the firm and, in some cases, having done so for many years. This return is, in some ways, a come-down, as industrial training is often, as one might expect, copied from the conventional school situation where discipline and teaching methods are adapted to youthful pupils who still have to learn how to study and who still require a great deal of guidance and supervision.

Many of the participants at industrial courses, however, have had a secondary or higher education and may be considered as having knowledge of what study involves. What is more, they are confronted every day, in exercising their duties, with situations which demand initiative and independent action and for which the ability to co-operate is an indispensable condition.

To bring such industrial officials back in the tutelage of a school-like situation obviously would mean a serious failure to recognize skills they have acquired and which they can utilise to such very great advantage in a new learning process.

2. BASIC IDEA

After this diagnosis, it is obvious that the remedy must consist of creating an atmosphere of study which shows the same characteristic features as the modern industrial situation. A teacher who prescribes and supervises the learning process down to the finest details has no place in this system. What does fit in is a method of study whereby those taking the course themselves organize the job of studying in order to attain the goal set by the teacher and, at the same time, assessing the correct execution primarily by themselves. The social situation has to be such that students have ample opportunity for individual enterprise and co-operation. In other words, a situation which, in school education and also in courses, is sometimes by way of exception created for certain forms of group work. This has led to the organizing of a system of permanent group study, the essence of which is the following:

- the course members study in small groups, assisting, guiding and checking one another;
- every course member also accepts a share of the responsibility for the performance of the others in this group;
- the entire learning process takes place during working hours;
- the 'teacher' guides, but never leaves the background.

The method developed, which will be referred to here as a 'learn-in' has, meanwhile, been tried out in a number of courses. Participants learn from and with one another in an atmosphere which, though free from constraint, is not unsupervised.

3. THE PROCEDURE AT A 'LEARN-IN'

The subject matter to be studied is first recorded in writing – in a textbook or by assembling chapters from books, notes, summaries, diagrams, etc., – and divided so as to form a number of rounded-off study tasks. It is important that each part should be provided with good questions on the matter to be studied and assignments in its application.

The teacher tells participants about the study task as a whole and gives them, before the start of each new part, an idea of the objectives of their study and of the requirements their knowledge and ability will finally be expected to meet.

On each such occasion he also makes an estimate of the time they will need. This estimate is based on his experience. For each study task the learning process comprises the following five phases:

3.1 *Orientation and organization*

From a quick perusal of the material to be studied, the participants from an idea of the aim, level and scope of the task before them. In common consultation, they draw up the plan of study and then divide the whole task into sub-tasks and prepare a time schedule.

3.2 *Private study*

In this phase, the participants study the material independently but are at liberty to consult one another incidentally. This phase is characterized by silence and concentration on individual study.

3.3 *Group discussion*

In their groups, the participants hear one another's 'lessons', point out to each other link-ups they have found in the material and the relation with other fields of knowledge or practice, so that they can correct and supplement each other. In the course of such discussions, the participants learn that their reading thoroughness and interpretation are often lacking in quality.

Consultation with other groups may prove necessary during this phase. If this does not solve the difficulties, the participants may approach the teacher for hints or

supplementary study material. In most cases it will be found that the difficulties are due to imperfections in the material. In this phase, there is a tendency for participants to become emotional, resulting sometimes in hot discussions.

3.4 *Practice in application*

As stated, a study task includes a number of questions and assignments. By answering the questions independently, the participants can test their ready knowledge and understanding. The assignments enable them to show how they can apply their newly acquired knowledge. If they get stuck, mutual consultation is, of course, permitted. Everyone should, however, be able eventually to supply his own answers and solutions.

3.5 *Evaluation*

The answers are compared in a group discussion and alternatives are discussed; participants pick out each other's errors. A good solution must now give way to one which is better.

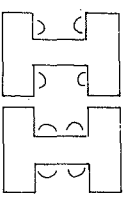
Finally, each group has to produce a single set of answers and solutions which is submitted to the teacher for assessment. Every study task is terminated by a test to be taken by every participant independently.

The moment of testing is decided in common consultation, bearing in mind the desirability of spreading the study tasks evenly over the pre-arranged duration of the course. Those who would like to devote some extra time to the subject are allowed at least one extra evening for this purpose.

4. THE SEATING ARRANGEMENTS AT 'LEARN-INS'

The participants sit together in groups of 3 or 4 in a single classroom, with each group in its own 'study corner'. The furniture in each such corner is arranged so that participants can either be separated for private study or sit together for group activities.

The following figure shows the two arrangements:



Seating arrangement for private study at 'learn-ins'

Seating arrangement for group study at 'learn-ins'

In order to change over from one arrangement to the other, participants need only turn their chairs by an angle of 90°.

For a 'learn-in' attended by 4-5 groups of participants (12-20 persons) a room of larger dimensions than that of an ordinary classroom is desirable. Another point is that the acoustic requirements for a 'learn-in' are exactly the opposite of those for a schoolroom where the teacher has to be clearly audible from the farthest corner.

Nevertheless, it has been found that, with a little goodwill, 5 groups of 4 can be accommodated even in a classroom of ordinary size and that, except during some climaxes in discussions, the groups suffer little inconvenience in the way of noise from each other.

5. EXPERIENCES WITH 'LEARN-INS'

The 'learn-in' method has meanwhile been used a number of times for a day course lasting six weeks *. This course was previously conducted in the traditional way, by setting homework regularly and by using a textbook compiled in the actual course of teaching.

In addition to imparting the necessary knowledge, concepts and understanding, the course is primarily aimed at learning solution methods for computer programming. The latter has a pronounced 'problem solving' character and necessitates many exercises. In the course of a 'learn-in', it becomes clearly evident where the imperfections of the study material lie. This is because the teacher receives the problems which participants have neither been able to solve within their own group nor in consultation with the other groups. Consequently, the teacher at the first 'learn-in' had his hands full in improving the material in accordance with indications from the participants. The textbook was rewritten after that first 'learn-in' and it was found that the next was thereby greatly improved.

A condition for the success of the 'learn-in' is that every participant accepts some responsibility for the performance of his group mates. Such acceptance occurs spontaneously. Experience to date shows that participants consider it a point of honour to ensure nobody in their group falls behind. A weaker member of a group is coached -- often with striking success -- by one or more other members. Precisely because it is possible to make contact so often and so easily with fellow students – contact which is essential for various phases – there is little chance of mistaken ideas being formed or of gaps in knowledge and ability coming into existence.

It has also been found that the assignments which conclude the study task have to be 'meaty', then making co-operation necessary.

The considerable consultation involved in working out these assignments consequently tends to blur the difference between phases 4 and 5; but the participants themselves remain strongly motivated to find final solutions. Active personal assimilation by each participant is thus ensured. No distribution of tasks results from this.

When, however, in the final stage of a study task, fair copies have to be drawn up for submission to the teacher's assessment, then this part of the study task is efficiently distributed among the members of the group. Because of the small number of groups, the teacher can devote much attention to these group answers and solutions. If necessary, he discusses the results with the group concerned, trying to get the participants themselves to discover the causes of weak performances and the means of putting them right.

At 'learn-ins', the entire learning process should take place in the 'study corner' during the assigned working hours. This restriction with regard to time and place is found to lead to regular and intensive study.

* One of the courses of the ISA Dept. (N.V. Philips), namely that which prepares students for the programming examination of the 'Stichting Studiecentrum voor Administratieve Automatisering'.

Traditional courses used to necessitate at least two hours' homework daily; good and average participants in 'learn-ins' have hardly needed to do any homework.

6. CONDITIONS

It will be clear from what has been said above that the 'learn-in' can certainly not be used in every training situation. The most important conditions for successful application are:

– The course must be full-time or spread over not less than half day periods, so that the entire or almost entire learning process can take place during the 'learn-in'.

– Participants must be sufficiently grown-up and possess sufficient experience of study and previous training to be able to accept personal responsibility for active and efficient co-operation in a group.

– The aims of the study must be established as clearly as possible in operationally formulated study requirements.

– Study material and study hints must be available to enable the participant, independently or with the aid of colleagues, to attain the proposed objectives.

– The room and furniture must be suitable for an arrangement which accentuates the group character of the 'learn-in' and also makes it possible for the participant to engage in individual activity.

– The teacher must naturally have an expert's mastery of the material and also show that he is prepared to remain in the background and knows how to do so. He must be able to play the far from easy part of someone who can always be appealed to for assistance or coaching – a discrete operating manager of the learning process.

7. WIDER APPLICABILITY

It seems obvious, in view of what experience has shown, that the 'learn-in' will be employed on a wider scale for courses in industry. Courses, other than industrial, will also undoubtedly be able to benefit from this method. We are thinking, for example, of university courses where complete freedom of study and the limited opportunity to evaluate short-term results of study are satisfactory to neither the authorities nor the students. Used in this context, the 'learn-in' need not follow the five-phase pattern rigidly.

We imagine that the 'learn-in' is also basically suitable for material which, from the instructional point of view, is not highly organized, e.g., selected articles from professional literature. We are planning to experiment along these lines, observing the necessary caution. Possible motives for experiments of this sort are:

– There is scarcely any specially compiled study material available for courses intended to keep up-to-date with recent developments.

– It is most important that people following courses should learn to find their way about professional literature, so that this might be made one of the objectives of such courses.

A number of successive 'learn-ins' would then eventually make it possible, on the basis of practice, to arrive at a draft for a definitive textbook. For people following the course at later dates, the course could be expanded by studying recent or even the latest developments drawn from professional literature.

Our aim with the 'learn-in' has been to create a study situation in which adults can study effectively in a manner which they find acceptable.

Examination results * and the favourable verdict of participants as revealed by interviews and surveys show that the 'learn-in' definitely attains that objective.

* 92% of those who took part in the first 'learn-in' passed the national examination in programming – an extremely high percentage!

APPLICATIONS OF FORMAL LANGUAGE THEORY TO PROGRAMMING LANGUAGES: (A COURSE DESCRIPTION WITH OBSERVATIONS)

Robert M. Aiken

1. PREREQUISITES AND LEVEL OF STUDENTS

This course is currently being offered to graduate students enrolled in a Master's Program in Computer Science at The University of Tennessee, Knoxville, Tennessee (USA). These students have already taken beginning and intermediate courses in computing that concentrate on basic programming and hardware principles. In the 'software' courses, which are prerequisites for this advanced course, the students are required to write programs in several computer languages including ALC (the machine language for the IBM 360 series), FORTRAN, ALGOL, and PL/I. In the 'hardware' courses the students have been exposed to topics such as Boolean Algebra, rudimentary Operating Systems Principles and basic Switching Theory. Though this latter sequence of courses is not required as a prerequisite to this advanced course in programming theory, it is helpful to the student if he has taken it. Other prerequisites for this course include the equivalent of two years of Calculus and at least one course in Probability and Statistics.

Though it is not a formal requirement it has been observed that practical programming experience is an aid to students taking this course. The students who have derived the greatest benefits from this material are those who have held programming jobs in either business or university environments. By combining this practical background with knowledge from their previous courses these students have built a foundation which they use to integrate the theory learned in this advanced course with their previous experience. Consequently, these students find it easier to understand how programming theory has developed and why it has influenced programming languages. Students with a better applied background also tend to more easily discern how programming theory can be extended to apply to other areas such as the description and generation of natural languages.

2. PHILOSOPHY AND ORGANIZATION OF THE COURSE

Since The University of Tennessee currently awards only a Master's Degree in Computer Science, this advanced course is one of a few required courses in the program. For the benefit of all the students, both those who are terminating at the Master's level and those who will be pursuing Ph. D.'s elsewhere, an attempt is made to build a theoretical structure which a student can use as one means of analyzing the strengths and weaknesses of various programming languages. The student can better appreciate some of the differences in form (and content) in the various programming languages by learning the theory necessary to explicitly describe these languages. Due to the availability of literature discussing the merits of the ALGOL and FORTRAN languages and because of the availability of these compilers at the University of Tennessee Computing Center, these two languages, particularly ALGOL, are studied most extensively in this course. It was found that the students develop a better understanding of not only the problems associated with constructing programming languages, but also the problems of implementing them by having an available programming language serve as the mechanism for introducing examples of the programming theory. With this background a student is better prepared to read and understand the current literature in the area of programming languages plus, in some cases, make contributions of his own.

This course is organized into twenty 70 minute sessions (two per week for ten weeks, on a quarter system) with an approximate distribution of the time as follows:

First Part

1. Two sessions – Intuitive and Formal Descriptions of a Language
2. One session –
   a. Development of the Four Types of Grammars
   b. Properties of Grammars
3. Four sessions – Context-Free Grammars
   a. Derivation Trees
   b. Chomsky Normal Form
   c. Greibach Normal Form
   d. Self-embedding Properties
   e. Special Types of Context-Free Grammars
   f. Parsing

Second Part

4. Two sessions – Use of Context-Free Grammars to Describe Programming Languages
5. Three sessions – Acceptors, Transducers, Processors
6. One session – 75 minute Mid-term Exam

Third Part

7. Four sessions – Discussions of outside readings that point out the influence of programming theory in the design and implementation of ALGOL 60 and EULER
8. Three sessions – Oral Summary and discussion of Research Papers
9. Individual Oral Finals

Though there are some deviations from this general outline, the above breakdown summarizes how the author allocates the time for various subject areas in this course.

3. MATERIAL COVERED

Much of the material used in the first part of this course is contained in books by Ginsburg (1966) and Hopcroft and Ullman (1969), plus some prepublication notes of a book to be published soon by Professor G.K. Krulee (1969). In particular, Chapters 1, 2 and 4 of Hopcroft and Ullman's book contain pertinent material as do the first two chapters in Ginsburg.

During the first part of the course students are informally introduced to some of the concepts of formal language theory. Where possible, programming and natural language examples are used as motivating devices in introducing the formal definitions of such concepts as languages, grammars, alphabets, procedures, and algorithms. In particular, a formal definition of grammars is developed whereby a grammar is described in terms of a 4-tuple consisting of an alphabet composed of terminals and non-terminals, a set of production rules, and a starting symbol. Motivating examples for a representation of a grammar are drawn from parsing diagrams of English sentences and ALGOL strings.

Having received an overview of grammars and language theory, students are introduced to the four types of grammars. Definitions are given for a type 0 grammar (an unrestricted grammar), type 1 (context-sensitive grammar), type 2 (context-free grammar), and type 3 (finite state grammar). These definitions are presented both formally and informally in such a way as to show how the grammar is effected by increasing the number of restrictions. Thus, each restriction placed on a grammar is shown to make that grammar a sub-set of the grammar from which it was derived. For example, every context-free grammar is a context-sensitive grammar, and every context-sensitive grammar is an unrestricted grammar.

Special properties of grammars are also discussed including problems encountered in working with the empty sentence, recursiveness of context-sensitive grammars, and the device of representing grammars through derivation trees. This last property is examined in some detail with the student being asked to extend this concept to context-sensitive and unrestricted grammars. Students are also required to work a great number of problems, since it is the author's belief that it is through working the problems that the student has the best opportunity to understand the material. Most of the problems used are drawn from Chapter 2 of Hopcroft and Ullman (1969).

The next set of lectures present material related to context-free grammars. In particular, various procedures are developed through which one can arrive at a Chomsky Normal Form Grammar (Binary Standard Form Grammar) and using this as a basis derive a Greibach Normal Form Grammar, both of these being special cases of context-free grammars. Other special cases of context-free grammars are examined plus additional topics are introduced such as the solvability of finiteness, the 'uvwxy' Theorem, and self-embedding properties. All of this material is generously supplemented by problems which the students are required to work. Most of these problems are drawn from Chapter 4 of Hopcroft and Ullman (1969).

In conjuction with this material the students are required to read two basic outside papers, one by Chomsky (1959) and one by Landweber (1964). It is hoped that these papers serve as a reference point for students in terms of their being exposed to some of the initial work published in this area. Further, these papers serve to bring into focus several relationships between this theory as applied to programming languages and the extension of this theory as it has been applied to natural language concepts. In particular, the limitations of this theory in relation to both programming and natural language representation is discussed.

In the second part of the course, in which the author attempts in part to demonstrate how and why programming theory is important in the development and implementation of programming languages, the students are required to do considerable outside reading. Some of the articles discussed in class (in the order listed) include: Samelson and Bauer (1960), Hamblin (1962), Floyd (1961, 1962, 1963, 1964), and Randell and Russell (1964).
These articles were chosen because of their contributions to the development of programming languages, particularly regarding their approaches for describing and implementing various syntactic expressions. In addition, some of the pre-publication notes of Krulee's on recognizers, acceptors, and transducers are introduced here. In a discussion of grammars the idea of level of complexity of various types of grammars was introduced. It was shown that grammars can be classified by examining the complexity of their replacement (or production) rules. In similar fashion, processors can be classified and, in fact, a major portion of this set of lectures discusses the connection between the complexity of a processor and that of the grammar that is built into its functioning. In addition to the pre-publication notes of Krulee (1969) and some notes prepared by this author, Aiken (1970), a paper by Evey (1963) is cited quite frequently in the development of this material.

The basic points made in this series of lectures concern the properties of finite state processors and the need for the addition of pushdown stores (memory devices). In particular, it is shown that a finite state processor can process a finite state language; a finite state processor with one pushdown store is needed to process a context-free language and a finite state processor with two pushdown stores is needed to process context-sensitive and unrestricted languages. Examples of each of these types of systems are discussed and the resulting complexities are introduced in a step-by-step fashion. In addition to the author presenting a series of examples, the student is required to work a number of problems related to the development of each of these types of processors. In addition, the topics of state diagrams and state tables are introduced in conjunction with finite state processors, and tree diagrams are introduced in the discussion of both finite state processors and finite state processors with one pushdown store.

The third and final part of this course primarily consists of outside reading that attempts to acquaint the students with how programming theory was used in the design and implementation of ALGOL 60 and EULER. The students read papers by Knuth and Merner (1961), Naur and Woodger (1963), Wirth and Weber (1966, a,b), and Knuth (1967). Class discussion involves not only a review of some of these papers but also an attempt to weld these topics into a unified structure. This includes analyzing the contributions of programming theory to the development of ALGOL 60, analyzing problems that occurred as a consequence of allowing non-phrase structure rules in ALGOL 60, and examining how additional rules could be developed to generate a more complete and satisfactory

programming language. This last topic is motivated and examined through a comprehensive discussion of the EULER system (Wirth and Weber (1966 a,b)).

In particular, the EULER papers provide a major portion of the material needed for a summary of the topics presented in this course. EULER is a language based on a method for defining programming languages in which a rigorous relationship is established between structure and meaning. The structure of the language is defined by a phrase structure syntax, and the meaning is defined in terms of the effects which the execution of a sequence of interpretation rules exert upon a fixed set of variables. Thus, there exists a one-to-one correspondence between syntactic rules and interpretation rules. In the execution sequence interpretation rules are determined by the sequence of corresponding syntactic reductions which constitute a parse. Thus, this system embodies the theoretical aspects which were discussed at the beginning of the course and applies them to the ALGOL language in a manner which extends and generalizes ALGOL. Specifically, various defects in ALGOL (defects in terms of properties in ALGOL which cannot be described by a phrase structure language) are remedied in the EULER system through the one-to-one correspondence between syntactic rules and interpretation rules. Thus, in EULER, unlike ALGOL, various concepts like block structure, procedures, parameters, etc, can be defined adequately and precisely.

In addition, the various features necessary to implement the EULER language are also discussed in the Wirth and Weber articles. Further, the problems encountered in implementing the ALGOL language, due in part to the problems of ambiguity for features which cannot be defined by a phrase structure language, are outlined in these papers and the corresponding solution is given for the EULER system. Thus, programming theory is shown to be valuable as a means of constructing an unambiguous programming language, EULER, based on a phrase structure syntax. In addition, the description of the implementation of this language is quite straightforward. The students seem to be in a position to understand the EULER papers and appreciate the facilities which EULER has incorporated. Though the EULER system is not available at The University of Tennessee, the students are motivated by the fact that it is a usable programming language and one that has some very interesting and useful features that can be derived from properties predicted by the theory.

Thus, in this course a series of lectures are given in which the author attempts to do the following:

1. Acquaint the student with concepts and developments in programming theory. The development of the formal ideas in these areas are motivated by the use of natural and programming language examples. The relationships in these areas are stressed so that the students can combine the theory with practical examples where at all possible.
2. In conjunction with the theory in the first part of the course, a series of lectures are given relating the grammars and their corresponding languages to processors. The obvious relationship between processors as computers and the languages as programming languages are made but the differences between these corresponding relationships are also emphasized.
3. The concluding lectures in the course are primarily based on the EULER papers which bring into focus how programming theory can aid the development of a programming language. Not only is the EULER language unambiguous, but through its formal definition there exists a one-to-one correspondence between syntactic rules and interpretation rules. Since the EULER system is an extension of ALGOL it is easy for the students to see the advantages of an unambiguous language both in terms of program interpretation and in terms of ease of implementation. These points are brought out in the EULER papers and are further strengthened by examples brought in by the author.

4. PROBLEMS, PAPERS, EXAMS

Problems were assigned about once a week, primarily as reinforcement for material covered in class. Both Hopcroft and Ullman (1969) and Ginsburg (1966) have some excellent problems; however, Hopcroft and Ullman (1969) have a more complete set of problems of varying degrees of difficulty. Neither Hopcroft and Ullman nor Ginsberg has worked solutions or an answer book. Since about half of the outside reading assignments paralleled material from the text, most of the problems related to this material were drawn from the exercises in the text. In addition, in this initial portion of the course, the student was usually required to work one problem and/or work through one example related to each of the outside papers. (See Appendix A for some sample problems).

In some of the second part and most of the third part of the course the students were assigned a problem, an example, and/or a proof to work out based on material in each of the papers they read. Also, in some cases, the students were required to make up and solve a problem based on the material currently being covered.

Tests consisted of a written mid-term and an oral final. The mid-term was divided into two parts with one-third being 'closed book' (short answer questions) and two-thirds being 'open book' (problems). The final was an individual oral final. Three-fourths of the oral final consisted of short answer and definition questions which were the same for all students. One-fourth of the oral consisted of a different general question for each student in which he was asked to extend or relate to another area a research paper he had presented in class.

The final requirement for this course was a research paper. The reasons for assigning this paper were twofold: (1) to help acquaint the student with the literature and (2) to encourage the student to start working on a thesis. Since the Computer Science Program at The University of Tennessee requires a thesis, the student has an opportunity through this course to do some research on a thesis topic while fulfilling a course requirement. Since this is the first course in a required two course sequence taught by the author, the paper is not due until the latter part of the second quarter. The requirement related to this paper for the first quarter is that the student turn in a critically annotated bibliography that evidences thorough research on his topic. In addition to the written report, the student is required to present the paper orally before the class and interested faculty. One reason for having the student give an oral presentation is that by presenting this paper before

a critical, yet receptive audience, the student gains experience which will better prepare him to present papers at technical meetings and symposia. Learning to present material orally is sometimes very difficult for these students, yet most, if not all of them, have said they think the effort is worth the experience. (See Appendix B for summaries of some of the oral reports).

5. CONCLUSIONS

This description outlines the course 'Applications of Formal Language Theory to Programming Languages; A Course Description With Observations'. It also contains details of many aspects of the course (including certain of the author's prejudices) that might be useful to others interested in offering a similar course. Though it has been designed for a particular set of circumstances, it is hoped that the substance of this presentation can be used by other individuals in various programs and universities.

Currently, the aspect that needs the most development is the third part of the course. Though the outside reading is quite interesting and informative, the material is too diverse to unify satisfactorily. In addition, it is also difficult at times to relate the material in this last part to the first two parts of the course. Thus, there seems to be a definite need for an approach that would link this material to programming theory in such a way as to clearly show the relationship existing between programming syntax, design of programming languages, the implementation of programming languages, and compiler design. Though discussing the EULER papers helps to establish these relationships. it would be beneficial if more material were available that presented these topics in some unified manner.

REFERENCES

This bibliography contains those articles referenced in the paper plus others which have been of use in preparing material for this course. These works are not meant to form an exclusive and exhaustive set, and in fact, are simply a representative cross section of papers covering the topics presented in the course.

Aho, A.V., and J.D. Ullman, 1968, Math. Systems Theory. 2, 97-125.

Aiken, R.M., 1969, SIGCSE Bull. 1, 7-11.

Aiken, R.M., Notes on the Relationships of Grammars to Acceptors. Unpublished Lecture Notes, 1970.

Chomsky, N., 1956, IRE Transactions. KT-2, 113-124.

Chomsky, N., 1959, Inf. and Control. 2, 137-167.

Chomsky, N., 1963, Formal Properties of Grammars. In: Bush, R.R., and E. Galanter, and R.D. Luce, eds., Handbook of Mathematical Psychology, vol. 2 (Wiley, New York) pp. 323-418.

Evey, J., The Theory and Applications of Pushdown Store Machines. Doctoral Dissertation. Report NSF-10, Harvard University, 1963.

Feldman, J., and D. Gries, 1968, CACM.,11, 77-113.

Floyd, R.W., 1961, JACM. 8, 579-584.

Floyd, R.W., 1962a, CACM. 5, 483-484.

Floyd, R.W., 1962b, CACM. 5, 526-534.

Floyd, R.W., 1963, JACM. 10, 316-332.

Floyd, R.W., 1964a, CACM. 7, 62-67.

Floyd, R.W., 1964b, IEEE Trans. on Elec. Comp. EC-13, 346-353.

Ginsburg, S., 1966, The Mathematical Theory of Context-Free Languages (Mc Graw-Hill, Inc., New York).Greibach, S., 1964, Cacm. 7, 499-504.

Hamblin, C.L., 1962, Computer J. 5, 210-213.

Higman, B., 1967, A Comparative Study of Programming Languages (American Elsevier Publishing Co., Inc., New York). Elsevier Publishing Co., Inc., New York). Relation to Automata (Addison-Wesley Co., Reading, Massachusetts).

Irons, E.T., 1961, CACM. 4, 51-55.

Knuth, D.E., and J.N. Merner, 1961, Cacm. 4, 268-272.

Knuth, D.E., 1965, Inf. and Control. 8, 607-639.

Knuth, D.E., 1967, CACM. 10, 611-618.

Knuth, D.E., 1968a, Fundamental Algorithms, vol. 1 (Addison-Wesley Co., Reading, Massachusetts).

Knuth, D.E., 1968b, Math. Systems Theory. 2, 127-145.

Krulee, G.K., Abstract Processing Systems. Unpublished Manuscript, 1969.

Landweber, P.S., 1964, IEEE Trans. on Elec. Comp. EC-13, 354-362.

Lewis, P.M., and R.E. Stearns, 1966, Syntax Directed Transduction. In: IEEE Conference Record of Seventh Annual Symposium on Switching and Automata Theory, pp. 31-35.

Naur, P., and M. Woodger, eds., Cacm. 6, 1-20.

Randell, B., and L.J. Russell, 1964a, Jacm. 11, 159-167.

Randell, B., and L.J. Russell, 1964b, ALGOL 60 Implementation (Academic Press, London).

Samelson, K., and F.L. Bauer, 1960, Cacm. 3, 76-83.

Wegner, P., 1968, Programming Languages, Information Structures and Machine Organization (Mc Graw-Hill, Inc., New York).

Wirth, N., and H. Weber, 1966a, Cacm. 9, 13-25.

Wirth, N., and H. Weber, 1966b, Cacm. 9, 89-99.

APPENDIX A

SAMPLE PROBLEMS
(with author's assessment of difficulty)

1. Write the production rules for a type 2 grammar that will generate the following language:
   $L(G) = \{\#a^n b^n \# / n > 1\}$ *

2. The dictionary of the English language contains only a finite number of words. One may argue that there is probably a finite number of rules of grammar for the English language. Explain why you think the number of sentences in the English language is finite or infinite. *

3. Write a context-free grammar for generating arithmetic statements in ALGOL 60 (Hint: Use the Backus Naur syntax in the Revised Report on ALGOL 60 as a guide). **

4. Consider the language over the input alphabet containing a's and b's such that the number of a's, n (a), $\geq$ number of b's, n (b), plus 2.
   a. Write a partially specified acceptor for this language.
   b. Show by some constructive procedure that you can always design a single pushdown store acceptor for any language of the form:
      $L(G) = \{\# x \# / n(a) \geq n(b) + K$, for K any positive integer$\}$ **

5. Write the production rules for a type 1 grammar that will generate the language:
   $L(G) = \{\#a^{n^2} \# / n \geq 1\}$ ***

6. Write the production rules for a type 1 grammar that will generate strings of the form:
   $L(G) = \{\# x x \# /$ a string x that repeats itself$\}$***

* easy
** moderately difficult
***difficult

APPENDIX B

SUMMARIES OF RESEARCH PAPERS GIVEN BY STUDENTS

'Game Playing Programs: An Application of Pattern Recognition to Learning'

An attempt is made to show why pattern recognition more closely simulates intelligent thought than other techniques – e.g. heuristic analysis. Secondly, the main body of the paper gives a concrete example of the application of pattern recognition to learning using a creative game. This study includes the author's own ideas concerning the validity and extensions of the techniques used.

'Programming Languages for Automata'

This paper, defines a programming language similar to the one developed by Knuth and Bigelow (JACM. October, 1967) and adapted for a simpler type machine. In particular, two way push down automata are discussed and a programming language is described for working with this type of automata.

'A Formal Procedure for Deriving Descriptive Grammars from Formal Descriptions of Games'

This paper considers the methods and techniques in the field of grammar and language theory which have been developed for the description of formal systems. In addition, this paper attempts to extend these techniques to a formal procedure for generating a useful grammar from a formal description of a game or game-theoretic situation. This derived grammar can then be utilized by an interactive acceptor/generator to play the game. This procedure is demonstrated using a simple example – the Eleven Match game.

'A Sequential-Circuit Compiler'

This paper examines some of the problems associated with designing a processor which produces a flow table that describes a sequential circuit, according to the specifications given via a design language. A description of the necessary and desirable characteristics of the language, together with an example of the language are given. Once the specifications are determined, the other problems discussed are: a. tree construction, showing the possible routes through the states; b. the pruning of equivalent states from the tree; c. state assignments; and d. cost minimization, resulting in a flow table for the system.

TEACHING HUMANISTS TO USE COMPUTERS WITH SNAP

Michael P. Barnett

1. INTRODUCTION

Several hundred graduate students, trained previously in the liberal arts and humanities, have been taught the elements of programming, file design, and mechanized text processing during the past three years in a course given by the author at the School of Library Service of Columbia University, using the Columbia University Computing Center for practical experience. Related courses have been given by the author at The H.W. Wilson Company for management and editorial staff. As the central element of these courses, the participants are taught to program in the SNAP (Stylized Natural Procedural) language. This is a kind of 'basic English' designed to express text processing operations in a way that is relatively 'natural' for editors, librarians, educators, and other users who deal primarily with verbal material, rather than computations or accounting (1) - (6). A prototype processor, that executes SNAP procedures was implemented in FORTRAN IV. The IBM 360 and RCA Spectra 70 versions are available from P.I.D. and P.A.L. (7). They are in operation in several University computer installations. The processor also runs on the computers of several other manufacturers. A new processor, that incorporates a powerful table driven syntactic analyzer and string synthesizer is nearing completion. It will permit easy adaptation to stylizations of other tongues, merely by changing the syntax that is used by the portion of the processor that translates a SNAP procedure into a numerical representation (SNAPIC) that an interpreter then executes.

A college text that explains programming in SNAP has been published recently (4). It serves also as an introduction to mechanized text processing. The language, and the examples and exercises that the author uses to teach it enable quite large classes of students who would shy away from more symbolic or arithmetically oriented languages to start writing, running, and correcting SNAP procedures on the computer, very quickly, without burden to the teaching staff. The students gain an understanding of programming concepts, and confidence in their ability to cope with the computer, that enables and motivates many of them to learn other languages, including IBM Job Control Language, that they would not have attempted at first.

The author believes that programming and related matters should be taught to students of the humanities and the arts for several reasons.

(1) Computers are being used to a rapidly increasing extent in publishing, librarianship, marketing, and education, and other non-arithmetical activities of public and private organizations. The advantages of the applications expert understanding computing principles and being able to program have been well established in other areas of computer application. Many of the applications experts in the areas just mentioned are trained in the humanities and the liberal arts, and in professions that are extensions of these.

(2) Teaching large numbers of students of the humanities and liberal arts to program may create a new source of personnel for those aspects of systems programming and systems design that do not require mathematical knowledge.

(3) Despite the increase in Business School training, and its technical content, 'command decisions' very often will continue to be made by administrators and legislators whose formal education was entirely humanistic. Since decisions that require insight into the realities of computing will be of increasing importance in the future, there are potential benefits to increasing the level of understanding of computers by students who are being trained in the humanities now.

(4) Students who may undertake research work in the liberal arts and humanities should be trained to use the computer since it may be a useful tool in their research.

(5) The computer and programming can be considered part of contemporary 'culture' (in the anthropological sense) and as such merit a place in general education. Learning to program moreover teaches a mode of thought that may well lead imaginative students to develop new ideas in other fields (e.g. stylistics, linguistics, subject classification and rigorous definition schemes), even though these do not require use of the computer per se.

These points are discussed further in references (4) and (6).

The essential style of the SNAP language is described in Section 2, by reference to a simple SNAP procedure. The order in which the elements of the SNAP language and their applications are introduced in the author's course and text is summarized in Section 3, and two major phases of this development are discussed in more detail in Sections 4 and 5. Some more advanced topics are considered briefly in Section 6, leading to an account in Section 7 of the capabilities of the new SNAP processor to perform syntactic analysis and table driven translation. These present some interesting possibilities including counterparts of SNAP for other natural languages.

2. THE SNAP LANGUAGE

The general style of SNAP is illustrated by a procedure to print the incremental ballad 'The Twelve Days of Christmas' which has parallels in other languages (8). The fixed words of the SNAP language are underlined in the procedure that follows, to facilitate explanation, not for input to the computer.

```
CALL "DAY OF CHRISTMAS/ MY TRUE LOVE SENT
TO ME" THE THEME.
CALL "A PARTRIDGE IN A PEAR TREE//, TWO
TURTLE DOVES AND,
THREE FRENCH HENS, FOUR COLLY BIRDS, FIVE
GOLD RINGS, SIX
GEESE A-LAYING, SEVEN SWANS A-SWIMMING,
EIGHT MAIDS A-MILKING,
NINE DRUMMERS DRUMMING, TEN PIPERS PIPING,
ELEVEN LADIES
DANCING, TWELVE LORDS A-LEAPING,"THE GIFT
LIST.
CALL "FIRST, SECOND, THIRD, FOURTH, FIFTH,
SIXTH, SEVENTH,
EIGHTH, NINTH, TENTH, ELEVENTH, TWELTH" THE
ORDINAL LIST.
SET N TO 1. (VERSE START) PRINT "THE" THEN THE
N-TH ORDINAL
THEN THE THEME. SET M TO N. (LINE ACTION)
PRINT THE N-TH GIFT.
IF M IS GREATER THAN 1 DECREASE M BY 1, AND
REPEAT FROM THE
LINE ACTION, OTHERWISE CONTINUE AS FOLLOWS.
IF N IS LESS THAN
12 INCREASE N BY 1, AND REPEAT FROM THE
VERSE START, OTHERWISE
TERMINATE. EXECUTE.
```

A CALL statement in SNAP gives a name to either a string, or to a list of strings. The first two statements in the procedure just displayed illustrate these two possibilities. The general form of the first kind of CALL statement is
CALL string-expression string-name.
The most elementary kind of string expression is a quotation. Other kinds are described later. Within a quotation, a slash causes a line-break in the output. The general form of the CALL statement that defines a list is
CALL 'first-element, second-element,..., last-element' THE generic-name LIST. where first-element,... stand for the actual strings that are the elements of the list. The comma acts as a separator between list elements. To include a comma within a list element it is preceded by an asterisk. An individual element of a list of strings also can be defined by a CALL statement of the form:
CALL string-expression THE m-th generic-name.
A SET statement in SNAP gives a value to a quantity name, as in SET N to 1.
or to a list of quantities (see reference 4, page 183). The PRINT statement has the general form:
PRINT string-expression.
A string expression in SNAP can be either (i) a quotation; (ii) an 'unsubscripted' name (e.g. THE THEME) that was given to a string by an earlier CALL statement or by an input statement (see reference 4, page 20); (iii) a 'subscripted' name for an element of a list of strings, e.g. THE M-TH GIFT, (iv) an extract expression of the form
THE m-th CHARACTER OF string-name
or
THE m-th THROUGH n-th CHARACTERS OF string-name
(v) a quantity name, (vi) two or more items of the kinds (i) to (v), joined by the word THEN. A name may be a word, or a phrase, or an arbitrary combination of letters and digits.

The unconditional transfer statements in SNAP have the form {(CONTINUE|REPEAT)}{(WITH|FROM)} [THE] transfer-address.
The transfer address may be a statement label that is displayed in parentheses before a statement in the procedure, or it may consist of certain words such as BEGINNING, NEXT SENTENCE and AS FOLLOWS. The conditional statement has the form IF condition success-action [OTHERWISE fail-action].
Omission of the OTHERWISE part implies OTHERWISE CONTINUE AS FOLLOWS. It could have been omitted accordingly from the first IF statement in the 'Twelve Days' example given earlier.

A SNAP procedure at present consists of a sequence of (i) imperative sentences, that begin with fixed words of language, and (ii) conditional sentences of the form shown above that contain clauses in the success and fail action parts which individually satisfy the syntax of the unconditional sentences. Indicative mood sentences are being added, to allow format and syntax definitions for use during the execution of a procedure. The principal verbs used in SNAP at present are PRINT, CALL, COPY (explained below) APPEND, INSERT, OVERWRITE, DELETE, READ, SELECT (an I/O device), FETCH (input from the selected device), PRINT, PUNCH, WRITE (on the selected I/O device), SET, INCREASE and DECREASE. Some simple constructions are used in conjunction with these verbs to express standard operations in a 'natural' manner, e.g.
OVERWRITE string-expression ON THE n-th AND SUBSEQUENT CHARACTERS OF string-name.
The COPY instruction originally had the form:
COPY string-expression, AND CALL THE COPY string-name.
The purpose of this instruction is to form, in memory, a copy of the string that is specified in the first clause, and to give to this copy the name that is specified at the end of the second clause. For conciseness, IT is now written in the second clause, instead of THE COPY, but with the same interpretation. The string that a COPY instruction produces is not affected, after the instruction has been executed, by a change in the interpretation of a name that forms part of the string-expression in the first clause. This is in contrast to an ordinary CALL instruction, which automatically changes the interpretation of the definiendum to correspond to changes in the interpretations of names in the defining expression, after the instruction has been executed. This 'implied redefinition' convention is very useful in a variety of circumstances. Some simple examples are given in reference (4), that include a procedure to print the 'Twelve Days' without using lists or indexing.

The SNAP language that is now in use has a certain amount of further detail. The 'Twelve Days' example and the general comments of this section, however, provide a sufficient basis for the discussion of the SNAP project in the sections that follow.

3. AN ORDER OF DEVELOPMENT

The elements of SNAP are presented in the author's course, and in the text (4) in an order that permits the student to start running simple examples on the computer after learning just to use MICROSNAP, i.e. instruction of the form

(i) CALL quotation string-name.
(ii) PRINT quotation string-name-defined-by-an-earlier-CALL-instruction.

(iii) EXECUTE.

The word THEN is introduced next, to form MINISNAP. The READ instruction and REPEAT FROM THE BEGINNING are then added to enable the students to print the entire contents of data decks, and then extract expressions, to permit the selection of individual card fields. Statement labels and REPEAT instructions that refer to these, and IF instructions that compare strings and test for an end of file condition are presented next. Arithmetic operations and tests are then introduced and indexing is explained. The use of literal ordinals in extract expressions, such as THE N-TH CHARACTER OF string-name then provides the basis for string scanning operations. The string processing verbs COPY, APPEND etc. can be introduced at this point, or left until after a discussion of simple uses of lists of strings, such as the 'Twelve Days' example quoted earlier.

The author's students use IBM 360 computers, under OS-360. Considerable impetus is given to the course by providing students during the first lecture with control cards to run their first SNAP procedure. The novelty of punching cards and the extent and apparent arbitrariness and unintelligibility of the control card conventions compound the likelihood that if this is not done control-card errors will paralyze some students initial efforts to use the computer. Each student is given a complete deck, ready to run, that contains the trivial SNAP procedure.

PRINT 'JOHN DOE'. EXECUTE.

This gets the student past the hurdles of submitting a job, obtaining the output, and finding the results within the morass of systems messages, which are not inconsiderable issues of morale and logistics. Success with these feats emboldens the student to replace the SNAP card with one that contains his or her own name in the PRINT instruction. After the machine prints this, the rest is incremental progress, for most.

4. MICROSNAP AND MINISNAP

The point that computers can reduce human effort is made at the start of the author's course by examples that print more than the user types. This capability in fact is an important element of many applications of computers to mechanized publication. A very simple prototype is the production of two line display labels that repeat a common item of information under items that vary from one label to another. This can be developed in relation to practically any discipline or special interest group and has multilingual possibilities (for the Pax Romana Trade Fair example in reference 4, the Mesopotamian Merchants Mart variously can use as their slogan – 'You cannot error make a bloomer with the wares of ancient Sumer', 'Il n'y a pas des erreurs avec produits de l'Ur' – etc.) Processional scenes of stylized animals, and panoramic vistas that repeat a few topographical elements, composed of patterns of letters, printed along the length of the output stationery enthuse some of the artistically inclined. These, and form letters and catalog cards that are of practical interest all fall within the scope of MICROSNAP.

The further capabilities of MINISNAP, provided by the word THEN allow the mechanized production of a surprising variety of printed products of general and specialized interest. Tourney rosters afford a convenient plot mechanism for introducing combinatorial problems, and whilst the Arturian Tiddlywinks match of reference 4 may seem a little parochial, outside England, a singing match at a fictitious Council of Varyablysz, requested as a diversion, perhaps by the Littoral powers, has potential for mediaevalists across the Continent.

MINISNAP permits the introduction of some more abstract issues, at the cost of negligible linguistic machinery. Problems of optimization, to minimize keyboard work, can be illustrated by the production of subject and author entries for articles in successive issues of a professional journal, to be included in a bibliographic index that covers a range of journals. These entries exhibit a kind of hierarchical repetition, that typically is produced by nested loops using the full power of a programming language. MINISNAP itself can save keyboard work by giving names to pieces of bibliographic data that are repeated from entry to entry. The number of levels at which this is advantageous can vary however, from journal to journal, depending on such factors as the length of the name of the journal and the average number of authors per article. This makes the point, at an early juncture, that the optimum tactic for a problem is often sensitive to the 'fine structure' of the data, that may vary from one corpus to another, in ways that often are known to just the applications expert.

Another optimization problem can be built around the production of a multilingual calendar, in which each day is represented by four lines that contained respectively (i) the day of the week, (ii) the ordinal of the day within the month, (iii) the month and (iv) the year, each printed side by side in ten languages. A direct MICROSNAP approach is tedious, but only about one tenth as tedious as typing the calendar directly. The word THEN in MINISNAP permits hierarchical definitions of string names, that can provide still further savings. Enumerating the alternatives, and selecting the shortest systematically may give pause for thought. The 'Dietmobile' example of reference 4, that is concerned with the production of 81 low calorie table d'hote menus, that each consist of a different combination of appetizer, fish, meat or poultry, and dessert dish, taken from three alternatives for each, can be handled in MINISNAP in a variety of ways, that include the use of implied redefinition, an the use of a procedure to write a longer procedure, to write a still longer procedure, to print the set of menus that is longer still.

These examples have been cited to stress the fact that knowledge of just a small amount of simple programming grammar can provide scope for ingenuity and sytematic analysis of the 'rhetoric' of programming – the way statements are used in conjunction with each other, over and above their individual grammatical structure. The author believes that this programming rhetoric warrants far more extensive study, systematization and pedagogic presentation than it has received to date. A study of programming rhetoric in fact may be made tractable by an initial direction to procedures that consist of instructions restricted to a relatively simple grammar. Interest in programming languages of late has tended to focus on the syntax of artificial languages of ever increasing complexity. This preoccupation with 'powerful' languages, and the tempo of the computational scene in some quarters seem to be leading to widespread programming efforts that use linguistic steamrollers to crack logical nuts, in disdain or perhaps ignorance of the simple solutions that exist to simple problems. The virtue of simplicity, an the traps of

tortuousness are dominant themes in the teaching of verbal expression, certainly to the author's knowledge in English, and finctional characters have epitomized linguistic extravagance in the literature of many lands. There are lessons to be learned here by computer users. The idea of cost effectiveness may well find application to programming languages, for example in relation to grammatical complexity and conciseness of expression, with possible bearing on the economics of processor implementation and use, and machine design. It may even have implications for the study of how natural languages have envolved.

5. THE DESIGN AND USE OF RECORDS AND FILES

. The introduction of the READ instruction, and the presentation of the further elements of SNAP, in the order described in Section 3, provide the basis for an extensive discussion of the design and manipulation of files of reference data, in the author's course. This is particularly important in the context of library school education, in view of the two way relationship to several major elements of librarianship. On the one hand librarians use files of bibliographic data concerning monographs, serials, individual articles and other published works, to find citations, descriptions and the actual physical location of these materials, for themselves and for library patrons. The task of putting these files in machine readable form, and processing them mechanically has been started in several centers, and this kind of work may be predominantly, if not totally, mechanized as the years progress. Reference librarians, moreover, deal with reference materials that span the totality of human knowledge, an in many instances these materials should be put into machine readable form and processed mechanically. Here then are problems of record and file design for materials that librarians already use, and in some cases write. Reciprocally, such traditional library issues as descriptive cataloging, the design of systems of subject headings, and filing rules, have direct bearing on the design of data bases that is being attempted in some instances by personnel unaware of the traps and snares that beset these matters, and of the elaborate systems that already have envolved through the overall activities of a profession of long standing and world wide experience.

To begin the discussion of record and file design, the author uses a simple fixed field, one card per record file, that contains for example, elementary biographical data of Presidents of the United States. Each student is required to suggest another topic for a file and to design records and files of reference information that pertain to it, and which illustrate the design features that are discussed in class. Within the framework of fixed format records, these include:

(i) the use of a fixed number of cards, greater than one, per record; (ii) the choice of words or phrases to express alternative 'values' of a qualitative property (this is intended to make the students think about the problems of designing classification schemes, and of defining terminology); (iii) the representation of a 'value' by an abbreviation or code, when this is advantageous; (iv) multi-component values of fixed multiplicity, that may be codified advantageously; (v) the elision of characters that punctuate or subdivide a field; (vi) the elision of units of measure; (vii) the elision of other redundant characters (e.g. century digits in a date), that can be restored trivially or algorithmically; (viii) the completion of an item of excessive length on a suitably identified continuation card; (ix) the representation of properties of variable multiplicity, in fixed format records, when the 'values' have to be given explicitly, when they can be codified, and when combinations can be codified; (x) hierarchically structured records, in which the levels of the hierarchy are of different status that can be designated by a code (e.g. records that pertain to countries, that contain subrecords of increasing subordination that pertain to states, counties, municipalities and wards); (xi) explicit and codified cross references between the records for items that have a special relationship, though of equivalent status in the file as a whole (e.g. a reference from the record for one monarch to that of his father if he ruled the same domain, in a dynastic file).

Then, within the framework of free format records, attention is given to (i) the use of field separators; (ii) the specification of field length at the beginning of a field; (iii) the use of a list of pointers (easily conveyed as analogous to a table of contents) at the beginning of a record; (iii) positional parameters; (iv) keyword parameters and precedence codes; and (v) default interpretations.

Although the design tactics that have been enumerated may seem self evident to many programmers, it is the author's experience that many of his students do not consider them so, and do appreciate their explicit identification and mention. The programming rhetoric to extract information from records that use these design tactics is discussed in the course and in the text. The discussion involves elementary scanning tactics and the use of subscripted variables (lists, in our terminology). Consideration is given also to the 'rhetoric' to augment the extracted information, to make it easily understood, for example by incorporation in verbal messages that are grammatically consistent.

The discussion of internally and externally expressed file structure follows naturally from questions of record design. The file of hierarchically structured geographical records mentioned earlier can be regarded as a hierarchically structured file of records that individually contain information about a territorial region as apart from its constituents. To facilitate certain kinds of search, structural information can be included in the records, for example (i) to point forward from the record for one territorial region to the record for the next region of equal or higher status, and (ii) to point backward from one record to that for the region that subsumes it. The former tactic can be applied in records that are added to a file to permit skipping over portions which share a characteristic by which the file is ordered, in a searching operation (e.g. including heading cards that give the number of records beginning with the same letter in an alphabetically ordered file).

Programming tactics that deal with lists (as we employ the term) permit the external expression of the structure of a file, by a table of pointers, to be used. Consideration of the internal and external expression of the structures of such tables opens up further possibilities of file design, and further problems of enumerating the alternative ways to organize a data base, and selecting the optimum. The trade-off between avoiding redundancy to save space, and allowing it to save time, and the self defeating aspects of some ways of trying to achieve economies, are stressed.

The author introduces input/output operations by reference to punched cards and the line printer. The discussion of file processing however leads inevitably to the consideration of magnetic tape and direct access storage

devices. Most students in the author's class show some antipathy, or worse, towards control card conventions at the outset of the course, that is sidestepped in the manner mentioned earlier. The enthusiasm engendered by processing a file on a topic that interests the student, and the confidence gained from running jobs successfully, and being able to find and correct the errors when jobs fail, however do create an excellent climate later for introducing the elements of job control language, in relation to input/output operations (9). Some students even go on to elementary use of data set utilities, during the semester. This is encouraging, in view of the central role of control language and utility programs in the mechanized processing of large bodies of reference information, and the exchange of reference files between installations that are almost, but not quite compatible; and the difficulties that often result from an excessive dependence on full time systems programmers in such circumstances.

6. SOME MORE ADVANCED TOPICS

The author ends his present course with (i) some elementary examples of text processing, (ii) examples of information that can be displayed as trees, and as multiply connected graphs, (iii) their representation for input to the computer, and (iv) simple algorithms to derive information from these. Genealogical tables (i) that show only male issue, (ii) show all issue and cousinly marriages and (iii) the deific relationships of a mythology, provide examples of graphs of increasing complexity, within the educational experience of most humanists. The author can only touch briefly on these issues in his present course, but the response has been extremely encouraging from students who, almost without exception, cannot differentiate a square.

These students have also responded very well to the use of set terminology, to systematize some aspects of file and record design. A 'Mathematics for Librarianship' course is being proposed accordingly, that would cover (i) the use of symbols for names and operators, and the grammar of symbolic expressions, (ii) enumerative subscripts and collective operator (e.g. sigma) notations, (iii) the elementary ideas and notations of sets, (iv) Boolean logic, (v) algorithms for sorting and merging, (vi) a treatment of trees and multiply connected graphs that is more extensive than the present course allows, (vii) combinatorial problems of optimization, (viii) statistical terminology, and (ix) some elements of an algebraic notation for symbol manipulation, that is easily adapted for input to the syntactic analysis and table driven translation subroutines of the new SNAP processor. These topics all bear strongly on the use of computers in information processing by persons trained in the humanities. The author believes that they can and should be taught to such people. In this regard, the increasing attention that finite mathematics is receiving in liberal arts curricula, and the publication of texts on this subject for humanists should be noted (see, for example reference 10). There seems to be scope also for more advanced systematic instruction in mechanized text processing, using table driven techniques, and the author plans to explore this area, in relation to the relevant capabilities of the new SNAP processor, that are discussed next.

7. SYNTAX AND SNAP

The prototype SNAP processor that was released to IBM P.J.D. last year, after running at Columbia University since 1967, was developed in a rather tentative fashion. It was written in a subset of FORTRAN IV, and can operate in 105K bytes on an IBM 360 computer. Portions of SNAP statements are delimited and translated by sections of code that would have to be rewritten (at least at the FORTRAN level) to allow even minor changes in the SNAP syntax. This resulted in part from the uncertainties about the SNAP project, when it was started at the end of 1966, to see if a processor could be written inexpensively for a stylized form of natural language that humanists could learn and would be willing to apply to problems that did exist in reasonable numbers. Because these points were all in question, a minimal subset was implemented first, as rapidly as possible and this was followed by a succession of short term incremental efforts. Writing the prototype processor, however, and using and teaching SNAP with its aid gave the assurances needed to undertake the new processor, in a way that promised much greater flexibility.

The new processor will work optionally in a restricted or an expanded character set, accepting input from punched cards, paper tape or magnetic tape images of these, (depending on the computer configuration) and recording output on line printers with upper case or dual case print trains, or on media to drive other devices. The processor consists of a main program that can immit syntax and transformation tables (or use ones that are compiled in labelled COMMON), and invoke five major subroutines: (1) An input subroutine that obtains, from an input stream, either a record or a sentence, or a string that is delimited in other ways that the user specifies by control statements. (2) A syntactic analysis subroutine that analyzes this record or sentence or other string, by reference to a syntax that has already been stored (see above) and represents the result of the analysis as a 'structure table'. (3) A synthesis subroutine, that constructs the corresponding string of SNAPIC codes, or some other string, by reference to (i) a 'transformation table' that has already been stored, (ii) the structure table of the string that was analyzed, and (iii) the string itself. (4) A subroutine to plant cross references from transfer statements to statement labels. (5) The SNAP interpreter, that interprets a procedure which has been translated into SNAPIC. This organization allows a little more generality than just processing SNAP procedures. The syntax and transformation tables that the main program immits directly (or gets via common) must be expressed in an internal representation that is an extension of the SNAPIC code. It is possible however to 'bookstrap' by immitting short coded syntaxes and transformations to translate, into this internal code, further syntaxes and transformations that are expressed in more convenient verbal or symbolic styles. The style that is employed at present for syntaxes is described below. When the processor is used to deal with a SNAP procedure, the input, scan and synthesis subroutines are invoked in a cycle until an EXECUTE instruction is encountered. The subroutine to plant transfer addresses, and the interpreter are then invoked. To deal with a procedure that is expressed in a non-English verbal or algebraic equivalent of SNAP, all that need be changed is the syntax and perhaps the transformation table that operates on the structure table which the analysis subroutine produces. Some problems of symbol manipulation however can be handled entirely by a

table driven translation in a cycle that uses just the subroutines (1) - (3) described above. The inclusion of appropriate control cards in the input permits the processor to be used in just this way, recording the strings that are synthesized at the end of each cycle (1) - (3), without proceeding to the steps (4) and (5). Table driven translation can also be performed within a SNAP procedure, since there are SNAPIC operation codes that make the interpreter invoke the analysis and synthesis subroutines. At the time of writing, the syntax and transformation tables that these use are immitted in the style that is adopted for the tables that perform the SNAP to SNAPIC translation, pending completion of the syntax of a verbal equivalent. This will resemble, at least for the syntaxes, the style that is used to describe the SNAP syntax in reference (4).

The form of syntax description that has been used for the last few months, to debug parts of the SNAP processor, and to test the consistency of the SNAP conventions, resembles that used by the author in his earlier SHADOW work (11). A generic name is devoted for simplicity by the letter S followed by a numeral; thus S6 denotes a quotation and S48 denotes a DELETE instruction in the current SNAP syntax. A generic name is defined by a syntax statement of the form:

generic-name IS generic-expression.

A generic expression consists of either (i) a simple generic expression, or (ii) two or more simple generic expressions, joined by the word THEN. A simple generic expression is either (i) a quotation, that begins and ends with quote marks (within a quotand *"signifies a quote mark, and** signifies a single asterisk), (ii) a generic name, (iii) an expression of the form SOME Ln, or SOME Ln THEN generic-expression, where n stands for an integer, and the Ln are names of lists of strings, that are augmented as a side action by the synthesis routine. In the SNAP to SNAPIC translation, for example, six of these lists are used, for unsubscripted and subscripted string and quantity names, and for apposited and referenced statement labels. The first form of SOME expression is satisfied as soon as an element of the list is matched, the second form progresses through the list until an element followed by a string that fits generic expression within the range of the statement that the comma delimits is matched.

(iv) an expression of the form

EITHER first-generic-expression OR second-generic-expression...

OR last-generic-expression,

(v) expressions of the form

[ARBNO MINNO n | MAXNO n | EXACNO n] generic-expression.

The ARBNO operator makes the analyzer try to delimit a substring that starts at the current scanning position and satisfies the generic expression that appears as its operand. If this cannot be done the construction is considered to be satisfied by a substring that is nul, and the analyzer goes on to the next portion of the syntax statement. If a substring that fits the generic expression is found, however, another is sought immediately after it. This process is repeated, for as many substrings of the requisite form that are concatenated from this point in the string being analyzed. The MINNO n operator requires at least n substrings that fit the operand expression, and if this minimum is satisfied, it takes as many as there are. The MAXNO n operator begins in the same way as the ARBNO operator, and discontinues when no more substrings of the requisite form can be found in sequence, or when n have been found, whichever occurs first. The EXACNO n operator requires n substrings of the specified form to occur in sequence.

(vi) ARBSTRING generic-name ENDED BY generic-expression,

This delimits a string that starts at the current position of the scanning pointer, and ends immediately before the first substring that satisfies the generic expression which follows the words ENDED BY. The scanning pointer is moved past this delimiting substring, and it is given the name that appears after the metaword ARBSTRING in the syntax statement.

(vii) ARBSTRING generic-name STOPPED BY generic--expression,

This creates an entry in the structure table similar to that described in (vi), but the scanning pointer is left at the beginning of the delimiting substring.

(viii) NOT generic-expression-1, THEN generic--expression-2

This is satisfied by string that starts at the current position of the scanning pointer and satisfies the generic expression 2, provided that a string which satisfies generic expression 1 does not start at the current position of the scanning pointer too.

Every generic name that appears in the object of a syntax statement must appear as the subject of a syntax statement, with the exception of a name that follows an ARBSTRING operator. Generic expressions may be nested, and statements may be non-recursive, directly recursive, or indirectly recursive.

Transformation expressions are written at present in an algebraic style. A transformation name is denoted by the letter T followed by a number. A transformation statement equates a transformation name to a transformation expression. This may consist of a single simple transformation expression, or two or more of these, joined by commas to signify concatenation. A simple transformation expression may be, for example, a constant string, a transformation name, or an expression that denotes a copy of a substring of the analyzed input that is identified by a row of the structure table. Several tree climbing operators are used in transformation expressions to shift attention within the structure table in an appropriate manner. Other operators append strings and quantities to name and pointer lists, and plant flags and replace them. The further details of the transformation expressions will be reported in due course. They have several convenient algebraic properties, that raise some interesting questions in relation to the mechanization of algebraic manipulations.

A superset of the SNAP constructions described in Chapters 3-10 of reference 5 can be defined by a syntax of less than 100 statements, that are almost all less than 80 characters long. The corresponding transformation table to form SNAPIC is even shorter. The SNAP syntax is premised on an approach that actually runs counter to the one which seems customary in compiler design. It may well be that the path we have followed is necessary to avoid self-defeating restrictions on the use of some very common words. A name can appear for the first time in a SNAP procedure in only a few kinds of context. A string name may be introduced as the trailing definendum of a CALL instruction, or as the object of an input instruction, or at

the end of a RESERVE SPACE instruction. A quantity name can be introduced in a SET instruction.

A string expression in say a PRINT or CALL instruction, is scanned from left to right testing for quotations and previously defined names, in alternation with the word THEN. The definendum of a CALL instruction then is identified when the last (or only) simple string expression in the direct object has been found by content, and is not followed by THEN. The definendum itself could contain the word THEN, and be used in the direct objects of instructions that came later in the procedure. In the analysis of a SET instruction, the quantity name that it defines is delimited by scanning not for just the word TO, but instead for TO followed by an acceptable quantity expression. This permits quantity names to contain TO, which at times would be hard to avoid (e.g. THE DISTANCE FROM AIX TO GHENT). Of course examples can be constructed to confound this scheme, but the objective is naturalness in circumstances that are likely rather than rigorous safeguards against the exceptional or deliberately bizarre. The principle of scanning for previously defined names, and treating the residue of the sentence as connectives will be particularly important when the subroutine capability is added to SNAP. The totality of the verb and this residue will identify the subroutine, without improving on the syntax an enumeration of the roles that these connectives serve, or limiting their use by writers of procedures (see reference 4, Chapter 11).

It is possible that the design of the SNAP language has been made particularly simple by the tolerances of English grammar that some other languages do not share. In the statements

CALL 'HENRY II, JOHN,...' THE ANGEVIN LIST.
PRINT THE 1-ST ANGEVIN.

the word ANGEVIN is used first as an adjectival modifier of the noun LIST, then as a noun that is qualified by an ordinal adjective. This could be avoided however, by CALL '...' THE LIST OF ANGEVINS. PRINT THE 1-ST ELEMENT OF THE LIST OF ANGEVINS. Then instead of dealing with singular generic names, that appeared in the contexts THE generic-name LIST and THE m-th generic-name, the translation would deal with pluralized generic names that appeared in the contexts. THE LIST OF pluralyzed-generic-name and the m-th ELEMENT OF THE LIST OF pluralyzed-generic-name.

Again, in the constructions

CALL 'ABRACADABRA' THE SPELL. PRINT THE 3-RD CHARACTER OF THE SPELL.

The SPELL does not suffer any inflectional change from its change of case. The nominative could be preserved however by a circumlocution such as PRINT THE 3-RD CHARACTER OF THE STRING THAT IS CALLED THE SPELL. The author hopes that variations such as these will facilitate the design of SNAP equivalents for other languages. A first attempt at a Spanish SNAP syntax, made with the help of a Spanish speaking colleague, paralleled the English one quite closely, in its hierarchical structure, with changes in about one half of the individual statements because of differences in vocabulary and grammar.

Even though English is relatively uninflected, word variations often are required for 'naturalness' to a reader, though inessential to the interpretation of procedures in which they occur. A simple expedient is to ignore such variations, by treating the variant forms as equivalent. It may even be possible to ignore some such variant words entirely. This is done at present with the articles A, AN and THE. Names of the form A s, AN s, THE s and s where s denotes a word or phrase, are considered synonymous. This facilitates the processing of constructions such as (START) READ A RECORD. PRINT THE RECORD. REPEAT FROM THE START. in which the switch from A to THE may have parallels in other languages (see reference 4, page 84 and reference 12)

Beyond its use in running procedures in English SNAP, and perhaps in equivalent stylizations of other natural languages to teach elementary programming and related matters to humanists, the SNAP processor may have several other educational uses. The development of an algebraic equivalent of SNAP, to teach symbol manipulation to symbolically oriented students is in hand, and could well find a place in the curricula of mathematical physics, in connection with the tedious algebraic manipulations that beset such work (see for example, reference 13). SNAP is being used in an experiment to construct illustrative examples mechanically, for classroom and printed instruction in such matters as systems programming. The examples illustrate variations and combinations of principles in a systematic fashion, and are produced by repeating and varying the details and combinations of pieces that are much shorter in aggregate than the sets of examples constructed from them. Sets of data set utility jobs, that print self explanatory messages concerning the control card options that they contain have been generated in this way. Once the pattern of such a progression of examples has been embodied in a procedure to generate them, moreover, new sets can be produced mechanically to correspond to local systems conventions and written languages, just by changing parameters and input data. And finally a systematic study of stylizations of natural language of increasing complexity could be attempted, for procedural, interrogative and descriptive forms of expression.

Language is stylized to some extent in many forms of communication and publication today, for example in concise accounts of biographical and geographical reference information. The power of sentential expression, albeit of restricted form, over tabular expression is illustrated by its ability to express situations concisely that have such varied emphases as:

He lived mostly in London, but visited Paris, Rome and Genoa in 1763, and Madrid, Avignon and Brussels in 1765.

and

He lived mostly in London, but visited Paris in 1862, 1866 and 1867, and Rome in 1863, 1864 and 1869.

A seminar on text processing could be organized around (i) the adaptation of existing stylizations of reference and other materials, to forms that are acceptable to human readers, and machine processable by e.g. the SNAP processor, and (ii) the design of procedural string processing languages, of a higher level than SNAP, suggested by the experience of writing SNAP procedures for a variety tasks. This could complement a presentation of the practical operations of text manipulation, required in mechanized publication, that have been enumerated previously (6).

ACKNOWLEDGEMENTS

The author's indebtedness continues to the several past and present colleagues whom he has thanked in earlier accounts of the SNAP project. In addition thanks are due

to Mr. James Humphry III and Mr. Leo M. Weins of The H.W. Wilson Company for their encouragement and support of the work on the new SNAP processor; to the staff of the Data Station Corporation, whose computing facility has been used for this work; to Mr. Raymond DeBuse of the Department of Libraries at Columbia University for his help in conducting the author's course; and to the several hundred students that the author has now taught, for their attention, enthusiasm, tolerance and ideas.

REFERENCES

(1) M.P. Barnett, Natural Language Programming and Text Processing, TAGA Proceedings, 276, Technical Association of the Graphic Arts, Rochester, N.Y., 1967

(2) M.P. Barnett and W.M. Ruhsam, A Natural Language Programming System for Text Processing, IEEE Transactions on Engineering, Writing and Speech, Vol. EWS-11, 45, 1968

(3) M.P. Barnett and W.M. Ruhsam, SNAP-An Experiment in Natural Language Programming, Proceedings of the Spring Joint Computer Conference, 75, AFIPS, 1969

(4) M.P. Barnett, Computer Programming in English, pp 259, Harcourt Brace & World, 1969

(5) M.P. Barnett, The SNAP Approach to Mechanized Publication, Seminar on Automated Publishing, University of Newcastle upon Tyne, in press

(6) M.P. Barnett, SNAP-A Programming Language for Humanists, Computing and the Humanities, 4 (4), 225, 1970

(7) PID reference 360D 03.3.010, Program Information Department, IBM, N.Y. PAL reference 97-20-270, RCA Information Systems Division, Cherry Hill, N.J.

(8) I. Opie and P. Opie, The Oxford Dictionary of Nursery Rhymes, 122, Oxford University Press, 1951

(9) Memorandum CUCC-159, Columbia University Computing Center, 1970

(10) M. Marcus, A Survey of Finite Mathematics, Houghton Mifflin Company, Boston, 1969

(11) M.P. Barnett and R.P. Futrelle, Syntactic Analysis by Digital Computer, Comm. A.C.M. 5,515,1962

(12) Otto Jespersen, A Modern English Grammar on Historical Principles, Part VII, completed by Niels Haslund, 417, Copenhagen, 1949

(13) M.P. Barnett, The Evaluation of Molecular Integrals by the Zeta Function Method, in Methods of Computational Physics, Vol. II, 95, Academic Press, 1963

(14) G.E. Koster, Columbia University School of Library Service, unpublished work

TEACHING AND LEARNING LANGUAGES

C. Bellissant

1. INTRODUCTION

In all instructional processes there are at least two characters: on one side the teacher and on the other, the student. The liaison between the two is established in various ways. The simplest way, oral communication, still seems to have a few golden days ahead of it. Books, whether programmed or not, are the mediums employed with increasing frequency. Audiovisual techniques have barely been mastered, and people are already including computers into the process.

A tenacious theory predicts that machines will soon replace teachers and will, of course, do a much better job in teaching than teachers themselves. Luckily, we have not yet reached such a point. The computer is only a tool and ought to remain as such, especially in an area as important as education. Computer assisted instruction systems should be considered in the same class as the other intermediary agents mentioned above. Moreover, it would be highly appropriate to employ a combination of the various means of communication between teacher and student, given the inherently complementary nature of these media.

One can introduce the computer into instructional processes at many different points: for example, in experimenting with models of knowledge attainment, in statistical evaluation of results obtained, in planning of the scholastic activities, etc... Here, I shall deal only with the use of computer by the teacher when he writes his course on the one hand, and by the student as he pursues his learning on the other, in short, at the two extremes of the instructional process.

These two applications of computers depend essentially on the degree of liberty which one wishes to leave to the student. If one wishes to guide the student along his way in the course, one will use a language of the type „course-writer", where the program involved strictly controls the progress of the session and practically nothing is left to the initiative of the student. Conversely, if one wishes to leave him with the ability to act on his own initiative, one will then employ a language with considerable conversational orientation.

2. TEACHING LANGUAGE

One may ask oneself what is the justification for specialized teaching languages. In fact, there already exist so many programming languages that it should suffice to use one of them in the context of automated teaching. The language most suitable for this purpose would appear to be SNOBOL 4. It contains a whole set of statements needed for parsing and constructing strings of text, as well as very sophisticated pattern-matching facilities. Unfortunately, its syntax makes it difficult to use by a non-specialist in computer science. If one wants to spread this mode of teaching, however, it is necessary to entrust to the teacher the task of writing his own course. One is therefore led to define a new type of specialized language, specifically adapted to this application.

The principal properties required of such a language are the following:

2.1 The language should be as general as possible. This is to say that it should be able to accommodate a wide spectrum of courses, with regard to the content, the form, and the teaching method they embody. Thus, it should not be restricted either to a particular discipline or to a particular pedagogical strategy.

2.2 It must be capable of being easily mastered by those teachers who wish to use it. They are not computer scientists, and are easily discouraged by complicated syntax or semantics. However, it should also avoid the opposite extreme, which consists of providing, in the name of simplicity, a language which is inefficient owing to an insufficiency of concepts. A careful balance between these two objectives must be achieved.

2.3 The teaching language should be readily expandable in order to allow the introduction of new procedures, necessitated by the needs of a particular teaching strategy. This implies that the definition of the language should be modular. The technique of interpretation would seem best suited to this purpose in that it is relatively easy, given this mode of operation, to add new facilities. For example, the teacher might wish to use in his course a different module for response analysis in place of that which he has been using. The computer scientist should then be able to quickly extend the power of the language in the desired direction.

2.4 The actual use of the language must be convenient. This strongly suggests a conversational environment, letting the teacher write his course at the terminal, where an incremental compiler can detect the syntactic and semantic errors as they appear.

2.5 The language must facilitate the preparation of courses which will not seem monotonous in the eyes of students. This can be obtained by a randomizing process, applied both to the manipulated quantities and to the statement sequences comprising the course.

3. LEARNING LANGUAGE

The first question to be asked about a learning language

is what is its intended purpose. There are, in fact, several conceivable applications of such a language. It could be, for example, an ancillary facility available to a student in order to perform computations during the learning process. In this case, it corresponds in function to an office calculator. It could also be a language which permits the student to pose questions to the system when he is interrupted in the course of a lesson. One has then to consider a program of assistance which recognizes a certain number of key words in the student's request, and responds appropriately. Finally, it could be a complete language allowing the description of data structures and operations to be effected on them. It is this type of language which one uses in simulation and gaming experiments. In this situation, the learning process is almost entirely encompassed within the domain of the learning language; the task of the teacher is confined to developing the situation to be simulated.

Obviously, such languages necessitate a conversational environment, since the requests must be executed on the spot. Here the technique of interpretation is evidently the most appropriate, since repeated evaluation is not a factor. These different objectives of a learning language could reasonably be combined within a single system, with the result that the overall flexibility of the system is significantly increased.

4. THE SYSTEM USED AT THE UNIVERSITY OF GRENOBLE

The Laboratory of Applied Mathematics at the University of Grenoble has at its disposal an IBM 360/67 computer accessible from about 50 terminals, including both typewriters and graphic consoles. Researchers of the laboratory use the conversational time-sharing system CP/CMS. CP is a simulator of „virtual machines", which gives the effect that each user has available from his terminal his own standard 360 with its memory and all of its input-output devices. The computer assisted instruction system has been imbedded in this environment. This fact permits the computer to support other tasks in parallel with that of teaching. The teacher has his own machine available as does each student. It is the teacher's machine which supports the first phase of the process – the development of a course. A conversational and incremental compiler analyzes the course typed by the teacher on his terminal and identifies all errors as they appear. The second phase is supported, for each student, on his individual machine. This phase deals with the execution of the interpretive code provided by the compiler. A student can disconnect at any moment and invoke the program of assistance which will analyze his requests.

5. MAGISTER, AUTHOR-LANGUAGE USED AT THE UNIVERSITY OF GRENOBLE

For the programmer, MAGISTER is a formal language oriented towards string manipulations. Within this language one treats characters or groups of characters by either presenting an item to a student or analyzing his answers. A course in MAGISTER is composed of one or several chapters. Each chapter is in fact a program for the compiler, and a pedagogical unit corresponding to a learning session.

Lines typed by the teacher are numbered by the compiler and can also be modified afterwards; they are located by means of this number. When the contents of a line is syntactically incorrect, the error is localized by a pointer and the compiler reprints the same line number until a correct line is obtained.

A chapter is composed of a series of statements whose execution is usually carried out in sequence, except when otherwise indicated. Variables manipulated in the language are text strings and they may be referred to by name. A string composed of numbers can evidently serve as the object of any ordinary arithmetical operation. A concatenation or partition may be applied on any kind of string.

5.1 *Goto statement*

All instruction can be labelled in order to allow branching. There are two kinds of branches: simple transfer to a labelled instruction, or activation of a sequence of statements before returning after the calling instruction. Two basic symbols correspond to these two kinds of transfers.

ALLERA and ACTIVER.

The instruction: ALLERA reading;

is a simple transfer to the instruction labelled „reading".

The instruction: ACTIVER reading;

has the same effect plus a return to the instruction which follows the transfer. This return takes place at the first appearance of a period (.) at the end of an instruction.

In fact, during the interpretation of an object program, when a period is encountered at the end of an instruction, two situations may be considered: if the sequence of instructions which has been just executed was called by means of an ACTIVER instruction, then a transfer to the instruction following the calling statement is executed, if not the next instruction in sequence is taken.

This notion of sub-program is similar to the execution of a PERFORM instruction in COBOL.

The two instructions for transfer, ALLERA and ACTIVER, may have as operand not only a label but a list of labels.

```
example: ALLERA  e1 | e2 | e3;
         ACTIVER e1 | e2 | e3;
```

During execution we will execute a transfer to one of these three labels with a random decision.

This allows the teacher to present random problems taken from a group of exercises having the same teaching result. This list notion will be found in writing and assignment instructions.

Some instructions of MAGISTER such as a reading during a limited period, a research or a comparison lead logically to the verification of a test. For example, whether we found such and such a sequence in the student's answer, or whether the student has answered or not within seconds, etc...

The success or the failure of this test can be verified by means of a prefix which permits the conditional execution of an instruction.

The two prefixes are: – (yes, the test is satisfied).
¬ (no, the test is not satisfied).

Example:
```
LIRE EN 10 SECONDES;
¬ECRIRE „faster! "
```

If the student has not answered within 10 seconds, he will see printed on the terminal the message: faster!

Execution of an instruction can also depend upon the success of a condition which is more elaborate than a simple test. It is a conditional instruction whose scheme is: if <Boolean expression> then <unconditional statement> else <unconditional statement>.

A Boolean expression is similar to those in ALGOL except that there are no real Boolean variables and that we always go back to the verification of a relation between arithmetic expressions.

Beside the two instructions already studied, there are in MAGISTER the following unconditional instructions:

5.2 *Read statement:*

The simplest form is: LIRE;

The line typed by the student is then retained in a specific buffer which will be named „last answer".

The program has access to this last answer by research and comparison instructions.

The teacher can add after LIRE an identifier to specify a buffer in which the student's string will be stored.

Example: LIRE answer;

It can also indicate a time limit for the student's answer.

Example: LIRE answer EN 20 SECONDES;

or: LIRE EN 1 MINUTE;

This delay in answering can then be tested by means of a prefixed instruction.

5.3 *Type statement*

It permits to present an item to the student:

Example:
```
ECRIRE „yes";
ECRIRE a;
```
(The contents of the string is typed on the terminal.)
ECRIRE „it is" & n & „o'clock";

(The message which will be typed consists of three parts which will be concatenated at printing time. Of course n is a string identifier, contents of which will be printed.)

ECRIRE a | b | c;

(Here is a list of three expressions appearing as operand. At execution time we will print one among these three expressions according to a random decision.)

ECRIRE a-b/c;

(We will print a result corresponding to the calculation of the expression. A concatenation operator has a priority lower than that of an arithmetic operator: The expression 10 & 5 + 7 has the value 1012 but (10 & 5) + 7 equals to 112.).

5.4 *Assignment statement*

It permits the assignment to a string the value of an expression. Once again, as in type statement, we find the notion of list of expressions.

Examples:
```
a : = a-b + 12 & a;
answer : = a | b | c | 17;
```

(We assign to the string „answer" one of the values a, b, c or 17).

5.5 *Compare statement*

It allows comparison of values between an expression and a list of expressions. The first expression can be the last answer provided, or an expression given explicitly.

Examples: COMPARER AVEC 17 | a + b;

(We compare the value of the last answer with 17, if it is not 17, with the value of the expression a + b).

COMPARER a & „ing" ET c;

The result of the comparison can be tested by a prefixed statement.

5.6 *Search statement*

It permits to search in a string the presence or absence of a certain amount of values separated or not by contexts.

The result of the research can be tested by a prefixed statement.

Examples: CHERCHER „ABC";

(We look in the last given answer for the appearance of the string ABC).

CHERCHER DANS a: „ABC" , „XYZ";

(We look in the string a for the appearance of ABC followed by XYZ eventually after a context of any length.)

CHERCHER „a" & 17 & a;

(We look for the number 17 preceded by a context named a, then we search after 17 for the same context a).

If the last answer provided is: 2172

then the research is satisfied and, in addition, the string a has the value 2.

Thus the search statement operates a truncation of a string followed by one or several assignments if there are contexts).

CHERCHER DANS a : „b" & " " & „a";

(This statement will execute a shortening of all characters preceding the first blank in the string a. It is a convenient way to isolate the first word of a string called here b. The remaining part of the string after the blank is the new value of a.)

Thus, if a is the string: „one two three four"

after the execution of this statement: b will equal to „one" and a will equal to „two three four".

Contexts may be alphabetic, numerical or free, and have any given length.

Examples: CHERCHER '<a,3>' & 12;

We look for 12 preceded immediately by a numerical context composed of 3 digits.

CHERCHER '(a,! b)' ,b;

We search for the value of the string b preceded immediately or not by an alphabetical context called a with the same length as b. The operator ! placed before a string identifier signifies the length of this string. If a equals „ABC", ! a equals 3.

Instead of a string, a substring may occur, for example: a (3 : 7) is the sub-string composed of the 3rd, 4th, 5th, 6th, and 7th characters of the string a. It is another way to cut up a string.

A formal grammar describing the syntax of MAGISTER in Backus normal form is shown in the addendum.

The definition of the learning language used by the student in our system is not yet completed. Its purpose will be to assist the student in getting information about the problemsituation (concepts) and to give him the operations necessary to resolve the situation.

Our system should become operative in October 1971 and the first applications will concern Computer Science at the University level, elementary mathematics and French grammar in the secondary school.

REFERENCES

BELISSANT, C., 1969, MAGISTER, un langage d'écriture de cours. In: l'enseignement programmé, 7, pp. 75-85.

FEURZEIG, W., PAPERT, S.A., 1968, Programming languages as a conceptual framework for teaching mathematics. In: Programmed learning research, major trends, (DUNOD, Paris) pp. 233-248.

FORSYTHE, G.E., 1968, Computer Science and Education. In: Proceedings of the IFIP Congress 1968, vol. 2 (North-Holland Publishing Company, Amsterdam), pp 1025-1037.

FRYE, C.H., GENNIK, F.D., FEINGOLD, S.L., 1968, User's Guides to PLANIT. System Development Corporation, Technical Memorandum. pp 96.

TONGE, F.M., 1968, Design of a programming language and system for computer assisted learning. In: Proceedings of the IFIP Congress 1968, vol 2 (North-Holland Publishing Company, Amsterdam), pp. 1349-1350.

WEIZENBAUM, J., 1968, ELIZA: A computer Program for the Study of Natural Language Communication between Man and Machine. C.A.C.M., vol. 9, pp. 36-45.

ZINN, K.L., 1968, Languages for programming conversational use of computers in instruction, Proceedings of the IFIP Congress 68, vol. 2 (North-Holland Publishing Company, Amsterdam), pp; 1388-1394.

ADDENDUM : Syntax of MAGISTER

<course> ::= <chapter> ,| <chapter> <course>

<chapter> ::= <statement> <punctuation> |
<statement> <punctuation> <chapter>

<statement> ::= <label> : <statement> | <prefixed statement> |
<unprefixed statement>

<punctuation> ::= ; | .

<prefixed statement> ::= <prefix> <unprefixed statement>

<prefix> ::= _ | ¯

<unprefixed statement> ::= <conditional statement> |
<unconditional statement>

<unconditional statement> ::= <read statement> |
<type statement> |
<goto statement> |
<empty statement> |
<search statement> |
<assign statement> |
<compare statement>

<conditional statement> ::= <condition> <unconditional statement> |
<condition> <unconditional statement> SINON
<unconditional statement>

<condition> ::= SI <boolean expression> ALORS

<boolean expression> ::= <boolean factor> |
<boolean factor> OU <boolean expression>

<boolean factor> ::= <boolean secondary> |
<boolean secondary> ET <boolean factor>

<boolean secondary> ::= <boolean primary> | ¯ <boolean primary>

<boolean primary> ::= <relation> | (<boolean expression>)

<relation> ::= <arithmetic expression> <relation operator>
<arithmetic expression>

<relation operator> ::= = | < | > | ¯= | <= | >=

<read statement> ::= LIRE | LIRE <read attribute>

<read attribute> ::= <string> | <string> <time limit> |
<time limit>

<string> ::= <identifier>

<identifier> ::= <letter> | <identifier> <letter> |
<identifier> <digit>

<letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<time limit> ::= EN <unsigned number> <time unit>

<unsigned number> ::= <digit> | <unsigned number> <digit>

<time unit> ::= MINUTE | SECOND

<type statement> ::= ECRIRE <list of expressions>

<goto statement> ::= <transfer verb> <list of transfer>

<transfer verb> ::= ALLERA | ACTIVER

<list of transfer> ::= <transfer> | <list of transfer> <stroke> <transfer>

<transfer> ::= <label> | <label> <chapter specification>

<stroke> ::= |

<label> ::= <identifier>

<chapter specification> ::= DU CHAPITRE <unsigned number>

<compare statement> ::= COMPARER <compare attribute>

<compare attribute> ::= AVEC <list of expressions> |
<expression> ET <list of expressions>

<assign statement> ::= <left part> <list of expressions>

<left part> ::= <string> :=

<list of expressions> ::= <expression> | <expression> <stroke> <list of expressions>

<expression> ::= <simple expression> |
<expression> <concatenation operator> <simple expression>

<concatenation operator> ::= &

<simple expression> ::= <arithmetic expression> | <alphanumeric expression>

<arithmetic expression> ::= <term> | <add operator> |
<arithmetic expression> <add operator> <term>

<add operator> ::= + | -

<term> ::= <factor> | <term> <multiply operator> <factor>

<multiply operator> ::= * | /

<factor> ::= <arithmetic primary> | <factor> ** <arithmetic primary>

<arithmetic primary> ::= <length> | <variable> |
<unsigned number> | (<expression>)

<length> ::= ! <variable>

<search statement> ::= CHERCHER <search attribute>

<search attribute> ::= <search sequence> |
DANS <string> : <search sequence>

<search sequence> ::= <search element> |
<search sequence> <concatenation operator> <search element>

<search element> ::= <context> | <alphanumeric list>

<context> ::= , | ' <alphanumeric context> '

<alphanumeric context> ::= <alphabetic context> |
<numeric context> | <contents of context>

<alphabetic context> ::= (<contents of context>)

<numeric context> ::= < <contents of context> >

<contents of context> ::= <string> , <extent> | <extent> | <string>

<extent> ::= <length> | <unsigned number>

<alphanumeric list> ::= <alphanumeric element> |
<alphanumeric list> <stroke> <alphanumeric element>

<alphanumeric element> ::= <alphanumeric expression> |
(<search sequence>)

<alphanumeric expression> ::= <variable> | <constant>

<variable> ::= <string> | <sub-string> | <part>

<sub-string> ::= <string> (<amplitude>)

<amplitude> ::= <bound> | <bound> : | : <bound> | <bound> : <bound>

<bound> ::= <string> | <number> | <length>

<part> ::= <string> < <number> >

<number> ::= <unsigned number> | <sign> <unsigned number>
<sign> ::= <add operator>

<constant> ::= <text> | <number>

<text> ::= " <characters sequence> "

<characters sequence> ::= <character> | <characters sequence> <character>

<character> ::= <digit> | <letter> | <capital> |
" | < | ; | : | ' | > | * | (|) | _ | + | ! |
- | & | $ | # | , | . | ¯ | ? | / | <stroke>

<capital> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N |
O | P | Q | R | S | T | U | V | W | X | Y | Z

COMPUTER AIDED PROGRAM EXPLANATION (CAPE)

J.M. Bennett and C.S. Allison

1. INTRODUCTION

The documenting of large and complex programming systems is one of the most significant problems facing the computer industry today. Its economic importance increases with applications of increasing sophistication. Moreover, as industry, commerce and government become more and more dependent on computers, the dangers of major dislocation inherent in software systems, the details of which are often not properly understood by those charged with maintaining them, continue to mount.

The situation no doubt is very similar to that which existed in the early days of the evolution of engineering as a profession. So long as a controlling designer could supervise all the details of construction of a building or a machine himself, only rough sketches or models were necessary. However, with new materials and techniques, it became necessary to plan more complicated structures and machines.

As a result, the practice arose of separating design and specification from construction either by an independent contractor or by a separate section of the same organisation. The technical difficulties of informing all parties of what was required of them, together with questions of legal responsibilities inherent in the use of the joint efforts of a number of separate organisations, led to increasing emphasis on full documentation. As a result, the drawing office to-day plays an essential part in any large engineering concern, both in making possible the completion of large undertakings and in providing a repository for information necessary for maintenance and later modification.

The early fifties were the heyday of the individualistic programmer, and the attitude that proper preliminary specification, flow charting and documentation is a waste of time is still all too prevalent, particularly among scientific workers. Scientific workers can often justify such an attitude when their programs are small and of usefulness which terminates with the completion of the immediate experimental projects for which they are designed. Moreover, the brightest members of this group have accomplished some prodigious feats of unaided system building – though they tend to be a great deal less enthusiastic when it comes to writing up their work and subsequent 'tinkering'. Even to-day, there are eminent proponents of the "shock-troop" approach (MICHAELSON [1968]) – the approach of handing over the hardware to a small team of brilliant amateurs who, by living laborious days (and nights), can come up with a better system more cheaply than their more pedestrian colleagues in large industrial software groups.

However, it is clear that an established industry cannot be dependent on small groups of prima donnas. And so, inevitably, industrial groups are becoming structured and formalised, and are observing standards of specification, documentation and test (HUNT [1967]) which are more appropriate to a mature industry. Thus preliminary and acceptance test specifications are no longer written by the same group which plans programming strategy for a software system, and yet another group constructs and "factory tests" individual sections of the system and fits them together.

Under such an arrangement, much of the problem of documentation subsequent to completion, is avoided, as these documents have become an essential part of handing over information from one group to the next.

However, it is at this point that the industry finds itself on the horns of a dilemma. For the better the documentation, the more formidable is the pile which the newcomer must go through to become acquainted with the details of a system. In fact, many reputations have been built on an individual's ability to master this or that system, to know its idiosyncracies (which in practice means those features which have never been properly debugged or which have been ambiguously written up), and even in some cases to make small changes to it despite the obscurantist virtuosity of its author.

Attempts have been made to enlist the aid of computers. Automatic flow charters (e.g. MAEKAWA [1965] and ROBERTS [1969] and other references listed at the end of the article) are becoming common, and there are even some organisations offering to produce such flow charts commercially. Computer assistance of this type is invaluable in cases in which flowcharting was not done properly in the first place, and in fact, without a flow chart (with break-downs of individual sections if necessary) showing all decisions and processes, group discussion of a complicated system is well nigh impossible.

This paper is concerned with an attempt to invoke computer aid, not as a documenter, but to provide a guided tour of individual sections of a program. The proposal is based on the idea that it is unnecessary to have to understand the whole of a program in order to tinker with a part of it. What is needed is a means of putting the program through the type of operation we wish to modify and to have it describe what it is doing.

First, it would do so in approximate terms only.

Then, more detail of individual sections would be supplied on request, until finally the appropriate section of code or controlling parameter is pin-pointed.

The scheme (to which the acronym CAPE is given) assumes the copious use of the comment facility permitted (and ignored) by most compilers and assemblers. It is discussed in terms of the English Electric KDF 9

USERCODE: however, no special features of this assembly language are invoked.

2. DESCRIPTION OF CAPE

The process makes use of a monitor (HADDON [1966]) which is a modification of one supplied by English Electric (ANON [1964]). This monitor normally provides printout of specified registers at a number of test points. Moreover it is possible to use it between stated limits, and for several depths of testing. With CAPE, test points become check points.

Monitors of this type are of course not new, and are normally used as diagnostic aids. In fact, one of the authors (J.M.B.) has designed two such schemes (BENNETT [1953] and BENNETT et al. 1957). The second, which was intended to use a minimum of storage space, produced a paper tape output which was subsequently re-input for suitable conversions and formating a similar arrangement to that outlined here.

The proposed scheme intercepts the output of the monitor and stores it on magnetic tape for subsequent processing. The processing takes the form of integrating the output with the original text (which includes details of check points.) The integration consists of locating check points and printing out accompanying comments and the contents of registers which form part of the output of the monitor program.

So that this output can serve as a guide to fuller documentation, comments include references to a flow chart. In the case of the example discussed below, the flow chart has been computer-drawn by a line printer using a standard program (WAITE [1966]), and each "unit" operation shown in the chart has coordinate values associated with it. These coordinate values, with corresponding page numbers, are included in comments for ease of cross reference. The output also includes check point references. These check point references are used in requesting information in greater depth between quoted check points.

As the monitor provides printout of designated registers in five forms, the comments also include a specification of the form in which the register contents are required. This specification is suppressed in the output, but is used by the section of CAPE which integrates comments from the text with monitor output in order to select the final form of output most suitable for didactic purposes.

The monitor routine includes a facility which allows a path between two stated check points to be monitored a given number of times, register information being output on each occasion. An alternative, which can be used at the same time if required, outputs trace information giving the course taken by control, described in terms of check point numbers.

When these facilities are used for explanation, comments are printed out the first time individual check points are encountered and suppressed thereafter, using the alternative facility, modified to include a reference to the number of times the check point has been encountered. However, while the facility is still "turned-on", specified register contents are output when check points are encountered on subsequent occasions. Trace information will also be given within the specified limits.

The scheme has been extended so that, as simple loops with up to five check points are detected, trace information is not repeated. Instead, details of check points encountered during the first and last execution of a loop are given, together with the number of times the loop is repeated.

Ideally, the quest for increasing depth of explanation would take the form of man-machine interaction using a versatile display and a light pen to indicate for example, test point limits for requirements for explanations in greater depth.
Full facilities for making this possible were not available at the time the work described here was planned, and so the preliminary trials were carried out in the batch processing mode. The decision about whether to extend the plan to provide immediate interactive facilities will be based on experience, and will not call for a great amount of additional programming.

3. PREPARATION OF COMMENTS

The use of the technique requires the careful preliminary preparation of comments (in English) and the placement of check points. The scheme does not require the use of any "documentation" language (except perhaps references to a parallel set of flow charts if such a set is available); the availability of comments placed next to the check points in the original text is all that is needed. However, in a situation in which the strategy of a program is planned by a group so that coding can be carried out elsewhere, sufficient detail must be set down to describe what is to be done. As this information should provide at least a basis for explanatory comments, the additional labour should be small.

The selection of check points is crucial to the success of the scheme, as is the amount of detail required at each point. Fortunately, the same considerations apply to the choice of check points as to the selection of test points for diagnostic purposes when the program is first written.

Thus check points should appear at least

(1) after each transfer of control,
(2) on entering a subroutine,
(3) on leaving a subroutine, and
(4) after each significant step of a calculation.

The levels assigned to each check point and the information requested would also be expected to correspond to the levels one might assign to test points. These would be chosen so as to correspond to the various levels of coarseness which flow diagrams used for planning would represent. Thus the first (highest) level of test point would correspond to points at the ends of major blocks in an overall flow diagram, the second level to strategic points in diagrams showing the first breakdown of these blocks, and so on.

So the proposed technique should require very little work over and above what should have been carried out as preparatory planning in any case. In program testing, care must be exercised when logical errors in the overall structure are discovered to ensure that comments and check/test points numbering are adjusted accordingly. However, the discipline required to ensure this adjustment is necessary if the comment feature is to be helpful for the purpose for which it is normally used – viz., for understanding program text presented in its entirety.

To reduce the length of the comments, a glossary should be available. Such a glossary would set out the meaning of terms used in describing counters, switches, etc.: if it is not provided, the meanings of these terms may

have to be set out several times. For even if they are defined in comments when they first appear, there is no guarantee that the course of control for the particular test example being examined will bring defining comments into print.

4. WHAT CONSTITUTES A GOOD EXPLANATION

BERKELEY [1967] has recently examined the question of computer-assisted explanation in general, and has some apposite comments which are as applicable to the problem of conveying information about computer programs as to the more general examples with which his book is concerned.

He points out that explanation, though part of an exposition, is a great deal more. It is a deliberate attempt to make "some previously obscure topic clear and plain and understood".

A successful explanation is one which conveys the information it is intended to convey to someone who is not strongly motivated to understand.

This point is particularly significant in the computer field, where many users with some computer background often have a choice between making minor changes to an existing program and writing a new one. When faced with a so-called documented program which consists of program text plus a sparse peppering of obscurantist annotations, anyone faced with such a choice will usually prefer to begin again.

This is particularly true when the program concerned is designed by its author as a demonstration of coding virtuosity, and when it is known to contain one or two obscure bugs which manifest themselves under conditions known only to the cognoscenti. A little thought given to explanatory aspects of a program is likely to increase its useful life considerably.

5. AN EXAMPLE

The rest of the paper will be concerned with an example of the application of CAPE. The example comes from a program with which CAPE was initially tested. The program chosen is a KDF 9 program entitled "PRINT ANYTHING" (BENNETT and BLATT [1966]), the purpose of which is "to enable a user to obtain a printup of 8-channel tape punched in flexowriter or in ASCII code, and 5-channel tapes punched in either SILLIAC code or Telex code on the line printer". The details of the non-standard codes are not relevant to the purpose of this article.

The method of operating is simple. The paper tape concerned is input. "Codes are differentiated by leading runout, and no operator's action is required. Any mixture of 5-channel or 8-channel may be printed in one run." Output consists of a series of line pairs on the line printer, each pair of lines representing what would have appeared as a single line on the device producing the punched paper tape. One vertical character pair represents one input character. Apart from the leader characters, there is some representation for every character on tape and for the effects of such facilities as case shift, underlining and tabulate.

The tabulate facility has been set initially so that the printed output has the appearance it would have with tabs set every eight printable characters.

The documentation available consists of a users' description (setting out code equivalents, etc.) and a computer-drawn flow chart. The program, though a small one, is sufficiently complicated to be regarded as non-trivial. There are, for example, fifteen switches, five involving more than two alternatives. Comments include flow chart reference numbers, subroutine reference numbers, decision point reference numbers and labels designating the destination of transfers of control.

Thus, in the first comment given below C116 FLOWCHART REFERENCE AI B2-G3 PATH R 67-41-24-32 means that the course of control between its predecessor and C116 is through the labels with reference numbers 67 (R67), 41, 24 and 32, and is shown on page AI of the flow chart between references B2 and G3. The check point occurs at R32 which is shown at reference G3 on page AI of the flow chart. Where the course of control through switches and decision points is undefined at the level selected, the labels selected are those common to all switch positions and decisions. To ensure that register output is easily understood, the first comment produced by CAPE is an explanation of the form of this output. Moreover, wherever the contents of registers are output, they are preceded by a short remark stating what they represent.

Much of the reference information is perhaps unnecessary for the explanation required. However, it has been included to enable a user to investigate more of the performance of the program under examination if he so desires.

Suppose we wish to determine how to alter the tab settings, which according to the write-up, are specified by a series of tab numbers: the scheme should therefore be used to track down the location of these tab numbers.

To find this information, a tab character is supplied as input, and output at level 1 (the coarsest level) is requested. For this example, the most relevant of the comments obtained, that concerning tab numbers, is shown below together with the register output preceded by an explanatory remark:

```
C116  FLOWCHART REFERENCE A1 B2-G3 PATH
      R67-41-24-32 TAB NUMBER * EXTRACTED.
      BRANCH TO SWITCH M* POSITION 1. SWITCH
      C* SET TO POSITION 2 AND SPACES IN-
      SERTED AS NEXT VERTICAL CHARACTER
      PAIR. TAB NUMBER IN N01
      N01        +8
```

Starred items are described in the glossary. The relevant entries are as follows:

TAB NUMBER. The position in a line at which the next character appears after a tab character has been received.

SWITCH M. This switch makes possible the changing of the tab counter*. When the tab number equals the flexowriter character counter*, it is changed from position 1 to position 2 so that the next time a tab character is encountered (even if it is the next character), the tab counter is advanced and switch M set to position 1.

SWITCH C. This switch enables the tab and line feed facilities to be handled. Its setting is changed to position 2 when a tab character is encountered, and to position 3 when a line feed character is encountered. These settings each close a loop which allows the required line (tab) or page (line feed) position to be attained. On completing the

operation, switch C is set to position 1 allowing the translation of the next character to begin.

Starred items referred to in the above entries are also contained in the glossary, the relevant entries being as follows:

TAB COUNTER. This refers to the position in a list of up to eight tab numbers giving successive tab settings. The list provided gives tab settings every eight characters, the last tab setting being character position 63.

FLEXOWRITER CHARACTER COUNTER. This refers to a running count of the number of characters which would have been output in a line had that line been output on a flexowriter.

Further details of the information we are seeking may be obtained by requesting output at the levels 2, 3 and 4 in succession, between the higher level check points concerned with the tab character. The check point limits required at any one level are easily determined using output at the level immediately above it. The comments (with register output deleted) obtained at these finer levels which concern our problem are as follows:

Level 2:

C216 FLOWCHART REFERENCE A1 B3-C4. PATH R41-24 TAB NUMBER* EXTRACTED AND FOUND TO BE NON-ZERO INDICATING THAT MORE TAB POSITIONS REMAIN. BRANCH ACCORDING TO THE SETTING OF SWITCH M*.

Level 3:

C1301 FLOWCHART REFERENCE A1 B3 PATH R41 APPROPRIATE TAB NUMBER* EXTRACTED FROM REGISTER V34 BY USING THE TAB COUNTER* TO INDICATE THE CHARACTER POSITION IT OCCUPIES.

Level 4:

The comments associated with level 4 check points do not differ from those of level 3 except that they are preceded by the actual USERCODE text which they describe. This provides an opportunity for the examination of the code together with a description of its operation. In this case, the code involves the extraction of the appropriate tab number from register V34, using the value of the tab counter as a pointer to its position as a six bit binary number packed in register V34.

With such information, our problem of determining how the tab settings could be altered, is resolved. The tab numbers have been found to be located in register V34 from which the appropriate one is extracted (as shown by the USERCODE text accompanying the level 4 check point) using the value of the tab counter at the time. To alter the tab settings then, it is merely necessary to adjust the contents of register V34 to suit our requirements.

6. CONCLUDING REMARKS

The CAPE scheme as outlined is an initial step towards the use of a computer to make documentation in the usual sense unnecessary. Although it has been demonstrated in the batch mode, provision of inter-active facilities will not require much additional effort.

The paper describes the mechanics of providing the explanatory facility. The question of its effectiveness depends on the way in which the explanatory matter is prepared, a remark which is true of all CAPE-type schemes. A realistic evaluation of its worth must await the accumulation of sufficient experience of its use in practice.

REFERENCES

ANONYMOUS, 1964, L1 - Monitor at Selected Test Points, English Electric Leo KDF 9 Service Routine Library Manual, vol. 2, English Electric Leo Computers Ltd., Kidsgrove, U.K.

BENNETT, J.M., 1953, Some Engineering Applications of Digital Computation, Doctoral Dissertation, Cambridge University.

BENNETT, J.M., J.C. BUTCHER and M. CHAPPLE, 1957, A New Diagnostic Routine, Proceedings of Conference on Data Processing and Automatic Computing Machines, Commonwealth of Australia, Department of Supply, Weapons Research Establishment, Salisbury, South Australia.

BERKELEY, E.C., 1967, Computer Assisted Explanation, Information International Inc., Cambridge, Mass.

HADDON, B.K., 1966, Monitor (Basser Computing Department Version), Basser Computing Department, University of Sydney, Sydney, Australia.

HUNT, P.M., 1968, The Production of Computer Software, Computer Weekly, nos. 83, 84, April 18, 25, 1968.

MAEKAWA, M., 1968, Automatic Flowcharting, Information Processing in Japan, Vol. 8, p. 72.

MICHAELSON, S.A., 1968, How to Succeed in Software?, Proceedings IFIP Congress 68, North Holland Publishing Co., Amsterdam.

ROBERTS, K.V., 1967, The readability of Computer Programs, The Computer Bulletin, vol. 10, no. 4, p. 17.

WAITE, W.M., 1966, K-Autochart, Basser Computing Department, University of Sydney, Sydney, Australia.

TEACHING COBOL WITH A MINIMUM OF PERIODS

A. Berger

1. INTRODUCTION

In the schools of many countries, computing is becoming a subject of its own, with about 2 periods a week. If this subject extends over one or two years, it is not easy to decide which chapter should be stressed and which parts of the big field of computer science should be omitted.

But everyone will agree that at least one high-level programming-language must be taught. In such a short time as indicated above, it is impossible to teach the pupils thoroughly, that means with the result that they are perfect programmers after having passed the course. It is important to teach them in such a way that:

1) they are able to program a complete problem however small it may be;
2) that they get acquainted with the principal structure of a high-level language.

To reach this aim it is our proposal to start not with a very simple but with a medium-sized model problem. After having drawn the flow-chart, the teacher reads the program of the model problem with the pupils and explains the position at that moment (no special features of COBOL, just the general meaning of the different sentences). So the pupils get a first impression of the COBOL structure.

Now there are two different ways of continuing the course, viz., by covering the topics of 2.2 or 2.5. A third possibility is to mix them up in a proper way.

The next chapter is a sort of small handbook which is in the hands of the pupil. After each subsection a basic problem should be discussed with and coded by the pupils.

2. COBOL 1440

2.1 *Structural Principles and Basic Instructions*

Each COBOL program is of uniformly general structure. A great number of cards remain unchanged; many others have a strictly prescribed form and only one single word (a field name) may be changed at random.

Let us therefore start from a sample problem.

Such cards as remain unchanged are characterized by an exclamation mark in this description. If there is one word underlined on such a card this means that the name concerned can be chosen at random.

The punching of the cards is done beginning with column 8 and, partially, column 12. The model problem shows how the cards have to be punched.

Each program falls into two main parts:

a. Control cards and general description of data file and fields necessary; and
b. the program proper, beginning with the card PROCEDURE DIVISION (in the model problem characterized by two exclamation marks). Individual statements are always punched beginning with column 12. In case of branching, however, statements can never be addressed directly but so-called paragraph names must be used, which must precede the statement concerned and which are punched beginning with column 8 (cp. model problem).

The first 6 columns of the punch card may be used for continuous numbering of the cards. This numbering is done in the following way:

001010
001020
001030 and so on. The first 3 figures indicate the page of the code sheet.

In the model problem, the numbering has been omitted. What is very important is making full stops in the right places; after each statement or paragraph name a stop must be made. There are, however, some exceptions (e.g., in the FILE SECTION, cp. model problem).

2.2 *Let us not discuss in detail those cards which are not compulsory, i.e., which contain names that can be chosen at random.*

2.2.1 PROGRAM-ID. „Program identification".

The program identification must not be longer than 10 characters. One or several cards with remarks may follow. The first card must read:

REMARKS ..... further text .....

Other cards with certain remarks - which, however, are without any influence on the program - may stand in this division.

2.2.2 In the SPECIAL-NAMES paragraph, the various machine units may be allotted random names.

Example: 1447-CP IS CONSOLE

This line produces the effect that the console inquiry station may always be referred to as CONSOLE. To the left of the word IS the standard names of the machine units must be inserted. These are:

1442-R, I	.....	Card reader
1442-P, I	.....	punching unit
1443-P	.....	printer
1447-CP	.....	console inquiry station
1443-CT, n	.....	carriage tape channel n

Switches, too, may be characterized by symbolic names, e.g.

1443-P-CV ON STATUS IS SHEETEND.

To the left of the standardized word group ON STATUS IS there appears the standardized name of the indicator or switch. Such names are:

1443-P-C9	.....	a hole in channel 9 of carriage tape has been reached
1443-P-CV	.....	a hole in channel 12 of carriage tape has been reached
1440-SS X	.....	switch X is on (X may be A to G).

2.2.3 In the FILE-CONTROL paragraph the files are described, i.e., they are given names. Actually the names are taken at random but we will always use the names CARDS, PRINTER, PUNCHER and DISK for those files.

According to which files are to be used, several of the following cards must be taken:

SELECT CARDS ASSIGN TO 1442-R, 1
SELECT PRINTER ASSIGN TO 1443-P alternatively
SELECT PUNCHER ASSIGN TO 1442-P,1
SELECT DISK ASSIGN TO 1311-D, '

2.2.4 In the FILE Section paragraph the files and the corresponding input/output areas must now be described. This may be illustrated by the following example:

FILE SECTION.
FD CARDS
LABEL RECORDS ARE OMITTED
DATA RECORD IS GETFIELD
01 GETFIELD
02 CNR PICTURE 9(5).
02 NUMBER PICTURE S9(8).
02 FILLER PICTURE X(67).

Explanation:

FILE SECTION.
This card must be at the beginning of this section.

FD Name of file (here CARDS)
This card must correspond with the FILE-CONTROL name.

LABEL RECORDS ARE OMITTED
This card indicates that no labels are used.

DATA RECORD IS Name of input area (here GETFIELD)
This card indicates the name of the reading area for the first time.

Now the reading area will be described in detail:

01 GETFIELD.
The name of the whole reading area will be repeated on level 01. This means that by later entries on level 02 no new fields are defined, but that the area of level 01 will be subdivided. In our example, therefore, the area GETFIELD is subdivided into several fields:

02 CNR PICTURE 9(5).
CNR is the name of the field. The PICTURE entry indicates how many and which storage locations this field is to have (more details about PICTURE will be given later). 9(5) means that 5 storage locations will be reserved for figures.

02 NUMBER PICTURE S9(8).
In the reading area GETFIELD the field CNR is immediately followed by the field NUMBER. Here the PICTURE entry means the field to be of length 8, numeric and bearing an algebraic sign.

02 FILLER PICTURE X(67).
The word FILLER has not been chosen at random but means space or interval. In this case, therefore, the remaining 67 places of the reading area are vacant, i.e., filled with blanks.

2.2.5 In the WORKING STORAGE SECTION the remaining calculation fields and constants are described as follows:

Example: 77 BALANCE PICTURE S9(17) VALUE ZERO

Explanation: These fields always have level 77 (a characteristic feature of COBOL).s,
BALANCE is the field name.
PICTURE S9(17) means that a numerical field of length 17 with algebraic sign is concerned.
VALUE ZERO means that at the beginning the field BALANCE is to be filled up with zeros.

2.3 *The PICTURE entry*

Size and type of a field are determined by the PICTURE entry. There are the following types:

PICTURE 9(10)	.....	length 10 numerical field
PICTURE X(10)	.....	length 10 alphanumerical field
PICTURE A(10)	.....	length 10 alpha field

Instead of the bracket entry it would also be possible to repeat the field symbol several times.

Example: PICTURE 999999 or PICTURE 9(6) means exactly the same.

If the numerical symbol 9 is preceded by an S, this means that in the field concerned the algebraic sign is being considered.

Example: PICTURE S999 or PICTURE S9(3) means a 3-digit numerical field with a sign.

The position of a comma in a numerical field may be indicated by inserting a „V".

Example: PICTURE 999V99 is a 5-digit numerical field with 2 decimals.
PICTURE S9(5)V99 is a 7-digit numerical field with 2 decimals and sign.

Masks and constants are also defined by PICTURE entries. For masks the following symbols are available:

Comma and full stop will remain unchanged in numerical fields.

9 positions for figures.
Z positions for figures in which, however, preceding zeros will be suppressed.
B symbol for true blanks in a numerical field.
* leading asterisks (comma or stop will also be replaced by a preceding protective asterisk).
– floating minus sign. The first valid figure is preceded by a minus symbol.
+ floating plus sign. The first valid figure is preceded by a plus sign.
CR (inserted after the number) will be retained or not if the number is positive or not.
+ (inserted after the number) will be retained or not if the number is positive or not.

For relevant examples, see enclosed exercise sheet EDITING (fig. 2).

2.4 *The VALUE entry*

Constants are defined by adding a VALUE entry to the PICTURE entry.

Example: PICTURE 9(7) VALUE ZERO means that this 7-digit numerical field is to be filled up with zeros.
PICTURE X(4) VALUE IS SPACES. This 4-figure alphanumerical field is filled with blanks at the beginning. The word IS may or may not be written after VALUE.
PICTURE A(5) VALUE „YEARS". In this 5-figure field the word YEARS is stored.
PICTURE 9(6)V99 VALUE 008826.69. In this 8-digit field the figure sequence 00882669 is stored; the comma is supposed to be inserted between the two sixes.

2.5 *The statements in COBOL*

The PROCEDURE DIVISION contains the statements proper.

2.5.1 At the beginning and the end there must stand OPEN or CLOSE statements.

Example: OPEN INPUT CARDS OUTPUT PRINTER.
CLOSE CARDS PRINTER.

2.5.2 Reading statements: READ name of file.

Example: READ CARDS

This statement may be supplemented in two ways:

a) READ CARDS INTO KEEPFIELD
Upon this, transfer is made immediately from the reading area to the preserving area named KEEPFIELD.

b) READ CARDS AT END GO TO .....
After the final condition branching is made to any paragraph desired.

2.5.3 The printing statement: WRITE name of file.

Example: WRITE PRINTER.

This statement may be supplemented in two ways:

a) WRITE PUTFIELD FROM KEEPFIELD.
Transfer from a preserving area (KEEPFIELD) to the printing area is carried out and then printing is done.

b) WRITE PUTFIELD AFTER (BEFORE) ADVANCING 4 LINES.
Here printing is done only after (before) a 4-line advance.
WRITE PUTFIELD AFTER (BEFORE) ADVANCING NEW-PAGE.
If, in SPECIAL-NAMES, NEW-PAGE is equalled 1443-CT, 1, printing is only after/before advance to channel 1.

2.5.4 Moving and caculating statements:

Field names are indicated by small letters, COBOL statements by capital letters.

a) MOVE fielda TO fieldb
b) ADD fielda TO fieldb
c) SUBSTRACT fielda FROM fieldb (GIVING fieldc)
d) MULTIPLY fielda BY fieldb (GIVING fieldc)
e) DIVIDE fielda INTO fieldb (GIVING fieldc)
f) COMPUTE fielda = algebraic expression.

An algebraic expression may contain + – / * and **.
The sequence of calculations may be indicated by brackets.

2.5.5 Unconditional branching: GO TO name paragraph.

2.5.6 Conditional branching: IF condition instruction (ELSE instruction)

The following conditions may occur:

fielda GREATER fieldb	fielda NOT GREATER fieldb
fielda LESS fieldb	fielda NOT LESS fieldb
fielda EQUAL fieldb	fielda NOT EQUAL fieldb

Example: IF fielda GREATER fieldb GO TO PRINT ELSE MOVE SPACES TO fielda.

In connection with IF, the two logical operators AND and OR may be used.

Example: IF fielda = Ø AND fieldb = Ø OR fieldc GREATER 12 GO TO ..... etc.

The AND operator is more powerful than the OR operator. In order to secure a certain sequence, brackets may be used.

2.5.7 The PERFORM statement:

PERFORM paragraph name has the effect that branching is done into a subroutine. Return is effected automatically. In case the subroutine consists of several paragraphs, the first and the last paragraph name must be indicated in the following way:

PERFORM first paragraph name THRU last paragraph name.

2.5.8 The statements OCCURS and VARYING (builtup of multidimensional charts).

Let us start form an example:

We have 7 types of cost. Each of the 7 types of cost may occur within each one of 11 accounts. Therefore we need for differentiation purposes 77 different fields which are defined as follows:

```
01 FIELDS
    02 BER OCCURS 7.
        03 SUM OCCURS 11 PICTURE S9(2)V99.
```

The new feature of this field definition is the statement OCCURS. This statement results in the field being stored so often side by side as is indicated by the number to the right of the OCCURS statement. The total area of the 77 fields is given the name FIELDS in level 01. This total area is divided into 7 zones called BER. Each of these 7 zones is subdivided into 11 fields named SUM. Thus we have 77 fields with the name SUM. In order to address these fields unequivocally, the field SUM must be indexed. This is done by writing the 2 indices in brackets beside the word SUM.

Example: MOVE NUMBER TO SUM (3,8)

By this command the 8th SUM field in the third BER area is addressed. If all 77 fields are to be processed in the same way (e.g. store values, check values, etc.), an interesting PERFORM statement is at our disposal for this purpose. This statement reads (with regard to our example):

PERFORM SUBROUT VARYING X FROM 1 BY 1 UNTIL
X GREATER 7 AFTER Y FROM 1 BY 1 UNTIL Y GREATER 11.

In this command, SUBROUT is a random paragraph name for the subroutine. X and Y are random names for the indices. The figures 1 mean the beginning and the pace length of the changes. 7 and 11 are the top limits. In the subroutine itself the field name SUM must be indicated by X and Y.

Example: SUBROUT.
MOVE NUMBER TO SUM (X,Y).

Fig. 1 MODEL PROBLEM

```
!   IDENTIFICATION DIVISION.
!   PROGRAM-ID. 'DEB-AND-CRED'
    REMARKS. THIS IS A MODEL PROBLEM.
!   ENVIRONMENT DIVISION.
!   CONFIGURATION SECTION.
!   SOURCE-COMPUTER. IBM-1440.
!   OBJECT-COMPUTER. IBM-1440 MEMORY SIZE 4000 CHARACTERS
!       NO-PRINT-STORAGE
!       NO-MULTIPLY-DIVIDE.
    SPECIAL-NAMES.
        1447-OP IS CONSOLE
        1443-OT, 1 IS NEW-PAGE.
        1443-P-CV ON STATUS IS SHEETEND.
!   INPUT-OUTPUT SECTION.
!   FILE-CONTROL.
!       SELECT CARDS ASSIGN TO 1442-R, 1
!       SELECT PRINTER ASSIGN TO 1443-P
!       RESERVE NO ALTERNATIVE AREA.
!   DATA DIVISION.
!   FILE SECTION.
!   FD CARDS
!       LABEL RECORDS ARE OMITTED
!       DATA RECORD IS GETFIELD.
        02 CNR PICTURE 9(5).
        02 NUMBER PICTURE S9(8).
        02 FILLER PICTURE X(67).
!   FD  PRINTER
!       LABEL RECORD OMITTED
!       DATA RECORD PUTFIELD.
!   01  PUTFIELD.
        02 ZCNR PICTURE 9(5).
        02 FILLER PICTURE X.
        02 DEBIT PICTURE ZZBZZZBZZ9.99.
        02 CREDIT PICTURE BZZBZZZBZZ9.99.
        02 BALANCE PICTURE BZZBZZZBZZ9.99.
        02 SIGN PICTURE XX.
!   WORKING-STORAGE SECTION.
    77 ZBALANCE PICTURE 9(8)V99.
    77 ZCNR PICTURE 9(5).
    77 SDEBIT PICTURE 9(8)V99 VALUE ZERO.
    77 SCREDIT PICTURE 99(8)V99 VALUE ZERO.
    77 SSDEBIT PICTURE 9(8)V99 VALUE ZERO.
    77 SSCREDIT PICTURE S9(8)V99 VALUE ZERO.
!!  PROCEDURE DIVISION.
!   START.
!       OPEN INPUT CARDS OUTPUT PRINTER.
        MOVE SPACES TO GETFIELD PUTFIELD.
        READ CARDS AT END GO TO END
    ANFANG.
        MOVE CNR TO ZCNR DCNR.
    LOOP.
        IF NUMBER GREATER 0 MOVE NUMBER TO DEBIT ADD NUMBER TO
        SDEBIT ELSE MOVE NUMBER TO CREDIT ADD NUMBER TO SCREDIT.
        WRITE PUTFIELD.
        MOVE SPACES TO PUTFIELD.
        READ CARDS AT END GO TO END
        IF CNR = ZCNR GO TO LOOP.
        IF CNR LESS ZCNR DISPLAY 'ERROR' UPON CONSOLE.
    UP.
        MOVE SDEBIT TO DEBIT.
        MOVE SDEBIT TO CREDIT
        COMPUTE ZBALANCE = SDEBIT & SCREDIT.
        IF ZBALANCE LESS 0 MOVE 'CR' TO SIGN.
        MOVE ZBALANCE TO BALANCE.
        IF SHEETEND WRITE PUTFIELD AFTER ADVANCING NEW-PAGE
        ELSE WRITE PUTFIELD BEFORE ADVANCING 1 LINES.
        MOVE SPACES TO PUTFIELD.
        ADD SDEBIT TO SSDEBIT.
        ADD SCREDIT TO SSCREDIT.
        MOVE ZEROS TO SDEBIT SCREDIT.
    FIN.
        GO TO ANFANG.
    ENDE.
        PERFORM UP.
        MOVE SSDEBIT TO DEBIT.
        MOVE SSCREDIT TO CREDIT.
        COMPUTE ZBALANCE LESS 0 MOVE 'CR' TO SIGN.
        MOVE ZBALANCE TO BALANCE.
        WRITE PUTFIELD AFTER ADVANCING 1 LINES.
!       CLOSE CARDS PRINTER. STOP RUN.
```

Fig. 2

EDITING

Transmitting Field		Receiving Field	
PICTURE	data	PICTURE	report
99999	12345	ZZ,ZZ9.99	12,345.00
99999V	00123	ZZ,ZZ9.99	b123.00
9(5)	00100	ZZ,ZZ9.99	100.00
9(5)V	00000	ZZ,ZZ9.99	0.00
9(5)	00000	ZZ,ZZZ.99	.00
9(5)	00000	ZZ,ZZZ.ZZ	blanks
999V99	12345	ZZ,ZZ9.99	123.45
V99999	12345	ZZ,ZZ9.99	0.12
9(5)	12345	**,**9.99	12,345.00
9(5)	00123	**,**9.99	***123.00
9(5)	00000	**,***.99	******.00
99V999	12345	**,**9.99	****12.34
V99999	12345	**,**9.99	*****0.12
S99999V	12345–	–ZZZZ9.99	–12345.00
S9(5)V	12345+	–ZZZZ9.99	12345.00
S9(5)	00123–	–ZZZZ9.99	–bb123.00
S99999	12345+	ZZZZ9.99	12345.00
S9(5)	00123+	––––––.99	123.00
S9(5)	00001–	––––––.99	–1.00
S9(5)	12345–	ZZZZ9.99CR	12345.00CR
S9(5)	00123+	+++++.99	+123.00
S9(5)	12345+	ZZZZ9.99CR	12345.00
S9(5)	12345–	ZZZZZ.99+	12345.00–

USER-ORIENTED LANGUAGES FOR SELF-EDUCATION

Lea M. Bohnert

1. INTRODUCTION OF PROBLEM

In the spring of 1968, a project was undertaken at the School of Library Science at the State University of New York at Albany, SUNYA, to investigate the feasibility of teaching one of the new user-oriented computer languages as an introduction to learning how to communicate effectively with computing personnel.

Most people without data processing experience, such as librarians, have considerable difficulty in describing to computer personnel the work they want to have done. They also have difficulty in determining the kind of work appropriate to delegate to computing facilities. Both of these difficulties are linguistic, and so, require linguistic analysis and linguistic solutions.

People are accustomed to using natural language with all its richness and versatility, ambiguity and contradiction, for everyday purposes. Most people employ natural language for delegation of work to other human beings. This is efficient as long as the people know each other and can communicate directly with each other whenever they wish. It is best, of course, when the work to be delegated is obviously simple, or when it is so complex that not much needs to be, or can be, communicated.

Languages for data processing have been modeled on the languages of mathematics. Compared with natural languages, mathematical languages are rigid and limited. One can only deal with things or processes that can be specified unambiguously. Yet, in many concerns, human beings have learned to value the indirectness and ambiguity of natural languages.

Persons who are primarily literate, such as librarians, lawyers and teachers are certain to have communication difficulties when trying to talk to individuals who are primarily numerate, such as statisticians and computer scientists. Computer scientists have the difficult job of translating from an ambiguous natural language, such as English, into data processing languages. Computer scientists can be substantially helped by knowledgeable customers who themselves are able to translate their work into data processing terms.

Sometimes in exasperation, data processing consultants make the rude remark, 'The customer does not know what he wants done.' It would be closer to the truth to say that 'The customer does not know how to state in data processing terminology the job he wants done.' When the statement is so rephrased, the responsibility of the computer scientist becomes clear. Computing scientists should develop and provide a variety of methods to assist the users in specifying the work to be done. Such specification aids, like flow charts, help to guide decisions as to the kinds of jobs that are appropriate for data processing solution. At the same time, they assist in the choice of techniques for performing the job.

Most professionals have the responsibility of delegating the simpler tasks in their own field to other workers, or when appropriate, to machines. The simplest kind of tasks to delegate are 'clerical routines,' where an explicit and limited set of rules is to be followed repetitively. Of course, not every task called a 'clerical routine' has the necessary simplicity that permits precise description. A good example of a 'clerical routine' that is more complex than it at first appears to be is the alphabetization of a list of proper names.

When one delegates clerical routines, either to a person or to a machine, it is advisable to **prescribe** the steps in the work to be performed as well as to **describe** the results wanted. Prescription of explicit steps becomes a necessity when the machinery operates only according to formal instructions of a limited kind. This is the case with computing equipment. This kind of explicit prescription has not been part of the training of many people, including librarians.

2. ALTERNATIVE METHODS OF SELF-EDUCATION

As in most new fields or disciplines, the main way for an outsider to discover the capabilities of computer facilities has been to become an 'insider'. One learns how to use the facilities at one particular place. This requires considerable aptitude. Until recently it was a time-consuming activity.

There are a constantly increasing number of textbooks and how-to-do-it manuals that testify to the keen desire of computer experts to communicate the accomplishments and the possibilities of the new field. There are an increasing number of good case studies of data processing applications in particular fields, which are presented in the languages of the fields. One can discriminate the better case studies by the small amount of description devoted to data processing equipment or techniques, and the large amounts of description devoted to the application being made. A good example of this in the library field is the book, The Computer and The Library, by N. S. M. Cox, JD. Dews, and J. L. Dolby, Northumberland Press, England, 1966 and Shoe String Press, 1967.

Finally, one can hire a consultant, an expert in the techniques of data processing and with some skill in problem solving. Because one is delegating to another the responsibility of translating one's particular requirements presented in natural language into the language of a particular computing facility, it is preferable that the consultant has some understanding of the field in which the problems are to be solved.

3. NEW POSSIBILITIES

After twenty years, experts in the data processing field are beginning to be able to describe, in languages other than computing languages, the work they have done. Also, the logic and the semantics of computer languages are beginning to be generalized, simplified, and standardized. In fact, some languages are being developed to facilitate direct experimentation with computers by non-experts. These new kinds of languages boast hyphenated names, such as 'user-oriented', 'problem-oriented', or 'higher-level' languages.

In 1969, a monumental compilation of these kinds of higher level or programming languages was published. The author, Miss Jean E. Sammet, states:

> 'The most well known language (FORTRAN) is merely one of approximately 120 languages described in this book. (Of this total, approximately 20 are completely dead or on obsolete computers, about 35 are receiving very little usage, about 50 are for specialized application areas, and about 15 are widely used and/or implemented.)'
> SAMMET (1969)

The development of effective user-oriented languages was not possible until substantial progress had been made in the following three areas:

1. The proof by computer scientists that computers were 'general purpose' machines by creating machine independent languages for controlling computers. In 1958-1960, international committees of data processing experts developed ALGOL, ALGOrithmic language for general use.
2. The experience gained in exchanging programs when computer language and equipment were nearly the same. In 1960 in the United States, programmers began to exchange programs that were written in FORTRAN, FORmula TRANslation, language. These were usuable on most of the large IBM computers.
3. The development of simple and inexpensive ways for using computers by people with or without data processing expertise. This has now been accomplished by use of time-sharing terminals that can be located anywhere, so long as a connection with a telephone is possible.

4. REQUIREMENTS FOR SELF-EDUCATION

Like most other professional schools, the SUNYA School of Library Science desired to develop a good way to introduce graduate students to the proper use of computing services. We wished to introduce our students **in terms of library work.**

First, we decided to use only those computer languages that were alphabetically oriented because we wanted to counteract the prevalent myth that computers can handle only numbers. Furthermore, librarians mainly serve by providing easy and frequent access to primarily alphabetic materials.

Second, we required that any computer language chosen should be easy to learn by people who had no previous data processing experience or aptitude. Still, the language had to be powerful enough to be able to manipulate both alphabetic and numeric data. Fortunately, the two requirements turned out to be interdependent. A language easy for librarians to learn was also necessarily a language capable of manipulating strings of alphabetic characters in ways suitable for library work.

Third, we decided to select computer languages in which the names or codes, whether for the instructions or for the data to be manipulated by the instructions, would either be identical to, or be related mnemonically to the familiar words in the English language. Of course, the grammar or logical structure of the language should be simple and consistent.

Fourth, we wanted the introduction to the new technology to be economical and simple, so that our library students could experiment and explore the new tools as much as they wanted. We selected only off-the-shelf equipment, and chose among the operational computer languages developed for time-sharing use.

Finally, we wanted to have the support and guidance of the local computing personnel. In our case, we were most fortunate in the excellent help we received from the SUNYA Computing Center. They have doublechecked our choices of software and hardware.

5. CHOICE OF ADVISORS

We had several outside consultants to advise on the project as we had no data processing expertise in the SUNYA School of Library Science. First and foremost, we were able to secure the consulting services of Mr. Calvin N. Mooers, the creator in the early 1960s of TRAC, Text Reckoning And Compiling, language.* He had designed it as a '...general tool for dealing with text.' MOOERS (1965) Equally important, Mr. Mooers was a pioneer in the field of 'information retrieval', a term he had coined in 1950. MOOERS (1952). Therefore, he well understood the nature of library work. His guidance was crucial in the first two months in getting us started quickly and well.

We named our project, the Reactive Typewriter Project, because of another new term Mr. Mooers had created in 1961. The words 'reactive typewriter' were chosen to distinguish teletypewriters connected over telephone lines to computer services from those connected over telephone lines with other teletypewriters for purposes of long distance record communication. MOOERS (1963)

We also had the benefit of the advice of Mr. R. A. Fairthorne, a British information scientist, Dr. John W. Mauchly, one of the first computer scientists, and Mr. Joseph Hilsenrath, creator of the OMNITAB (not and acronym) language for scientists and statisticians. HILSENRATH (1966)

6. STUDENT ASSISTANCE

Dean John J. Farley of the SUNYA School of Library Science approved the payment of three graduate students to work on the Reactive Typewriter Project. The students were selected for their understanding of library work and their academic excellence. They were Mrs. Phyllis Betz Ochs, Mr. Jackson Davis and Mrs. Karen Brown Anable.

Mr. Edwin D. Reilly, Director of the SUNYA Computing Center had started in 1967 a partial(three hours each week day) time-sharing service. The service was named SUNYATSEN, SUNYA Time-Sharing Electronic Network. However, the only computer language available over SUNYATSEN was a CDC (Control Data Corporation) version of FORTRAN, FORmula TRANslation, which was not a suitable language for our purposes because it had been

developed primarily for numerical work. Mr. Reilly solved the problem by assigning graduate students from the SUNYA Computing Center to work on our project.

7. CHOICE OF FIRST COMPUTER LANGUAGES

We chose to experiment with two particular computer languages, because between the two of them they fulfilled our requirements for data processing languages for library applications.

We chose TRAC, Text Reckoning And Compiling, language, as the language that best met our requirements of ease of learning by alphabetically-oriented persons and for its capability of easily manipulating alphabetic and numeric data. Mr. Calvin N. Mooers of Rockford Research Institute, Inc., a non-profit organization located in Cambridge, Massachusetts, developed TRAC language for time-sharing use between 1963 and 1965. He arranged for us to use TRAC through the computing service of Bolt, Beranek and Newman, Inc., in Cambridge, Massachusetts, which was 150 miles away from Albany, New York.

Mr. Mooers had decided early in the development of TRAC language to facilitate its wide use by rigorously standardizing the language. His purpose was to ensure that any customer of any commercial computing facility which offered 'TRAC language' would be using exactly the same language. Consequently, the customers would not have to learn a new variation of the language each time they wished to use it at some other location. For this reason, if TRAC language is to be offered commercially, it must carry the TRAC trademark secured from the Rockford Research Institute, which indicates that the language offered meets the Rockford Research standards. MOOERS (1967 a, b, c, d,).

According to Mr. Mooers, the SUNYA School of Library Science project in 1968 was the first group to investigate the ease of learning TRAC language by adults who had no data processing training or aptitudes.

In 1967, some school children with mathematical aptitudes had experimented with TRAC as a self-education tool. The group named themselves R.E.S.I.S.T.O.R.S., The Radically Emphatic Students Interested in Science, Technology or Research Studies. They were:

> '...an informal group of boys and girls aged 11 through 16 who met regularly on weekends during the school year and more frequently during the summer in their old but sturdy computer-laden Red Barn outside of Hopewell, New Jersey, to teach one another computer skills and otherwise have fun.' R.E.S.I.S.T.O.R.S. (1969)

They became so enthusiastic about TRAC language that they were the first licensed users, for the large sum of $1.00.

8. CHOICE OF SECOND COMPUTER LANGUAGE

The second language of choice was BASIC, Beginner's All-purpose Symbolic Instruction Code, that had been developed at Dartmouth College in Hanover, New Hampshire. Its original purpose was to make possible student homework in mathematics and engineering from terminals on the campus. It was developed cooperatively by faculty and undergraduate students. KEMENY et al. (1968), KURTZ (1969)

A student can learn BASIC in an hour or two by sitting down and experimenting with it at any of the more than 70 campus terminals. BASIC was designed to have a clear logical structure and English-like codes. It has proved useful for many purposes, including manipulating alphabetic data.

In 1968, BASIC was the most widely available time-sharing language in the United States. It is still so today. Even IBM found it advantageous in 1969 to offer BASIC as a software option with their equipment. It is therefore highly probable that our graduates would be able to employ BASIC again in the future.

9. HARDWARE CHOICES

The choice of input equipment was not difficult because there was available only one inexpensive and simple keyboard. This was the Teletype Model 33 ASR teleprinter.

The keyboard is similar to that of the ubiquitous typewriter, except that only upper case alphabetic letters are available. This lack is unfortunate because librarians have to have high standards of readability since they provide rapid and frequent access to the same alphabetic materials.

The cost of a Teletype Model 33 terminal in Albany, New York, in 1968, was approximately $65.00 for installation, and the same amount for monthly rental. Connection to the computer facilities was over regular telephone lines using standard dialing methods. It was charged according to the regular station-to-station rates. By way of comparison, for a two hour demonstration of TRAC language, the telephone costs between Albany and Cambridge amounted to about twice the cost of the computer, approximately $40.00 compared to $26.00.

We rented two Teletype Terminals in order to be able to give demonstrations easily to classes of 30 to 40 students. We found we could accomodate groups of about 10 students for a 15 minute period around each terminal, for observation and 'hands on' experience with TRAC language.

We also rented a paper tape punch and a tape reader attached to each terminal. This ensured rapid and accurate input of typed material while we were connected to the computing facility. The cost was approximately $15.00 additional for installation and the same for monthly rental.

We found the punched paper tapes provided a good visual aid when explaining to students the codes that were being transmitted from the keyboard over the telephone lines to the computing facility. Also, paper tape was easy and inexpensive to store, or duplicate, and it could be used to produce additional printed copies of the demonstration materials.

10. LECTURE BEFORE THE TRAC LANGUAGE DEMONSTRATIONS

In his first introductory lecture, Mr. Mooers characterized a computing facility as merely another central utility or service. It should be thought of in the same way as one thinks of the heating plant of a university. Users expect such a central facility to accomodate with ease to the variety of demands that they choose to place on it.

Users of computers should not have to be concerned with the 'brand name' or the technical specifications of the particular equipment that they are utilizing. The computing service merely has to be able to assure the time-sharing user of two things. First, it must be able to assure them that the

equipment available to the service has the versatility required to handle the language preferred by the user. Second, it must be able to assure them that the service can provide economically the amount of storage space the user requires. Therefore, according to Mr. Mooers, the computer has finally become the 'remote computer', a reliable servant, distant and largely unknown to their time-sharing masters. The two computers we used were respectively 1000 feet and 150 miles distant from our terminals. Such distances had no effect on our operations.

Librarians are lucky because the kinds of manipulation of alphabetic data that they have to do are simple and basic since they are the common ways of handling records. Therefore, librarians usually can use general language, general programs, and general equipment. They do not have to suffer the cost or discomfort of developing specialized tools.

The kind of record that lends itself best to data processing operations is a record that can be designed and controlled by the user and that has easily identifiable parts that need to be searched or rearranged. Bibliographies or catalogs, whether in card or book format, are library examples of this kind of record. Librarians have mainly delayed utilizing computer services because of their limited capability of handling and printing alphabetic data. Now easy-to-learn computer language and simple-to-operate terminals compensate for these deficiencies.

Two main operations are performed on bibliographic entries by librarians, namely arrangement and searching. The matching of the descriptions of what is requested against descriptions of what is available is an easy operation for computer services.

To summarize, the main benefits of user-oriented languages for librarians, or users primarily interested in manipulating alphabetic data, are:

1. As an aid in identifying suitable tasks to delegate to computer services.
2. As an aid in the analysis of clerical tasks, whether delegated or not to computer services.
3. As an aid in learning how to gain access to and control of the increasing amounts of records being written in computer languages.

11. ACCOMPLISHMENTS OF THE REACTIVE TYPEWRITER PROJECT

The Total cost of the Reactive Typewriter Project to the SUNYA School of Library Science was nearly $47,000. This does not include payment for rent nor to any full-time staff, including the project director, Mrs. Bohnert. The project officially began in May, 1968, and ended in July, 1969. However, activities begun during the project were continued through the 1969-1970 academic year.

During the summer session of 1968, we gave class lectures and demonstrations to nearly 100 SLS students in three library science courses. Mr. Mooers gave the first lecture and demonstration of TRAC language, including 'hands-on' experience at the terminals for every student. Later two SLS students gave the demonstrations. They learned TRAC language in short sessions with Mr. Mooers and by experimentation at the terminals. The students were Mrs. Phyllis Betz Ochs and Mrs. Karen Brown Anable.

Only four TRAC instructions, out of a total of 34 available, were sufficient to give an exercise in a basic library task, namely the selection of bibliographic data to be stored and the determination of how it can be retrieved. The four instructions were:

RS, Read String
DS, Define String
CL, CalL
PS, Print String.

To bring TRAC language into operation over a time-sharing terminal requires;
(1) connection over telephone lines with a computer service than can provide a TRAC language facility,
(2) typing the code name of TRAC language wanted,
(3) typing the customer identification number assigned by the computer service.

To check that TRAC is operating correctly, a simple 'ECHO' test is used. The user types any string of characters and if the computer service immediately responds by typing the same string of characters, TRAC language is operating correctly.

To command a computer service to perform specific work;

The user types:	**Examples:**
1) a special code for the beginning of a command	#
2) The codes of the instructions being used.	RS
3) The data to be manipulated by each instruction	DS
a) when one datum or argument, the instruction codes, comma, and datum,	Ps, OKAY
b) when more than one datum, the instruction codes and commas before each datum.	DS, B, THE BIBLE
4) The codes for indicating scope of a command are parentheses enclosing the data and instruction code.	(DS, B, THE BIBLE)
5) a special code for end of a command.	'
6) Complete commands.	#(PS,OKAY)'
	#(DS, B, THE BIBLE)'
7) multiple or 'nested' commands.	#(DS, N1,#(RS))'

To make TRAC language even more English-like and simple to use, Mr. Mooers provided us with two TRAC programs.

The first program, named EASY, eliminated the necessity of the users having to type codes to indicate either (1) the beginning of commands or (2) the scope of commands. The EASY program automatically inserts at the appropriate places these codes. These codes are invisible to the user, since they are not typed out at the terminal keyboard. The four already mentioned TRAC instructions suffice to bring into operation the EASY program. Then to call EASY into operation, the user types:#(CL,EASY)'

The seond program, named DEF, DEFine, is used with EASY. It provides a format that distinguishes the name from the text to which it has been assigned, and replaces with one instruction what previously took three instructions to accomplish.

The program creates a 'table of contents' of names associated with the addresses of the stored texts to which the names have been assigned. Then to call DEF in operations

The user types:	**the computer responds by typing:**
CL,DEF'	NAME-- (Waits until user types in a name, such as 'LB'.)
	TEXT-- (Waits for user to type in a text, such as 'LEA BOHNERT'.)

To retrieve the text,

the user types: **the computer responds by typing:**

LB' LEA BOHNERT

Also, during the summer session of 1968, a SUNYA computer science student, Mr. Lester Dowell, developed a concordance program to be shown SLS students over SUNYATSEN, SUNYA Time-Sharing Electronic Network. We had received a copy of a concordance program in BASIC language from DR. T. E. Kurtz, head of the Kiewit Computing Center of Dartmouth College, Hanover, New Hampshire.

According to Dr. Kurtz, the program named VOCAB was in the spring of 1968, the only BASIC program designed to manipulate text, or alphabetic strings. The program had been written by Mr. Kirby Nichels, an undergraduate student at Dartmouth, to demonstrate to the English Department how a time-sharing computing facility could aid in the making of a concordance of a literary work. Mr. Nichels chose as the text 'ALTARWISE BY OWL-LIGHT', a poem by Dylan Thomas.

Mr. Dowell translated the BASIC program into the FORTRAN language, which was primarily designed for arithmetic work. He did have one advantage, the concordance work to be done had already been specified in data processing terms since it was a BASIC program. Mr. Dowell developed a special English-like program for our use, so the SLS students did not have to learn FORTRAN language. He also assisted in class demonstrations to nearly 50 students.

The program would (1) search a text for the occurrences of identical or nearly identical (same suffix) words, and (2) would type out the number of occurrences, the variations if any of how the word had occurred, the number of each variation, and the fraction of the text involved in these occurrences. For instance, when the word being searched was 'SEA', the computer would report:

'SEA' HAD OCCURRED TWO TIMES,
'SEAS' HAD OCCURRED ONCE,
'SEASTRAW' HAD OCCURRED ONCE, FOR A TOTAL OF FOUR OCCURRENCES OF 'SEA', OR .055556 OF THE TOTAL TEXT.

Finally, at the end of the summer session of 1968, Mr. Jackson Davis, a SLS student, wrote with the help of Mr. Mooers, the first TRAC program of the project. It was named TITLE SEARCH PROGRAM, TSP. The program demonstrated what is involved when a computer service is commanded to search a list of bibliographic entries. Mr. Davis, in addition to three of the four already mentioned instructions, had to use five other TRAC instructions:

EQ, EQuals
SS, Segment String
IN, INitial
DD, Delete
CR, Call Restore

The TSP program searches a list of bibliographic entries for a specified string of characters in a specified part of the entries. If the string is located in the specified part, the complete entry is printed out on the terminal keyboard. The search continues to the next entry whether the specified string was located in the preceding entry or not, until all entries on the list have been searched. Neither of the two TRAC programs, DEF or EASY, were used.

In the academic year of 1968-69, two SLS graduate assistants to faculty, Mrs. Phyllis Betz Ochs and Mr. Walter Doherty, gave demonstrations of TRAC language to more than 200 SLS students as well as concordance demonstrations to nearly 100 SLS students.

Mr. Walter Doherty learned TRAC language from the other students and by experimentation at the terminal. He not only gave class demonstrations of TRAC language but wrote two TRAC programs. He named the programs, CARDS and B, neither being an acronym.

The CARDS program demonstrated what is involved when a computer service is commanded to print a bibliographic entry in three different formats from one typing in of the data. He used the DEF program and two instructions, SS and CL. In addition, Mr. Doherty doublespaced the computer typing so that readability, a constant concern of librarians, was substantially improved.

To call the CARDS program into operation,

the user types:

CL. CARDS, APPLETON, TOM SWIFT AND HIS GIANT MAGNET, GROSSET, 1943.

the computer service responds by typing:

AUTHOR: APPLETON
TOM SWIFT AND HIS GIANT MAGNET
1943 GROSSET
TITLE: TOM SWIFT AND HIS GIANT MAGNET
APPLETON
1943 GROSSET
PUBLISHER: GROSSET
APPLETON
TOM SWIFT AND HIS GIANT MAGNET
1943

The B program demonstrated what is involved when a computer service is commanded to search a list of bibliographic entries by title or author name, and there is more than one entry with the same author's name. The DEF program could not be used because it requires an unique name for each part of an item or entry.

If 'author' and 'first title' is equal to 'author' and 'second title', the computer service responds by typing: 'ALIKE'. If the second title is different but the author's names are identical, the computer service responds by adding the second title onto the bibliographic entry in which the author's name first occurs. This results in all the titles by the same author being printed together.

For example, if there are two books by Shakespeare listed in a bibliography and **the user types:**

#(CL, SHAKESPEARE. WM.)'

the computer responds by typing:

SHAKESPEARE, WM.
HAMLET
AVON PRESS 66
MACBETH
AVON PRESS 67

Also, during the 1968-69 academic year, we requested the SUNYA Computing Center to provide an operational program for literature seaches. Mr. Paul Cleveland was assigned to help us. We chose as the material to be searched the bibliographic descriptions of the more than 350 seminar papers that have been prepared by SLS students over the last twenty years. The seminar paper, worth three academic requirements, is a requirement for the Master of Library Science degree. Previously there had been no access to the seminar paper except by the name of student(s).

To provide the bibliographic descriptions, about 50 SLS students in the required Research Methods course, Lib. 608, prepared during the 1969 fall semester FORTRAN coding sheets for all the seminar papers. Each student:

1) established the descriptive cataloging data for each paper, including title, author, faculty advisor and date,

2) expanded the title of any seminar paper whose title was not suitably descriptive of its contents by adding words in brackets after the title, 3) assigned to each seminar paper one or more of the 26 subject categories designed to provide some subject access to the seminar papers. The subject categories consisted of well-known library topics such as reading, user surveys, readers' services, public libraries, and library networks and cooperation.

During the 1969 spring semester, the SUNYA Computing Center keypunched the bibliographic data and developed a search program for use over SUNYATSEN. However, the final program had to be rewritten in another language because of the acquisition during the summer of a more powerful computer, an UNIVAC 1108 operating under the EXEC 8 time-sharing system.

The search program was named LIBSCI, LIBrary SCIence. It was written in COBOL by two computer science students, Mr. Paul Bailey and Miss Kaye Marron. They also developed an English-like command language for us to use over the time-sharing terminal. At least 50 SLS students made searches during the fall semester, 1969, to assist them in choosing or preparing their own seminar papers or other academic work.

There are six search commands in the LIBSCI program. They are directions to search for a specified string of characters appearing in a specified part of the file. The six parts that can be searched are:
1) the title, including any expansion in brackets, 2) the name of the author(s),
3) the name of the faculty advisor,
4) the codes of the subject categories assigned,
5) the year,
6) the accession number.

The computer service responds to any search command by typing: THERE WERE 000 ENTRIES LOCATED. The search commands can be used sequentially. For instance, one can search for a specific word in the titles, and if any entries are identified with the word, one can search them for the name of a particular faculty advisor.

The second, fourth and sixth searches are inefficient to perform individually over a time-sharing terminal and are available to all users in computer printed bibliographies for easy manual reference.

There are four file control commands in the LIBSCI program.
They are:
1) rearrange file by one part of the entry, such as name of author,
2) store rearranged file for future use,
3) call back file from storage,
4) type out at terminal keyboard a file that has been searched or recalled from storage.

As a part of the preparations for calling into operation the LIBSCI program, **the computer service types:** DO YOU WANT A LIST OF AVAILABLE COMMANDS.

If the user types:
YES
the computer service types:

TITLE	=	SEARCH ALL ENTRIES FOR TERMS ENTERED (60 CHARACTERS MAX.)
SUBJECT	=	SEARCH ALL ENTRIES FOR SUBJECT-CATEGORY CODE ENTERED.
AUTHOR	=	SEARCH ALL AUTHORS FOR NAME ENTERED.
ADVISOR	=	SEARCH ALL ADVISORS FOR NAME ENTERED.
YEAR	=	SEARCH ALL YEARS FOR 2 DIGIT YEAR ENTERED (EX. 65).
ACCESSION	=	SEARCH ALL ACCESSION NUMBERS FOR NUMBER ENTERED
SORT	=	SORT THE SUB-FILE BY YEAR, AUTHOR, ADVISOR, OR ACCESSION.
SAVE	=	SAVE THE CURRENT SUB-FILE in SAVE 1 OR SAVE 2.
RECALL	=	RELOAD EITHER SUB-FILE 1 OR SUB-FILE 2.
PRINT	=	PRINT THE ACTIVE SUB-FILE.
NEW	=	ERASE THE CURRENT FILE, START A NEW SERIES OF SEARCHES.
BYE	=	END THE PROGRAM – FOLLOW WITH A FIN TO SIGN-OFF.

We also had requested three computer print-outs of the bibliographic descriptions of the SLS seminar papers. Two would be alphabetically arranged by name of faculty advisor or code of subject category assigned, and chronologically within each group of papers. The third would be arranged alphabetically by name of student.

The main purpose of the biliographies was to provide access by subject category and by faculty advisor. A secondary purpose was to provide library students with a good example of an inexpensive but readable computer printed bibliography.

The printed bibliographies will:

1) be suitable for copying onto 8½ by 11 inch sheets of paper,
2) have pages numbered,
3) have no hyphenated words because too long words for a line have been moved down to next line.
4) have only complete bibliographic entries on each page. It was the first time that the SUNYA Computing Center had provided pagination of its output.

12. CONCLUSIONS

There were two main conclusions drawn from the Reactive Typewriter Project. First, we demonstrated that library students could learn easily and quickly to command a computer service to perform library tasks using a time-sharing terminal:

1) with the aid of one of the new user-oriented languages, such as TRAC,
2) with a special search program for a list of bibliographic entries, such as the descriptions of the more than 350 SUNYA seminar papers.

Second, the project demonstrated that SLS students who learned to program TRAC language, could communicate effectively with computer science students when we wanted a special search program and print-outs to be developed for the SLS seminar papers.

REFERENCES

1. Hilsenrath, J., et al, 1968, OMNITAB: A Computer Program for Statistical and Numerical Analysis (U.S. Government Printing Office, Washington).
2. Kemeny, J G., and Kurtz, T. E., 1968, Dartmouth Time-Sharing. Science 162, 3850, pp. 223-228.
3. Kurtz, T. E., 1969, The Many Roles of Computing on the Campus. In: AFIPS Conference Proceedings, 1969 Spring Joint Computer Conference, Boston (AFIPS Press, Montvale, N. J.) pp. 649-656.
4. Mooers, C. N., 1952, Information Retrieval Viewed as Temporal Signalling. In: Proceedings of the International Congress of Mathematicians, Sixth Congress, Cambridge, Massachusetts, 1950, vol. 1 (American Mathematical Society, Providence, R. I.) pp. 572
5. Mooers, C. N. 1963, Comm. ACM 6, 48.
6. Mooers, C. N. 1965, TRAC, a Text Handling Language. In: Winner, L. ed., Proceedings of the 20th National Conference of the Association for Computing Machinery, Cleveland, 1965 (Lewis Winner, New York, N. Y.) pp. 229-246.
7. Mooers, C. N., 1967a, Standards for User Procedures and Data Formats in Automated Information Systems and Networks. Part I. The Need for Standardization and the Manner in Which Standardization Can Be Accomplished. (Clearinghouse for Federal Scientific and Technical Information, Springfield, Va.) (PB 177 550)
8. Mooers, C. N., 1967b, Standards for User Procedures and Data Formats in Automated Information Systems and Networks. Part II. The Standardizable Elements of User Control Procedures and an Unified System Model. (Clearinghouse for Federal Scientific and Technical Information, Springfield, Va.) (PB 177 551).
9. Mooers, C. N., 1967c, Standards for User Procedures and Data Formats in Automated Information Systems and Networks. Part III. A Suggested Standard Keyboard Assignment for the Elemental User Control Actions. (Clearinghouse for Federal Scientific and Technical Information, Springfield, Va.) (PB 177 552)
10. Mooers, C. N., 1967d, Standards for User Procedures and Data Formats in Automated Information Systems and Networks. Part IV. A Standard Method for the Description of External Data Formats. (Clearinghouse for Federal Scientific and Technical Information, Springfield, Va.) (PB 177 553)
11. R.E.S.I.S.T.O.R.S., 1969, Brochure distributed by the R.E.S.I.S.T.O.R.S. during the 1969 Spring Joint Computer Conference in Boston, Mass.
12. Sammet, J. E., 1969, Programming Languages: History and Fundamentals. (Prentice-Hall, Englewood Cliffs, N. J.) p.vi.

NOTES

* TRAC is the trademark or service mark of Rockford Research Institute Incorporated.

MASTER - A SUBSYSTEM FOR PROCESSING STUDENT SUBPROGRAMS

I.N. Capon

1. INTRODUCTION

Student control systems are by no means new. They fall into two classes, the first of which requires each student routine to call the supervisor, while the second arranges for the supervisor to call the student routines. The first method is more common, and is easier to insert into most operating systems, but can be confounded by illegal (or missing) calls to the supervisor. The second approach offers greater control provided the operating system can be prevented from rejecting the entire job when an error occurs. The MASTER system is an example of this second class.

A grading system should have several properties, including:

(i) immunity to syntactical or execution errors met in processing student programs;
(ii) non-interference with the student's mode of writing, or with his control of the system;
(iii) provision of suitable summary or detailed reports to the instructor;
(iv) security of both input data to be supplied to the student (to prevent cheating) and of any marks awarded by the instructor;
(v) flexibility in dealing with various programming languages.

We first consider briefly the MASTER system in relation to these requirements.

Many student control or grading systems have difficulty in adequately dealing with 'fatal' student syntactic errors, and with tight loops. These problems are detected and reported automatically by MASTER.

The MASTER system interferes with the student only in that it requires all student routines to be written as subroutines with set parameter sequences.

The system automatically produces a report file containing status information for each student routine, and any 'answers' produced by the student. It is also possible for the instructor to examine the student's output and to write comments onto the student's output.

It is not possible for the students to interfere with data or other files used by the instructor.

The current version of MASTER is limited to only one programming system at a time. This paper is written around a FORTRAN version, but other varieties are readily produced.

The following paragraph outlines the procedure involved in using MASTER from the instructor's viewpoint. This overall view is necessary for an understanding of the more detailed description given later.

The instructor controls the MASTER system through two routines supplied by him, which are called MAIN and REPORT in this paper. The MAIN program is used to set up appropriate test data, to arrange for the required setting of time limits, to call (apparently) a single student program, and to store the student's results. Each student writes his own version of a subprogram, whose name and ordering of parameters is stipulated by the instructor. The entire set of subprograms, and MAIN, are processed by the MASTER system to produce, in parallel, a file which contains all the information relevant to the operation of the students' jobs and the normal (printed) output expected by the student. Finally, the routine REPORT is called in by MASTER to read back the file and to produce any desired summary and/or historical reports.

This is no limit to the number of students which can be processed at one run, nor need this number be fixed or specified to the system. Any student program may loop, fail to compile, or otherwise 'run wild', without upsetting the overall operation or affecting any other student.

2. PROGRAMMER'S DESCRIPTION OF THE SYSTEM

2.1 *Implementation*

The MASTER system is designed to run under the standard SCOPE 3 monitor, RUN FORTRAN compiler and GPSL loader in use on the CDC 6400, and the MASTER system is essentially a modification of the normal control sequencing among these systems.

The system is dependent upon the fact that the 6400 is an assembly of 11 functionally independent machines, one high performance and 10 peripheral processors (PP). The overall monitor routine resides in one of these PP's and allocates the main processor and the other nine PP's to various jobs on a multiprogramming basis. It is thus possible for the monitor, or any other, PP to be isolated from a program which "runs wild" and toconstruct a system which is immune to bugs in a program under test. This is of course done for SCOPE itself, and the MASTER system is in effect at a second level of immunity. The details of this procedure are available from the author.

2.2 *Processing for MASTER jobs*

The MASTER routine is called in by a normal job control statement, and itself generates the job control statements required to run student jobs submitted to it. It generates a set sequence of control statements for each student, thus simulating a "control statement loop" which is not normally possible. (A later version of SCOPE allows limited looping, but the reporting of errors is inadequate for current purposes.) These control statements make use of certain parameters which may be included in the MASTER control statement. (Details are given in Table 1.) The notation "P1" etc. in the following refers to these parameters.

The following assumes some knowledge of the SCOPE system; see (1).

The first statement generated is

```
RUN(S,,,,,SUPER,P5)
```

This statement forces compilation of the instuctor's MAIN program and stores the relocatable binary object code on the file SUPER.

The second statement is

```
RUN(S,,,,,REPORT)
```

which compiles the report tabulating program to be used at the end of the run into file REPORT. It is worth noting that MAIN and REPORT are compiled once only.

MASTER is now ready to start on the student decks. For each deck it copies a record form INPUT to COMPILE. If the end-of-file has not been reached, it checks the first card for a valid identification. The first 20 characters are saved as the deck identifier for the report file and are also entered in the dayfile ("operating log") and listed at the top of a new page on OUTPUT. The card is then blanked out and the files COMPILE and LGO are rewound. The sequence of control statements

```
RUN(S,,,COMPILE,,,P5)
LOAD(LGO)
SUPER.
```

is then executed. These produce an object version of the student routine on LGO, load this routine and the instuctor routine MAIN, and initiate execution of MAIN. When the program terminates, either normally or abnormally, a report record is written on file P6. If answers were returned they will be included, otherwise noise is written to fill out the record to the full length.

The dayfile is then searched and entries relating to this job are printed on OUTPUT.

When finally the end of file is reached on INPUT, the control statement REPORT. is generated. This calls the report program into execution to tabulate the data on the report file.

3. FACILITIES AVAILABLE TO THE INSTRUCTOR

The MAIN routine specifies the desired actions to MASTER by a series of calls to routines named QSTART, QTIME, QSTORE and QFINISH. The routines QSTART, QFINISH carry out necessary opening/closing actions for each student in turn, and the instructor has merely to include one call to each at the extremities of MAIN.

The routine QTIME(n) is used to set up a time limit of n seconds for the next execution of a student program. This limit can be dependent upon the data supplied for this execution. If the student does not return control to MAIN within this time his calculation is cut off, and an appropriate entry made in his report.

Provided each execution is successful, the student routine may be called again and again. The results and/or instructor's 'marks' for each individual run are placed in a protected buffer by means of a routine QSTORE, which writes P7 words on each entry.

A typical instructor's routine MAIN will thus contain calls somewhat as follows:-

```
CALL QSTART
  .
  .
  .
CALL QTIME (2)
CALL STUDENT (...L...)
CALL QSTORE (A)
  .
  .
  .
any coding concerned with "on the spot" checking
of answers, and commenting to the student
  .
  .
  .
CALL QFINISH
  .
  .
  .
END
```

After the desired number of executions (or after an unsuccessful attempt) a report record consisting of several sets of answers together with 4 header words, is written from the buffer onto file P6. Words 1 and 2 hold the first 20 characters of the student's name card. Word 4 holds the number of successful student executions. Word 3 holds the status data for the student in 3 low-order octal digits, which are called the CONTACT, ERROR and PHASE codes respectively.

The PHASE code indicates how far processing for the deck proceeded before termination. The other fields indicate the reason for termination. Details are given in Table 2.

If all executions terminate correctly, the status word will indicate a "clean sheet". If any execution terminates abnormally, those remaining for this student will be abandoned and the status word will indicate the nature of the abnormal termination. (It is therefore desirable to supply data in increasing order of difficulty.) It is possible to use the above mechanism to execute several different programs per student in one MASTER run, but if any errors occur the whole of the remainder of the student's work is passed over (not just the current program). This procedure is therefore not usually suggested.

4. GRADING STUDENT EXERCISES

The grading of student exercises submitted to MASTER can either be done by the instructor from the output supplied by the system, or by a suitable routine included as part of the REPORT processing. The grading formula to be used is of course somewhat dependent on the subject matter, but may include

(i) the correctness of any numeric or symbolic answers returned by the student routine,
(ii) the efficiency (time or space) of the routine,
(iii) the number of runs required to bring the routine to a workable state,
(iv) the generality of the routine,
(v) the 'error proof' or 'fail safe' properties of the routine.

It is probably not possible to satisfy every instructor simultaneously, and the 'normal' system is so arranged that various parts of the grading routine can be altered or ignored. The normal system allots marks separately for correctness, efficiency and expeditious check-out.

The 'correctness' of a routine is easily determined if there is only one answer, or at worst several discrete answers. In many cases however a numeric answer may be approximately correct, and it is desirable to allow some part of the marks for answers near the correct one. The

current version of our grading routine takes as parameters the correct answer and a penalty scale, which specifies the marks to be lost for departures from the correct answer. This scale is assumed linear. Errors can of course arise from many sources, ranging from arithmetical blunders and transcription errors to inappropriate methods, but it is not possible to allow for these variations in a general routine. In some cases it is possible for the instructor to arrange a graduated set of test data to 'explore' the student routine and so assess the method. This is however a difficult procedure, as may be illustrated by a trivial example. Consider an exercise in an elementary course in numerical methods, to solve a nonlinear function of one variable. The test data consists of a set of function and derivative evaluation routines and starting values. Each of these triplets is supplied in turn to the student routine.

Triplet 1: The linear function $f = x - 1$, starting value 0. Failure to obtain the correct solution exactly is an indication of a gross error in the student's algorithm or coding. No marks are awarded in such cases.

Triplet 2: The cubic equation $f = x^3 - x^2 + x - 1$, starting value 0. There is one solution, reached immediately by an application of Newton's method, which the students are expected to use. Again, an exact answer is required.

Triplet 3: Several 'text' examples (changed from time to time) with full marks allotted for solutions within a tolerance of 0.001, zero elsewhere. (This reflects the performance the students are told to aim at.)

Triplet 7: (approx., depending on instructor)

An ill-behaved example, such as $f = x^{20} - 1$, $x_0 = \frac{1}{2}$.

Most students at this level of experience are expected to 'catch' this example via some limit on the number of cycles, and a passing mark is awarded for any answer. This is reduced to zero if the routine was cut off by MASTER, or increased if the answer is better than that obtained by the blind application of the Newton-Raphson process. A student who used (e.g.) a secant iteration out of ignorance might thus obtain above average marks 'unfairly'.

Similar techniques may sometimes be used to assess the generality and "fail safe" nature of the routine, but obviously no generalized technique can completely check out any routine submitted to it. Indeed there are few situations in which even a particularized technique can completely check out a routine.

The efficiency of a routine may be assessed by timing it, or by examining the storage used, while the efficiency of the student involves both of these factors and the number of runs needed to construct the routine. In most cases it is sufficient to simply record these quantities for perusal by the instructor. 'Marking' is sometimes attempted by specifying an acceptable execution time and a linear loss function for overtime. This approach may be extended in the way described by Bennett (6), and our currently most elaborate version assumes a linear loss function for both overtime and extra runs. The instructor is asked to nominate the standard time and rate of loss, and the number of runs which would attract a given score if a 'perfect' time was achieved on that run. For example a 1 second run, with a loss rate of 2 mark/.2 second, with 5 runs allowed, is assessed as follows:-

Nominal value of exercise: 5

time runs	...	.75-.85	.85-.95	.95-1.05	1.05-1.15	...
.						
.						
.						
3	...	11	10	9	8	...
4		9	8	7	6	
5		7	6	5	4	
6		5	4	3	2	
7		3	2	1	0	
.		.			.	
.		.			.	
.		.			.	

Note that it is assumed that the rate 2/0.2 also dictates a loss of 2/run; a rate of 1/0.1 would imply a loss of 1/run.

It is however doubtful whether this procedure is justifiable as a grading device; it may serve to give the instructor a quick guide to performance. It is not true that that performance improves monotonically, let alone linearly, with run number as assumed by the above scheme. It is in fact necessary to decide which run should be graded. The current version always chooses the best performance, regardless of its position in the submission order. Finally, the system is used to supply summary statistics by student, to bring out his overall performance, and by exercise, to indicate exercises which are too easy or too difficult. This processing is done on demand by standard techniques.

5. DISCUSSION

The MASTER system is notably superior to many grading routines (references (2) - (5)), in that it does not depend upon there being a known number of student programs which are syntactically correct. Further, the student is not required to conform to any but the usual programming rules. He is not, for example, told to include a call to some instructor routine within his own program. He does of course have to conform to the instructor's ordering of the parameters of the routine, and he therefore cannot be asked to decide what the parameters should be, except perhaps from among a "noisy" set.

Under the existing compiler, it is theoretically possible for parts of the routine MAIN to be overwritten by a student program, but at least the status word will be correct in the report. In all cases of overwriting the time limit option will eventually return control to MASTER, so assuring recovery. The status word will then have phase code 3 and probably some error code. A specialized compiler is being written which will, inter alia, check for this condition.

It is not possible for the student to accidentally interfere with the report file.

If the instructor wishes to check the student's answer immediately for possible comment to the student, he may insert appropriate code after CALL QSTORE. Any output at this point will appear on the student's output. However, as MAIN is currently reloaded many times, extensive checking etc. is best left until REPORT.

The operational disadvantages of MASTER are slight. Potentially, it could degrade the performance of the multiprogramming system and it does increase peripheral processor time compared with independent running of each student job.

The MASTER system involves the FORTRAN (or other) compiler, and the entire run must therefore be made

with a large core assignment. Such jobs cause relatively inefficient machine utilization, especially when the overall time limit is large (for a large class) and the job of high priority (for academic reasons of class schedules, etc.) and probably I/O bound. Further work in the direction of using the whole of core for in-core files (in the interests of reducing backing store accesses, currently a bottleneck) will probably remove (rather than solve) this problem, as the system will then effectively become a single-program with no input or output activity.

The peripheral processor time is of course increased by a large factor as one PP is "active" for the entire run. However, as there is no shortage of PP's this is not yet a serious operating problem.

The overall elapsed time may be slightly increased due to the bookkeeping activities of Master. However this is usually counterbalanced by a saving from the student's reading of a file being replaced by parameter transfer from the instructor's routine. In many cases this effect results in an overall reduction in elapsed time.

The major reaction to MASTER by non-computing science faculty members was over the ban imposed on students reading data "directly" from cards. This is not a necessary restriction – an alternative version allowing such input has been designed but not implemented – but was deliberately imposed in the interests of economy and to obtain two pedagogical advantages. The economy of course arises from the elimination of large numbers of backing store transfers (as all reading of cards is done independently and results in backing store files). This could be amended, but the ban on student input ensures that student routines are not "tailor made" to known data, and also means that students need not concentrate on the input rules of the programming language, which are often a source of confusion in elementary classes. After some discussion, our colleagues were convinced that the "secrecy" advantage alone was sufficient reason to ban student data cards. In those few cases where students need access to unstructured raw data, it is possible for the instructor to set up a suitable file which can be read by the student.

6. PERFORMANCE

The MASTER system came into operation for the 1969 academic year (March-November) and has been accompanied by a reduction of over 50% in the number of separate student jobs. Some of this reduction was due to stricter control measures, but there is no doubt that a great deal of the usual class work was submitted under MASTER. To date there have been no problems with the system.

7. ACKNOWLEDGEMENTS

Many people have contributed to MASTER. Particular thanks are due to D.M. Haycraft, V. Fris and P.M. Ryan who have done much of the programming.

REFERENCES

(1) Scope Operating System Manual (1968), CDC Publication 60173800, 60189400. Control Data Corp., Palo Alto, California.
(2) Naur, Peter. (1964) "Automatic Grading of Students' ALGOL Programming", BIT 4 3177-188.
(3) Forsythe, G.E. and Wirth, N. (1965) "Automatic Grading Programs", Comm. A.C.M. 8 275-278.
(4) Berry, R.E. (1968) "Grader Programs", Comp. J. 9 3 252-256.
(5) Temperly, J.F. and Smith, B.W. (1968) "A. Grading Procedure for PL/1 Student Exercises", Comp. J. 10 4368-373.
(6) Bennett, J.M. BAGS – Basser Automatic Grading Systems. Tech. Report, Basser Computing Laboratory, University of Sydney.

TABLE 1 – MASTER PARAMETERS

Name	Value	Effect	Default Value
P1	L or LIST	print a source listing of each student program,	L
	else	no list	
P2	M or MAP	print a core map for each student program,	M
	else	no map	
P3	D or DAYFILE	print dayfile entries for each student program,	D
	else	no dayfiles	
P4	up to 4 octal digits	CP time limit for each student program	10 but see QTIME
P5	up to 4 octal digits	execution output line limit for each student program	200
P6	up to 7 character name	file for report	TAPE 10
P7	up to 2 decimal digits	dimension of answer array to be included in report for each student program execution	1
P8	up to 2 decimal digits	maximum number of answer groups in any student's report, i.e. if the maximum number of executions of a student program in any one MASTER run is n, P8 = n	1

TABLE 2 – ERROR CODES

Phase	Interpretation
0	Bad name card
1	FORTRAN errors ('failed to compile')
2	Loader error
3	Execution error (reached QSTART)
4	Termination error (reached QFINISH but did not terminate normally)
5	Normal completion
Error	**Interpretation**
0	No errors
1	Time limit
2	Arith. error (divide by 0, illegal address, etc.)
3	PP abort
4	CP abort
5	PP call error
6	Operator drop
7	Disc limit
Contact	**Interpretation**
1	Bad QSTART/QFINISH calls
0	O.K.

(Phase 4 errors are very unlikely, unless the instructor routine MAIN is in error, or if the time limit is reached between CALL QFINISH and the end of MAIN.)

THE SPECIFICATION OF AN ELEMENTARY LANGUAGE FOR COURSES UNDER THE UNITED KINGDOM COORDINATING COMMITTEE

D.E. Conway

1. INTRODUCTION

The United Kingdom Coordinating Committee for Examinations in Computer Studies set up a subcommittee to study teaching languages in the Autumn of 1968. The committee is still active and has not completed its objectives. It reported back to the main committee in the Summer of 1969 and received approval for its work and was asked to continue along the lines recommended by the subcommittee. During the first year, I was the chairman of the subcommittee. This position is now in the capable hands of W. Tagg who represents the B.C.S. on the subcommittee. When the committee was set up, it contained representatives from the following interested parties:

B.C.S.
C.G.L.I.
Institute of Computer Science
NCC
R.S.A.
SANCAD
SCCAPE

During the first year, various people joined the subcommittee by invitation, including a representative from the AEB, who offer an A-level course in Computer Science. In this paper, I intend only to outline the work of the subcommittee during its first year. It will be of considerable interest to report later the subsequent work of this hard-working subcommittee.

2. PRELIMINARY WORK

It was essential to establish a clear understanding of the aim of the subcommittee and what responsibilities it should undertake. With this in mind, the subcommittee discussed at some length the problem of providing common programming languages for use in examinations and courses in computer studies. This discussion was valuable, as the representatives had varied and extensive experience in both examining and teaching on these topics. With an array of conflicting objectives, contrasting experiences, and areas of special interest, it was surprising and reassuring that the subcommittee was able to support common objectives and terms of reference.

It was accepted that basic features of any common language must meet the educational requirements of the courses; i.e., the need to develop the basic concepts of the computer and programming. It was also recognized that the practical needs of the students and teachers would be a major influence and that the restricted availability of hardware and software would have to be continually kept in mind.

The subcommittee then considered the range of courses under consideration and identified two distinct facilities which are needed:

2.1 *A low-level language giving insight into computer concepts and leading to fundamental, low-level programming concepts.*

2.2 *A high-level language to enable problem-solving concepts and techniques to be explored. (In particular the demonstration of applications programming)*

These facilities are mixed in various quantities, depending on the purpose of a given course. It was also recognized that different instructors approach the same material in different ways, even when following the same syllabus. Some disagreement arose on the roles of high and low-level languages, in particular kinds of courses such as „appreciation courses" and „systems analysis courses". It seems clear that at present the evidence is still conflicting as different people achieve success by using exclusively one or the other and even a mixture of both. The picture is further confused, by the time restriction, which varies from course to course. However, the subcommittee concluded that nothing good would arise, if they could recommend a common basis for both, while not suggesting either be adopted in any specific course. If asked to recommend, I would suggest both be made optional features of all courses.

The subcommittee then decided to concentrate on the identification of the low-level facility. (The study of the other facility is now underway.) It was felt that the position of the low-level area would be more practicable, as the features are less conflicting and the number of facilities currently available very limited in number. The existing facilities were then discussed, in the light of their use by members of the committee and evidence they had obtained from their sponsoring bodies. There were two varieties of language, which were the ones in widespread use with recognized courses such as TAM2 and the C & G 319 anemonic code and ones used mainly in particular institutions such as MINAC. The subcommittee agreed that the low-level facility must be established, with a well-defined and consistent hardware base, together with an assembler program defining an external version of the hardware machine language. In the light of limited facilities and the wide variety of computer installations which have to be considered, the subcommittee thought the hardware base should be such that it is realisable by means of a simulation program, working in a variety of actual computers. It is worth emphasizing that this is different from providing a compiler. Each of the simulations should be able to use a common assembler (assembly language compiler.) The existing facilities did not meet these needs, being either too complex or not sufficiently well-defined.

The subcommittee then undertook the task of specifying a suitable facility. After considerable work, the subcommittee have evolved a fairly consistent draft of the specification of the low-level facility, which is the result of many discussions and, of course, considerable changes in direction and emphasis. I feel that the recommendation should meet the needs of most educators in this field. The writing of software, to test the ideas, is currently being undertaken by the subcommittee.

3. THE LOW-LEVEL FACILITY

The facility is best described as consisting of:

a. A well-defined hardware configuration
b. A simulator for the hardware
c. A well-defined assembly language
d. An assembler
e. Good documentation

I am not going to attempt to give either a detailed or exhaustive description of the facility, but rather indicate the lines along which the subcommittee have developed their ideas. The United Kingdom Coordinating Committee has authorized the City and Guilds of the London Institute to publish the documentation produced by the subcommittee. These publications should be of considerable interest to educators in this field. It is planned to produce special documentation as follows:

f. Detailed formal description of the machine and assembler
g. A guide description to the machine and assembler
h. An introduction to the machine and assembler
i. A version of the assembler written in its own code
j. A detailed description of the implementation of a simulator

4. THE SIMULATED MACHINE

It is an automatic stored-program digital computer. It is not intended to use this facility for serious or substantial programming. It is assumed that students will be able to use some suitable high-level language for this purpose (COBOL, FORTRAN, PL/1, BASIC). The machine provides arithmetic operations with binary operations and some logical operations defined as transformations of binary words as bit patterns. The binary words are interpreted in accordance with the 2's-complement convention.

The word length is to be 18 bits (6 octal digits or 3 six-bit character patterns). The store size is 1024 cells (1k) and they are to be addressed-512 in increments of 1 to*511 using a 10 bit binary pattern. The instruction format is to be single word one-address, which will have a 7 bit function specifier and a 10 bit operand specifier. This leaves a spare bit, which is to be used for diagnostic purposes(e.g. trace facilities). Instructions fall into three types:

IMMEDIATE	the operand 10 bits are extended to form an 18 bit immediate operand (8 copies of bit 9 extended to its left)
DIRECT	the operand 10 bits are the address of a cell which holds the 18 bit operand
INDIRECT	the operand 10 bits are the address of a cell which in turn holds the address of a cell which contains an 18 bit operand

4.1 *Some examples of instructions are:*

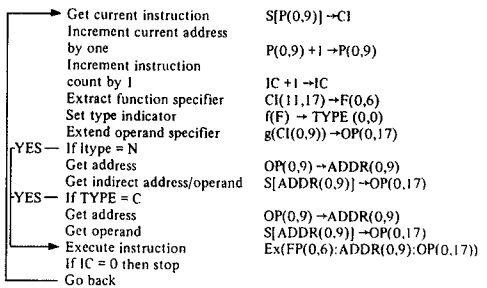
A PUT instruction which places the contents of an accumulator into a cell

Function Code			
binary	octal	decimal	description
0111000	070	56	Store Acc in cell specified
0111110	076	59	Store Acc in cell whose address is in the cell specified

4.2 *Certain hardware facilities are associated with reserved cells +0 to +15.*

+0	Z	a read only cell, containing 18 ϕ-bits
+1	P	the current instruction address cell
+2	A	the 18 bit accumulator
+3	B	the 18 bit accumulator extension (to the left of A)
+4	T	the read only test cell, used to record the results of comparisons, etc.
+5	R	the division remainder cell
+6	IC	the instruction count cell
+7	L	the subroutine link cell

4.3 *The basic operation of the machine can be best appreciated by looking at the instruction execution cycle.*



4.4 *Example of a logical instruction ---AND*

AND immediate	A(0,17)∧OP(0,17) →A(0,17) A(17) →B(i) for i = 0(1)17 test ([BA]) →T(15,17)

This would complete a logical „and" operation between the extended operand part of the instruction and the accumulator. It would then set the B-accumulator using the left-most bit of A and finally reset the test register.

4.5 *Example of an arithmetic instruction ---MULTIPLY*

MPLY direct	n(bxp) → BA(0,35) test([BA]) →T(15,17)

where b is the 36 bit 2s-complement numeral held in BA and p is the 18 bit 2s-complement numberal held in the cell specified by operand.

4.6 *Example of a transfer control function — JUMP IF ZERO*

JUMP indirect	If[T(16)] = 0	then ADDR(0,9) →P(0,9)

This couses the next instruction to be found in the cell whose address is specified in the cell, specified by the operand part of the instruction.

4.7 *Example of a I/O operation – (1/) INPUT (2/) OUTPUT*

4.7.1 INCHE – causes the input of the next character from an input stream character

→ 6 bit {from input-stream} →S[ADDR(0,9)](0,5)
z (6,17) →S[ADDR(0,9)](6,17)
move input stream forward.

4.7.2 OUTCH – causes rightmost 6 bits of specified operand are used as internal bit pattern to produce output character on output stream.
[OP(0,5)] →character →push down output stream

4.8 *Operational switches*

These will again be simulated and will provide such controls as:

START – loads predetermined instruction
MANUAL/AUTO
18 Hand Keys
LOAD CONTROL – using hand keys to set P
LOAD STORE – hand keys copied to store all specified by P and others

5. THE ASSEMBLER

This is a program that converts the assembly language into its machine language form. It is to be implemented in two forms, that is to say, in terms of itself, and loadable in the simulated machine, and in a special form independent of the simulated machine. This language is very simple and is explainable in every case, in terms of the simulated hardware. It is envisaged that the language can be taught at various levels of sophistication, including the most elementary.

5.1 *Example of program*

The assembler accepts a form of assembly language, which is not completely in a one-to-one relationship with the machine code. In many implementations, there will be four different classes of information processed by the computer, simulator directives, assembler directives, assembly code and program data. It is not intended to give a comprehensive view of the assembler, so a simple example is given for a simulated machine, with paper tape input and output (such as might be implemented on a PDP8 or Elliott803).

5.1.1 The program searches a data tape looking for the character „1" and then punches the next 10 characters.

5.1.2 Flowchart (see page III/174)

5.1.3 Job Tape

```
**START
* JOB
: EX. 1 :
* H = 20
* LIST
= 10
= 0
CHAN 1          (Select P/T I/O)
L1) ICHD 1      (Read Character to Acc)
SUBN 33         (Check for 1)
JAND L1         (Jump to L1 if not 1)
L2) ICHD 1      (Read Character)
OCHD 1          (Punch character)
WUND 21         (Count characters punched)
CPYD 20
SUBD 21
JAND L2         (Jump to L1 if Store 21 is not 10)
HALT
*RUN = 22

ABCDE 1 2 3 4 5 6 2 3 4 5 6 DATA TAPE
```

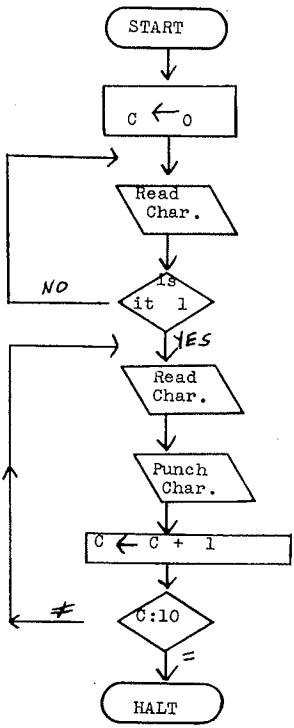
5.1.4 The program uses two constants, stored at the head of the program for counting. The test data tape should cause the output of the string of digits „2 3 4 5 6 2 3 4 5 6". Note that the internal form of the character „1" is 33. Most of the instructions are almost self-explanatory, but if may be worth mentioning that „WUND 21" will cause the contents of store 21 to be incremented by one.

5.2 *Diagnostics*

It is intended to specify a complete set of diagnostic facilities with the assembler. There will then be no ambiguities, or local dialects, which will confuse students in examinations. Some special debugging facilities, such as „optional printing" and „trace" will be specified. It will be also possible to build into the simulator program some good diagnostic facilities, to enable the educator to control the test programs being submitted by the students. Typical facilities of this kind will be described in the documentation referred to in 3.10. It will be easy to explain how diagnostics are produced, because the structure of the machine and assembler will be simple. This, it is hoped, will enable the educator to avoid any mystery.

6. CONCLUSION

The progress made by the subcommittee has been remarkable and I would like to thank all the members who gave so freely of their time, effort, and experiences. The achievements briefly described above are the result of much work and discussion by the whole committee and I would like to make clear that personal acknowledgements will be made, when the official documentation is produced by the City and Guilds of the London Institute. This paper is my personal view on the work and does not represent the final conclusions of the subcommittee.


START
C ← 0
Read Char.
Is it 1
NO
YES
Read Char.
Punch Char.
C ← C + 1
C:10
≠
=
HALT

DESIGN OF LANGUAGES FOR ELEMENTARY PROGRAMMING INSTRUCTION: LESSONS OF THE TEACH PROJECT

Robert R. Fenichel

1. INTRODUCTION

The TEACH system [Fenichel et al. [1970]] is an automated instructor in elementary programming. Like any other instructor in elementary programming, TEACH expresses its examples in a particular programming language. The language used by TEACH is UNCL, which was designed especially for the purpose.

The designers of UNCL and TEACH came to agree upon several principles of language design for elementary programming instruction. This paper describes some of those principles.

The work described here was a result of long, sometimes heated, and almost radially-symmetric discussions among Professor J. Weizenbaum, Mr. J.C. Yochelson, and the author.

2. DIGRESSION ON ELEMENTARY PROGRAMMING INSTRUCTION

In the early days of programming, students were initiated into this discipline with the study of machine language. In the very earliest days, there was no other language. Later, when the first higher-level languages became available, it was still almost essential for even the most casual user to have an intimate knowledge of the underlying machine. In any event, there was then little established substance to computer programming beyond the tricks of coding.

By the early 1960's, coding had become a minor skill, and the established problems of programming were problems of producing algorithms to solve various easily-described problems, such as those of searching and sorting. Elementary courses began to rely upon higher-level languages, using machine language only secondarily. The problems of man-machine communication were still thought to be severe, so it was usual to teach one's way through a language systematically, statement-type by statement-type.

Today, the dominant problems of programming are not problems of coding, or even problems of devising ingenious small algorithms. Instead, the dominant problems are those of **big programs** -- programs too big for any one man to comprehend completely. This shift in emphasis is only beginning to make itself felt in elementary courses.

Obviously, we cannot ask the fledgeling programmer to write a **big program** (in the sense just described) as part of his homework exercises, even if the notion of **big program** is appropriately scaled down. Instead, he must learn the nature of programming just as he once learned the nature of speech -- by example. Instead of asking him to produce a small program, we ask him to modify a big one. The big program demonstrates techniques of organization and convention which could be invisible or unjustifiable in any quantity of small coding exercises. While his early modifications may require him to master only a small portion of the language in use, he will be exposed from the first to much or all of that language.

2.1 *Principle #1: It is more important that programs be readable than that they be writable*

This principle derives as much from a view of programming *per se* as it does from any view of programming education. On the latter ground, the argument is simple. Those who are new to programming are not skilled at decoding programs, even their own. At the moment of need, they will discover from their manuals that, e.g., arrays in APL [Berry [1969]] are created by an infixed use of the Greek letter ρ. Of course, this letter is quicker to type (more writable) than the word „array". But an hour later, they will see their own

17pA

and be mystified again.

This argument is especially strong when student exercises consist of modifications to existing big problems. Even though these exercises may each involve a small subset of the language in which the big programs are written, the point of the exercise is lost if the remainder of the program is altogether unintelligible.

Finally, as the student's modifications to the big program become more and more profound, the case is clinched. For now he is up against the dominant problem in programming. That is, he is now concerned with making his programs comprehensible to people, as well as machines. His programs must not only work; they must also seem to work.

The truth-values in UNCL are named „true" and „false", as in Algol [Naur[1960]], BASIC [Kemeny and Kurtz [1966]], and SPL [Anger et al [1969]]. The pun encouraged by APL and PL/1 [Anonymous[1964]] (i.e., true = 1; false = 0) is eschewed. In APL or in Pl/1, the expression

5>6>7 (1)

has the same value as

6<7 (2)

while in UNCL, the expression (1) is not merely false, but even ill-formed.

Identifiers in UNCL may be of arbitrary length. Students take only a short while to give up the use of witless, non-mnemonic, one-character identifiers in the BASIC style; they tend to settle in the range of six to nine characters.

Most operators in UNCL are given mnemonic manes. The four arithmetic operators are given their usual infix

representations, of course, and UNCL uses FORTRAN's double asterisk for exponentiation. But UNCL's string-concentration operator is spelled „CONC", despite the inconvenience of this spelling relative to, say, PL/1's „||" or, even less readable but more writable, APL's comma.

The assignment statement of UNCL embodies a keyword, „set". In an earlier version of UNCL, we had allowed students to elide this word, so that assignment statements might look like the corresponding statements in FORTRAN or PL/1. We found that students then managed to confuse the notions of assignment and static equality was, in that version of UNCL not denoted by the equal-sign.

UNCL does not perform any implicit conversion between data of different types. Every operation performed by UNCL is one which the user has specified.

Unlike the keywords of PL/1. the keywords of UNCL are reserved. This prevents opaque constructions like PL/1's „if if then then = if;".

Unlike FORTRAN, BASIC and APL, UNCL does not use numbers as its primary means of referring to program statements. Instead, the student is able to utilize mnemonic labels.

Like Algol, APL, LISP [McCarthy et al. [1966], and MAD [Arden et al.[1965]], but unlike FORTRAN, BASIC, SPL, and PL/1 UNCL has distinct notations for (1) function-application and (2) reference to elements of arrays. Like LISP and MAD, but unlike any of the other languages just mentioned, UNCL has distinct notations for mention and application of no-argument functions.

Because functions in UNCL may return functions or arrays as their values, strange expressions are conceptually possible.

Instead of

```
set x = f.(3)
set y = x.(2)                    (3)
set z = g.( )
set w = z (5)
```

where the first three lines show function-application and the fourth shows an array-reference, we might allow

```
set y = (f.(3)).2
set w = (g.( )) (5)              (4)
```

Constructions like those of (4) are actually allowed in LISP and PAL [EVANS [1968]]. They are forbidden in UNCL -- the inconvenience of (3) is more than repaid in readability.

UNCL provides data-directed output as the standard means of output, and the results of this output (e.g.,

```
x = 3
y(5) = true
A = array of 0, 2-dimensional    (5)
A(1,1) = 3.5)
```

are always acceptable as input. When collimated output is desired, UNCL uses picture formats. The use of pictures is uniquely well-suited to the on-line environment, where the program input device is the same as the data output device. In this environment, the correspondence between pictures and the associated output lines is immediately evident.

2.2 *Principle #2: So that students may efficiently find their errors, these must be localized to the narrowest possible context*

Errors (indeed, all programming) must be restrictable to subsets of the language. A student must not, for example, be able to blunder into sophisticated errors involving obscure (to him) features of the system.

These restrictions are imposed in UNCL by including a numerical „lock-level" as part of the interpreter's knowledge of its user. UNCL's input scanner is controlled by top-down syntax tables, and each construction of UNCL has a lock-level of its own. If the user's lock-level is less than the lock-level of some construction, then he cannot use that construction.

UNCL's forty or fifty lock-levels are so arranged that whatever the lock-level of the user, the language accessible to him is a coherent one, without dangling, incomplete features.

In the batch-processing environment, the human cost of waiting for another run may be large. In such an environment, there may be good reason for a language processor to make assumptions, when presented with erroneous input, as to „what the user must have had in mind." These assumptions will usually be wrong, but when they are right they may save the user considerable ennui [Freeman[1964], Morgan [1970]].

In an on-line environment, the above argument is without force, and the user's on-line file storage may be endangered by executing miscorrected programs. In rejecting automatic program-correction, UNCL rejects not only the heroic measures of CORC, but even the default attributes and conversions of PL/1.

By demanding the use of keywords, much of UNCL is highly redundant, so that errors may with some certainty be accurately described (e.g., „faulty left-hand side of SET statement"). Other portions of UNCL are as irredundant as all-purpose infix notation always is, and simple diagnostics concerning such input are rarely helpful. On receipt of erroneous input, UNCL's input scanner unwinds its own recursion, producing diagnostic messages at several (typically 3 or 4) levels of specificity. Experience seems to show that the middle-level messages are most useful.

Semantic programming errors must be localizable to the smallest possible unit of action.

UNCL's READ statement fetches only one value; if it fetched more, the situation following errors in input format might be difficult to describe.

UNCL's mechanism for creation of local variables does only that; it does not complicate matters by giving initial values to them.

Unlike JOSS, where an array's number of dimensions is fixed incidentally when the first value is assigned, the number of dimensions of an UNCL array must be explicitly specified in an array-creating assignment.

Following an execution error, the offending statement must be specified in the diagnostic messages. This specification should not be an ugly, non-mnemonic account of the statement's position on the printed page. Instead, it should locate the statement in terms of such labels as the student may have used and, in any event, in terms of the block structure of his program.

In UNCL, a statement is specified by the name of the function in which it occurs, followed by one or more **steplocators**, each of which is a label or a step number within a block.

2.3 *Principle #3: Languages for student use must be based upon a small number of clear ideas*

In UNCL, as in LISP but as in no other widely-used language, a function is an object. Functions can be manipulated just like any other values, so that

```
set f = function of x
```

```
    return sqrt. (x) + sin. (x)
  end
  if simple_sqrt_wanted then f = sqrt
  type f. (3/4)
```

is good UNCL as (suitably transliterated) it is good LISP. When a function is a well-defined object instead of, say, a random collection of lines (as in BASIC or TCL (Moore and Main (1968))), the student's intuitions regarding argument binding, etc. can be borrowed from his background in mathematics.

The virtues of referential transparency (approximately call-by-value, in the sense of Algol) have been presented by Landin (1964 and 1966) and, in a somewhat broader context, by Quine (1960).

In an on-line programming structure subject to dynamic changes by the user, call-by-name or call-by-reference would sometimes pose the problem of having a parameter pointing back to a name or reference which the user, by action at some deeper level of recursion, has deleted.

On the other hand, UNCL's strict discipline of call-by-value might be expected to founder when arrays are handled. If, when an actual parameter had an array as its value, the UNCL interpreter copied the whole array, then UNCL would indeed be foundering. Not only would the implementation cost of array-valued parameters be exorbitant, but ordinary functions like

```
clear_first_cell = function of a
    a(1) = 0
    return
  end
```

would be impossible.

In fact, UNCL does not copy arrays given as actual parameters, nor does it copy functions given in the same context. Instead, what is passed is a copy of the **header** which moors the array or function in question.

This is not a pair of ad hoc devices created for this problem. Rather, functions and arrays happen to be the only examples in UNCL of **objects with modifiable substructure**. Such objects should never be passed by copying their detailed structure, since this may be dynamically changing. Instead, a system of headers is the natural technique.

Because data-directed output is standard in UNCL, it is easy to create a file consisting of lines like those of (5). Such a file has a natural interpretation as a sequence of SET statements.

Other sequences of statements may be found (a) by reading from the console, (b) in blocks similar to those bounded by **begin** ... **end** in Algol, or (c) as the bodies of no-argument functions.

In UNCL, these four cases are handled by one conceptual mechanism. Any of the statements

```
DO FILE sss
DO CONSOLE
DO BLOCK
  ....
  END
DO function_name
```

increases the depth of recursion and then causes execution of statements from the named source until either (1) the end of the source is reached, or (2) a RETURN statement is executed. At that time, the recursion depth is reduced and execution of the DO is complete.

Execution errors (including user interrupts) in UNCL are handled by a simple mechanism. A diagnostic message is printed and then:

(1) If the error occurred during execution of a statement taken from the console, then execution of this statement is simply concluded.

(2) If the error occurred during execution of a stored or filed statement, then UNCL increases the recursion depth and listens for statements from the console. When the user RETURNs from this listening, the erroneous statement (or its user-provided replacement) is restarted.

3.4.Principle #4: A language for use by students should facilitate good programming practice. In particular, the language may be designed so as to make certain types of bad practice impossible.

UNCL stores **comment** statements, and of course all UNCL programs shown to students are appropriately commented.

Unlike the editing facilities of APL and BASIC, the editing facilities of UNCL are of the modern, context-directed variety (Deutsch and Lampson (1967)).

UNCL allows the student to store his programs and data in named, device-independent files.

UNCL provides no means of tracing the assignments to and use of selected variables. Instead, UNCL allows the student to trace the arguments and values of selected functions. This means of tracing encourages modularization into short functions to which explicit arguments are given.

Although programs need not be typed in to UNCL with indentation reflecting their block structure, they are never typed out without such indentation. Similarly, even when the SET keyword was optional during input of assignment statements, it always appeared when such statements were typed out.

REFERENCES

ANGER, A.L., Bushkin, A.A., Clark, D.D. and Coffman, J.R., 1969. The Programming Language PL, Massachusetts Institute of Technology.

ANONYMOUS, GE-635 Fortran IV Reference Manual, CPB-1006, General Electric Company, 1964.

ANONYMOUS, PL/1 Reference Manual, C28-8201-1, International Business Machines Corporation, 1968.

ARDEN, B., Galler, B., and Graham, R., 1965, The Michigan Algorithm Decoder (University of Michigan, Ann Arbor).

BERRY, APL/360 Primer, International Business Machines Corporation, 1969.

BRYAN, G.E., and Smith, J.W., The Joss Language, Memorandum RM-5377-PR, The RAND Corporation, 1967.

DEUTSCH, L.P., and Lampson, B.W., 1967, CACM 10, pp 793-799, 803.

EVANS, A., Jr., 1968, ACM Conference Proceedings 23, pp 395-403.

FENICHEL, R.R., Weizenbaum, J. and Yochelson J.C., 1970, CACM 13, pp in March issue.

FREEMAN, D.N., 1964, AFIPS Conference Proceedings 26, pp 15-34.

KEMENY, J.G., and Kurtz, T.E., 1966, BASIC, Dartmouth College.

LANDIN, P.J., 1964, Computer Journal 6, pp 308-320.

LANDIN, P.J., 1966, A d Calculus Approach. In : Advances in Programming and Non-Numerical Computation (Pergamon Press, Oxford), pp 97-141.

McCARTHY, J., Abrahams, P.W. Edwards, D.J., Hart, T.P., and Levin, M.I., 1966, LISP 1.5. Programmer's Manual, 2nd ed. (M.I.T. Press, Cambridge).

MOORE R.K., and Main, W., 1968, AFIPS Conference Proceedings 33, pp 193-200.

MORGAN, H.L., 1970 CACM 13, pp 90-94.

NAUR, P. (ed.), 1960, CACM 3, pp 299-314.

QUINE, W.V.O., 1960, Word and Object (Technology Press, Cambridge), §30.

THE TEACHING OF NUMERICAL ANALYSIS AT A COMPUTER TERMINAL

G. J. Makinson

1. INTRODUCTION

A program suite has been developed to enable a student to further his understanding of Numerical Algorithms by communicating with a computer in a conversational manner via a terminal. Practical experience of using algorithms is essential for the understanding of numerical analysis, but to obtain experience using a desk calculator, which is the usual way, is a laborious process and errors are easily made.

An alternative is to program the algorithms and test them on a computer through a batch processing service. Whilst this is very educational, it is also very time consuming, and hence the number of algorithms which can be examined in this way is very small. By incorporating tested algorithms into a program suite which can be run on line instantly without involving the user in any programming, the student can learn the basic features of many algorithms in a short time.

A student is guided to a particular algorithm in the course of a dialogue with the computer system. Having made his selection the student is asked to choose from a number of test functions and to feed in test data in a prescribed manner. The algorithm, in the form of an ALGOL program, is then executed and the results communicated directly to the student. Programs can be run interactively so that data can be fed in, in the light of the algorithm's progress. Repeat runs of the algorithm with new test data and runs of an alternative algorithm can be carried out without recourse to the whole selection process. Only a small amount of textual information is conveyed to the student at any stage. Full descriptions of the algorithms and suitable tests to apply are supplied on a printed guide to the system. If a visual display were available this material would be incorporated into the system. However the slow rate of producing printed copy makes the idea not feasible at present.

2. COTAN SYSTEM

The suite runs under the control of the COTAN multi-access system which is available on KDF9 computers with suitable configurations see Poole (1968) Cotan 3 Service Manual.

The KDF9 computer is connected to a small PDP8 computer which acts as a multiplexor for a number of teletype terminals. The user is allowed to type in a whole string of input which is collected in the PDP8. The main computer is then asked to suspend its background job to attend to the user's requests and usually all the requests will be serviced without further interaction. Thus the COTAN system could be said to be interactive only when necessary.

Entry to the COTAN system is by a special pass sequence which guards against unauthorised use. To enter the teaching package it is first of all necessary to enter the COTAN system and so a universal password has been arranged. This allows anyone to enter but the user is given only a restricted number of facilities. This limits the danger of files being contaminated inadvertantly.

The system enables a user to input, amend, store, or manipulate data by means of built in commands. Files are stored permanently on a disc, with back up facilities on a magnetic tape archive system. Some of the comments call for responses by issuing prompts. In the listing of a file the prompts are 'From? ' and 'To? ' which enable the user to specify just that part of the file which he wants listed. An exceptional command, on which the teaching package relies, is called ALGOL. On typing this the main computer suspends its background job, inputs the program specified by the user, compiles it, executes it, outputs the results to the teletype and then resumes its suspended job.

There are restrictions on the size of the program and the length of the run that can take place using this command but the programs in the teaching package are not seriously hindered by these restrictions.

The teaching package makes a great deal of use of a COTAN facility, to build up macros which are files consisting of commands. They can be stored and manipulated as data files but control can be switched to them and then they are obeyed as a series of separate commands. Macros can generate further macros and they can also be stored and then obeyed an unlimited number of times.

Many users can have simultaneous access to the COTAN system even using the same password. This makes the teaching system accessible by a whole class simultaneously. A difficulty arises if data is to be stored permanently as users with the same pass will be competing for the same space on the disc and overwriting might take place. The teaching package has been designed so that when in operation no new disc files are created and so this danger is eliminated. COTAN allocates a temporary working space to each user and thus any relevant information can be retained without danger of corruption from other sources.

3. STRUCTURE OF THE TEACHING PACKAGE

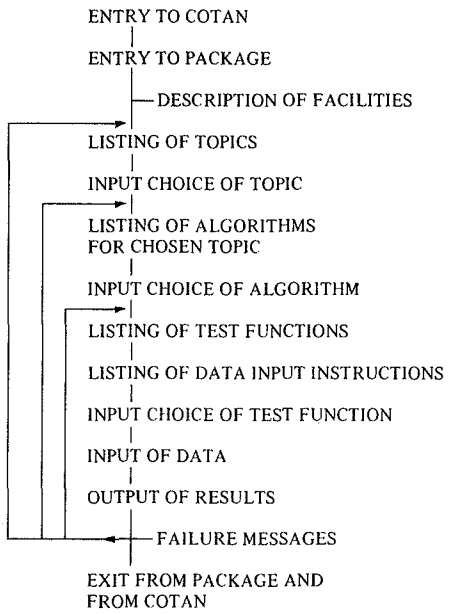
The flow chart fig. 1 outlines the steps that the package performs. Some sections are optional. The 'Description of Facilities' section is only examined by a newcomer to the package and it explains, amongst other things, the use of the teletype keyboard, the correction facility, the notation adopted, the suppression of print out facility and other points which need a fuller description than is possible in the main sections of the package.

All the listings are performed automatically and the amount of typing required by the student is kept to a minimum. The 'Failure Messages' section gives information as to how the user might proceed when a failure has occurred. This can happen for instance if the student has initiated a computation which runs longer than the time allowed or maybe occupies too large a space.

The amount of choice that the structure allows is quite large.

FIGURE 1.

Structure of the Teaching Package



A student can select a topic from a list of topics, then select a method from a choice of methods, and then select a test function from a list of test functions. Finally he is given a free hand to select the appropriate data for the actual run to be performed.

4. PROGRAM SUITE

The areas which have been included so far are,
i) Real roots of an equation.
ii)Solution of linear equations by direct and indirect methods.
iii) Quadrature.
iv) Interpolation.

There are many possible methods of finding the roots of any given equation. Six methods have been included in the suite. The standard test functions that are supplied include a simple cubic polynomial equation and an equation involving trigonometric functions. Using the system it is extremely easy to compare the rates of convergence of the six methods on the same test function and of course this will show that some methods fail to converge in certain circumstances. The behaviour of the iterations when the limit of accuracy is reached can also be shown very vividly. Such a feature is not easy to demonstrate on a desk calculator because of the large number of calculations involved.

Ill conditioning is an important aspect of linear equations. This system enables the student to observe how, in the case of an ill conditioned system, very difficult solutions are obtained with sets of data which differ only slightly. The importance of various techniques such as interchanging of rows can be shown by running comparison programs, some including the technique and others not.

In numerical quadrature a comparison of methods can be made. Other points can be indicated such as the fact that there is agreement to n decimal places in successive estimates based on different intervals does not mean that the n decimal places in the iterate are correct.

Proofs of various properties which are complicated to derive are often omitted in a course although the properties are sufficiently important to warrant inclusion. Students are much more willing to accept the statements of the properties if they can verify the validity of them in a number of cases. The non-convergence of interpolation for certain functions over special ranges is one such instance and the student can be told how to verify this by using the programs available.

5. STUDENT REACTIONS

In general those students who have used the package have been favourably impressed. The early versions of the System were tried out on M.Sc. candidates who had no previous knowledge of the subject but were attending an introductory course on numerical analysis. It is difficult to assess how successful the scheme has been in teaching them as no tests have yet been tried. However it is clear that they have enjoyed using the system.

The simplicity of operation and possibly the novelty has encouraged them and all were eager to try out as many algorithms as possible. Some attempts were made of course to break the system in order to see how it would stand up to accidental mistakes. COTAN is well constructed to deal with invalid requests for instance. If non existent filenames are used then the user is informed that the file cannot be found and the command begins afresh asking for a new filename. This facility carries over to the teaching package which is thus quite robust as far as misuse is concerned. It is not possible to leave the system satisfactorily at any arbitrary stage. An opportunity occurs after every run of an algorithm and this comes round very frequently.

Experience of using the system indicates that students quickly become irritated at receiving redundant information, a feature which is unfortunately only too common in a batch processing service. Steps have been taken to remove all irrelevant output. COTAN already has the means to suppress success messages for most commands. Typing a decimal point before the command works in most cases. For other commands it has been necessary to use an empty file as a 'dustbin' in such a way that the student is never informed of a file's disposal.

The response time of the system is not as quick as one would like but it is sufficiently fast for students not to become impatient. On the suppressed commands the teletype 'chatters' without printing and this is reassuring in that it indicates that work is proceeding.

6. FUTURE DEVELOPMENTS

It is felt that the scheme has already proved itself to be a worthwhile teaching aid. The slow speed of the teletype restricts severely the amount of information which can usefully be printed at any stage and puts a big responsibility on the programmer to produce statements that are short clear and unambiguous.

The COTAN system allows for a higher degree of interaction than has been used so far. An Algol program may suspend itself in mid stream and ask for more data. The binary program is dumped and later resumed when the new data has been received. This method of operation entails rather large overheads and its use has been discouraged. However for some problems it will afford the student the possibility of including a visual monitoring into the numerical process. An example where such an inspection would be useful is in trying to decide whether a function is sufficiently smooth to be used in some other numerical process such as interpolation. In this situation it would be advantageous to see the difference table before proceeding.

An extension of the package which is in many ways desirable, is for the student to be allowed to input his own test functions. This is certainly possible but brings with it the difficulties of teaching the correct form of input and also of dealing with the many different types of mistake that would occur.

The structure of the package shows its basic simplicity and it is clear that it could be used to help teaching many other subjects. Numerical analysis because of its need for computational power takes most advantage of the link to the main processor but the information retrieval properties of the scheme could be exploited much more, especially when a visual display unit becomes available.

REFERENCES

Poole P.C. (1968) A user's guide to COTAN.
Computing and Applied Mathematics Group, Culham Laboratory, United Kingdom Atomic Energy Authority.

LEARNING ABOUT COMPUTERS VIA AN INTERACTIVE PROGRAMMING LANGUAGE

R. S. McLean and L. D. McLean

1. INTRODUCTION

It is widely suggested that exposure to computer concepts and applications is desirable for nearly all public and secondary school students. The belief seems to be that the computer will play an increasingly important role in the shaping of lives and environment and that, therefore, all students should have some firsthand experience with the computer. The most common way of obtaining this experience is through actual programming by the student, but learning a programming language such as FORTRAN does not quarantee any insight into the computer and its possibilities. In particular, interactive use of computers is difficult with such languages.

More and more, experience with computers will be obtained earlier in one's schooling and will occur with less emphasis on the mystique of computer use. Computers will soon appear in the primary grades where they will be used as an adjunct to other curricular areas such as reading, mathematics and science. As computation becomes more economical and widespread, computer education will be accepted in the earlier grades as a tool and learning aid.

It seems to us that for this to be worthwhile, much thought should be given to the experiences that will accomplish this exposure most effectively. In particular, the programming language used must have features that facilitate early learning and use, yet provide a richness that can hold the interest of students who want to explore more complex problems and make use of the power of modern computers. The language should be conceptually current, in that it should take account of the development of computer systems, hopefully indicating the future state of the art rather than recalling the history of computation. With the increasing availability of timesharing and of small computers, direct human interaction with a computer may be expected to increase; interactive construction, debugging, and use of programs is likely to occupy a significantly greater place in computation when today's students leave school. Their computer training should reflect this.

There are already a number of small computers in schools for instructional purposes; many of them are Digital Equipment Corporation PDP-8's, and offer an interactive language, FOCAL[1], which has some of the qualities we have envisioned above. The language, a descendent of JOSS (Shaw, 1964), is implemented as an interpreter and is accessed through a teletype as an on-line terminal.

This paper discusses some of the reasons that suggest the use of this language for student programming. This is not meant as a promotion of the language, but a discussion of features of student programming based on a concrete example.

2. EARLY LANGUAGE LEARNING

For the young or beginning programmer or for the student who is being introduced to programming as a supplement to his study of mathematics, algebra or science, FOCAL provides a simple and straightforward feature that will give him „desk calculator" results immediately upon his first approach to the computer. Statements typed in the „immediate mode" in FOCAL are not stored but are executed as soon as the line is typed. Thus, the student can start with a simple output statement containing an expression and see immediately that the computer can perform useful arithmetic operations. For example, he can type „TYPE 1+2," to which FOCAL will reply „ 3.00000," introducing the normal format in which outputs from FOCAL will appear. This result is immediately verifiable by the student and shows him that he can write statements that are comprehensible to the machine. The concept of a variable and the storage and replacement of values in variables can be introduced through the immediate mode. The student can experiment and verify these mechanisms for himself without the complexities of program structure.

At this point, part of the processing the computer is doing has already been removed from the direct observation of the student: the computer is using stored information that does not appear in the current statement. It may be useful to capitalize on this as an explanatory device; no program has yet been stored, but the mechanisms of storage have been introduced. When the student encounters programs that are stored and executed from memory rather than from the teletype, the concept may be more easily grasped if introduced first in the immediate mode.

Nearly all features of the language can be used in the immediate mode, allowing students to experiment and verify each new point about the language as it is introduced. They can verify that changes in the order of parentheses and operators do make differences in the results obtained – and in the processes executed. Since any arithmetic expression may be included in a TYPE statement, many demonstrations consist of one line of code typed in with immediate feedback. Statements may be combined on a line. The FOR statement, for example, may be used in the immediate mode, and the one-line program

SET A=1; FOR I=1,300; SET A=A*I;

followed by a TYPE A, computes 300! without storing a program. The amount of computation that can be done in a short time is illustrated for the student by simple programs such as these, and he can experiment and discover for himself what differences the alteration of some parameters makes in the difficulty of the program for the computer. Discussion of the reasons for these differences can lead naturally to fundamental computing processes and hence to important concepts that have comprehensible examples.

Although the usefulness of the immediate mode calculation ability is likely to continue, the novelty may soon wear off, and some students will be eager to perform operations requiring stored programs. This can be introduced by use of some of the same examples, where the statements which set initial conditions of the problem are set off in separately numbered lines. The demonstration can then show that the same results occur although the visible commands are fewer – in fact probably just the word GO. The idea of a stored program can be illustrated further by changing one statement and observing the results; a statement that sets the initial conditions of the problem might be a good one for this purpose. The student can be reassured that the program is still present and that the appropriate changes have taken place by the use of the WRITE statement, which prints out all or selected parts of the program. As the student progresses, the idea of on-line editing of text may be introduced through the command MODIFY which permits the alteration of individual characters in a line without having to retype the entire line. Some of the more interested students will want to find out how the MODIFY command works.

FOCAL contains no formal subroutining mechanism with parameter passing conventions; nor does it have a formal block structure that characterizes advanced algebraic languages. These concepts may be introduced, however, by use of the line numbering scheme of stored programs. Lines are numbered with a decimal number, with one or two digits to the left of the decimal point and two to the right. The left digits denote a group affiliation of like-numbered lines, and this may be employed to obtain a block-like structure in programs. Subroutining is permitted by the DO command which specifies a line or group of lines to be executed in the midst of the current line being executed when the DO is encountered. All variables are global to the entire program, so some care must be exercised but this is likely to be of more concern to those who have learned another language previously.

Plotting of functions and other results of computations is facilitated by two functions that set the x and y coordinates of a point on a display, if the student's equipment configuration has an oscilloscope. If not, there are interesting plotting techniques available for the teletype. The TYPE command doesn't provide a carriage return or new line unless instructed to by a special character (!) so that one may type spaces within a FOR loop, one per pass through the loop. The placement of a character on the graph thus depends upon the number of passes through the FOR loop. This is determined by the upper limit provided in the FOR statement, but this limit can be an expression – in particular, the function being graphed (plus some normalization if necessary). With a little cleverness, a one-line program can produce the graph of a function as shown in figure 1. This kind of output can be a useful teaching aid and even a motivator. Students, at least for a while, appear to be fascinated by graphing capabilities of a terminal, and FOCAL has features which make graphs relatively easy to produce.

Perhaps one of the most important features of interpretive and interactive languages – and particularly of FOCAL – is the way in which immediate commands and a stored program can be combined in debugging. The use of the immediate command to set the initial conditions of variables and to cause execution of portions or all of a program is important. It also allows the inspection of results of execution of the stored program. All of this is accomplished in an interactive dialogue between user and computer, which gives an exciting and meaningful „trouble-shooting" or diagnosis flavor to the debugging process. This is extremely important in teaching the concept of a „bug" as something real that can be found by examining its effects and reasoning logically about their antecedent conditions.

3. THE ADVANCED USER

If a student is introduced to programming by means of FOCAL, will he find the language useful and interesting after he has mastered it? Is there sufficient richness to encourage his use of it for more complex problems? No detailed analysis of the characteristics that might make it useful at higher levels will be attempted here, but evidence that the language is useful – even by adults and advanced researchers – is found in Digital's user group library. The most recent catalog (Digital, 1969b) of programs in the library lists many FOCAL items. Examples include analysis of variance, grade averaging, polynomial curve fitting, matrix manipulations, and applications in surveying, seismic refraction and instrumentation calibration. The authors include both Digital's own programmers and research scientists in diverse academic and industrial positions. That these programs have been contributed to the library indicates that their authors consider them to be nontrivial.

4. DRAWBACKS OF THE LANGUAGE

There are ways in which FOCAL may be disappointing to the advanced user or fall short of illustrating critical computer concepts. The lack of facilities for multidimensional array storage and external files of data are of primary concern.

Without claiming any long-run merit for the limitation of arrays to one dimension, it is fair to suggest that the manual's illustration of a method for linearizing multidimensional arrays may have some educational value when done or demonstrated for the first time. The serious user is likely to find this a limitation.

All input and output on most versions of the language is limited to the teletype. An exception to this is the facility for the storage of parts of programs using the LIBRARY commands where the language is implemented on larger machines. This will not limit the user with computer-bound procedures, but for the user with a pool of data to be analyzed by many programs this is quite important. Expansion of capabilities to include bulk input/output would not constitute a major problem, although many users would then find the array restrictions confining.

5. IMPLEMENTATION AND AVAILABILITY

One of the prime reasons that FOCAL has been considered as a language for student programming is that it is available on several small and relatively inexpensive computers. The PDP-8 series of computers has already enjoyed a long life in the product line and shows no sign of fading from the small computer scene in the near future. The machine is now available in Europe, Australia and Japan as well. FOCAL is also offered on the larger PDP-9 and PDP-15.

The interactive features of use of a terminal to access a

computer, rather than the batch procedures used in many schools today, seem worth teaching. Surely interactive computing will increase rapidly, and it is likely that the world in which today's students will live will be characterized by many types of interactive access. FOCAL is a language that takes advantage of interaction and provides at least token amounts of on-line editing of materials, providing a useful environment for learning about computers. Anyone who has worked with students will acknowledge the motivational and interest-arousing properties of a terminal that is on-line, and the educational problem becomes one of sustaining interest. A language which can be used immediately and is capable of complex development is a crucial requirement.

An obvious alternative language having many of the desirable characteristics noted above is APL (Iverson, 1962). It is available on System 360 Model 50 and larger IBM computers and a subset is available both on the 1130 and on the 1500 CAI system. Debugging by executing parts of programs is easier in FOCAL, while numerous complex operators give APL considerable power. Zinn (1969) provides further comparisons of FOCAL, APL and other languages.

The costs of a FOCAL terminal and appropriate PDP-8 computer vary. A single terminal operating from a PDP-8L can be acquired for about $10,000 (U.S. prices); this falls to about $7,000 per terminal for a 16-user TSS/8 time-sharing system. In the latter system, other languages are simultaneously available; the purchase price can go below $6,000 per terminal in a FOCAL-only, 7-user PDP-8I system. FOCAL is clearly available at less cost than APL at the present time.

6. CONCLUSION

We have attempted to explore a number of attributes that make a language suitable for student programming. Among these were ease of initial use of the language, its modularity, a richness that allows sophisticated use as well as interactive features that make its execution understandable and make it easily debugged. In addition, it should reveal the future state of the art while remaining relatively inexpensive.

REFERENCES

DIGITAL EQUIPMENT CORP., 1969a, Introduction to Programming (Digital Equipment Corp., Maynard, Massachusetts), ch. 9

DIGITAL EQUIPMENT CORP., 1969b, DECUS Program Library Catalog (Digital Equipment Computer Users Society, Maynard, Massachusetts).

IVERSON, K. E., 1962, A Programming Language (Wiley, New York).

SHAW, J. C., 1964, JOSS, A Designer's View of an Experimental On-Line Computing System, Proceedings, AFIPS 1964 Fall Joint Computer Conference, vol. 26 (Spartan, Baltimore), pp. 455-464.

ZINN, K. L., 1969, A Comparative Study of Languages for Programming Interactive Use of Computers in Instruction, Final report, ONR Contract N00014-68-C-0256 (University of Michigan, Ann Arbor).

[1] A registered trademark of Digital Equipment Corporation.

```
  WRITE ALL

C FOCAL V3A
04.20 COMMENT   GRAPH AFTER DIGITAL  (1969)
04.40 FOR I=0,1,30;TYPE "*",!;FOR J=0,20+15*FSIN(I)*FEXP(-.1*I);TYPE " "

*>DO 4
*
```

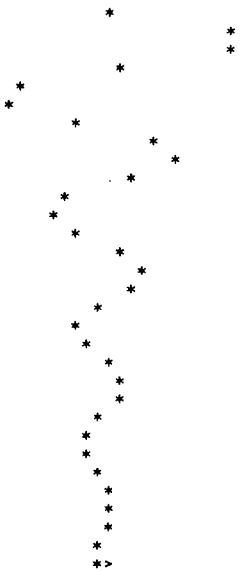


Figure 1

A GROWING INFORMATION PROCESSING SYSTEM FOR EDUCATIONAL PURPOSE

Alfonso Miola
Gian Vittorio DiGiorgio
Signorina Anna Terribile

1. INTRODUCTION

In the past few years, the fast increase in computer usage has brought serious problems in training computer users and operators and, moreover, the time employed in this instruction is quite considerable. The subject should be brought in contact with the computer as soon as possible; continuing his education by means of direct trials and experiments. This is the only way to become qualified. The computer itself is required to assist the beginner during his education.

The subject we wish to deal with is teaching Assembler language with the assistance of the computer.

We think that the knowledge of the actual way the computer proceeds in operations is essential in order to solve several types of problems, if the use of special and sophisticated languages can avoid these problems. Assembler language contains all the elements of programming and also, in its study, the beginner can develop a correct habitus for handling all future problems.

This teaching program is executed on a simplified machine, because actual complex operations could impair the process of learning. But, at the same time, all the elements of the program will be logically consistent and self-contained in order to pursue the end we have described. In order to learn about computer operations, it is necessary to explain the results of each operation (instruction to the computer and its execution).

The machine procedure we simulate is to be clearly understandable, so it will be possible to follow the internal configuration at each step. With this characteristic, we can achieve several results in an efficient and fast way. Assuming this possibility, the student will be brought gradually into contact with the actual computer and Assembler language.

The structure of this software system is built in two levels. At the first level, the hardware behaviour is simulated with an absolute octal language.

At the second level, we have a symbolic assembler language with a larger number of instruction, but, always limited. This level requires the introduction of the assembler with the approach to compilation problems.

By increasing the number of instructions and enlarging the hardware simulation of the computer, the system develops to the size of an actual assembler. The system is programmed in FORTRAN in order to use it on different computers. At present, we are studying the input-output problem for a third level of this system.

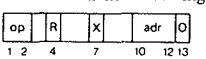
2. FIRST LEVEL

The simulated hardware features are 1000_8 36-bits of storage words, a program address register, a current instruction register and six control registers, that can be used as accumulators and index registers.

The basic instruction word consists of an octal function code, accumulator designator, index register designator and operand address. The indirect addressing is not available at this level.

The instruction repertory is now quite reduced, but the selected set is sufficient to achieve the solution of many programming problems. Because of our didactic interest, some hardware instructions will be introduced at the next level, requiring the applicant first to produce a software routine performing the same operation. In this way, the student can understand better the structure and logic of the machine.

The card instruction format is the following:



op is the instruction octal code;
R is the number (from 1 to 6) of the accumulator register;
X is the number (from 1 to 6) of the index register;
adr is the operand address (from 0 to 777_8);
0 is an option with the following meaning:

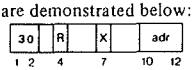
- * causes the printing of the initial and final contents of the location used by the instruction;
- $ dumps all used storage after the execution of the current instruction.

Note that all programs at this level are loaded in core beginning from the location 10_8.

In the following description:
(U) means the content of the word of the location u;
(X) means the content of the index register, and if this register is missing it means zero;
U means the address adr + (X).

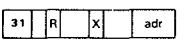
The available instructions are demonstrated below:

LOAD – 30 or



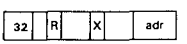
(U) is loaded in to R.

STORE – 31



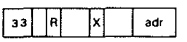
(R) is stored in to U.

ADD – 32



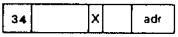
the value (R) + (U) is loaded in to R.

ADN – 33 (add negative)



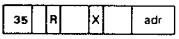
the value (R) – (U) is loaded in to R.

JUMP – 34 (unconditional)



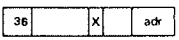
this instruction causes the computer to execute the instruction at location U.

LMJ – 35 (load modifier and jump)



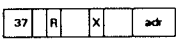
the location of the next instruction is loaded in to R and the computer executes the instruction at location U.

TZ – 36 (test for zero)



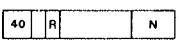
if (U) is not zero the computer executes the next instruction, otherwise the computer skips the next instruction and proceeds from there.

TG – 37 (test for greater)



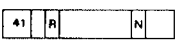
if (U)>(R) the computer executes the next instruction; otherwise the computer skips the next instruction and proceeds from there.

RSC – 40 (right shift circular)



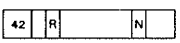
(R) is right shifted of N_8 places and as bits are pushed out of one end of register they re-enter at the other end, thus no information is lost.

LSC – 41 (left shift circular)



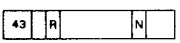
This instruction is like the RSC but the shift is on the left.

RSL – 42 (right shift logical)



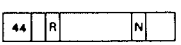
(R) is right shifted N_8 places and, as bits are pushed out, the vacated bit positions are filled with zeros.

LSL – 43 (left shift logical)



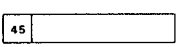
This instruction is like the RSL but the shift is to the left.

RSA – 44 (right shift algebraical)



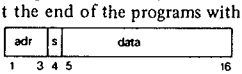
(R) is right shifted N_8 places and, as bits are pushed out, the vacated bit positions are filled with the first bit of (R); the sign of (R) is not lost.

STOP – 45



This instruction causes the computer to halt.

A card with 77 in the two first digits indicates the end of the program instructions.
As these aren't the input-output instructions, the data must be loaded as constants at the end of the programs with the following card format



data is a positive octal number $2^{35}-1$,
s is the sign,
adr is the octal location in which data, with its sign, is loaded.

A special data card with 0 in the three first digits indicates the end of the data.
At this level the only diagnostic messages are ILLEGAL OPERATION, and ILLEGAL DATA for instructions and data not conforming to the codes and to the established formats. The execution stops in consequence of an error and a dump of the core storage is produced. There are two control cards which must be used to run a program.

**AAL at the beginning to access to the first level, Absolute Assembly Language

**FIN at the end.

Let's see now an example of a first level program, which uses a special subroutine to perform the product of octal positive numbers, simulating the hardware instruction MULT not present at this level.
We represent the proposed problem with the following diagram where A, B, C are data:

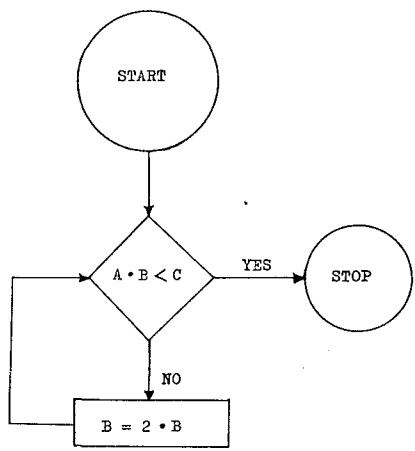
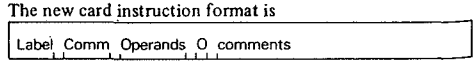


fig.2.1

A listing of the running program follows (see pages III/190 - 196)

3. SECOND LEVEL

The second level is a symbolic assembly language with an increased set of hardware instructions and an assembler that provides for the translation of the mnemonic codes and the assignment of core locations to the labels and variables. At this level there is also a diagnostic system to assist the user during the debugging.
The new card instruction format is



Label is the possible label of the instruction;
Comm is the mnemonic code of the instruction;
Operands means the registers and the variable; this field may contain three subfields separated by commas, blanks are not allowed.

In the following description:
Ri is the accumulator register indicated with i=1,6
Var is the name of the variable
X is the index register indicated with

Note that U now means the address of the variable Var plus (X), and that R+1 indicate R(i+1).
At this level, the available instructions are those of the first level plus the following:

MULT R,VAR,X
(R) is algebraically multiplied by (U), the most significant 36 bits of the product are stored in R, the least significant 36 bits of product are stored in R+1.

DIV R,VAR,X
The contents of R and R+1 are algebraically divided by (U), the quotient is stored in R, the remainder is stored in R+1.

SZ VAR,X (store zero)
Zero is stored in U.

TE R,VAR,X (test for equal)
If (R)=(U) the next instruction is skipped.

TL R,VAR,X (test for less)
If (U)<(R) the next instruction is skipped.

OR R,VAR,X
(U) is compared bit by bit with (R) to form a logical sum in R+1.

AND R,VAR,X
(U) is compared bit by bit with (R) to form a logical product in R+1.

NOP (no operation)

A special instruction RES permits to load the data:

RES VAR,M,N
Reserves M words, beginning from the address of the variable VAR, and loads them with the decimal constant N.

The diagnostic messages are
*1 Double definition for a label
*2 Syntactical error
*3 Undefined instruction code
*4 Memory overflow
*5 END card is missing
*6 STOP instruction is missing

The control cards of this level are
**SAL at beginning to access to the second level, Symbolic Assembly Language;
**EXEC at end of program cards to execute the assembled program;
**FIN at end.

Let us see now an example of a second level assembly program performing the ten first prime numbers.
(See page III/197)

REFERENCES

Flores I., 1965, Computer Software, (Prentice-Hall, INC, Englewood Cliffs, N.J.).
IBM 7040 Data Processing Systems, Student Text, 1963, (IBM).
Knuth D.E., 1968, The Art of Computer Programming, Vol. 1/ Fundamental Algorithms, (Addinson-Wesley Publishing Company).

```
**, THE GROWING SYSTEM FOR EDUCATIONAL PURPOSE - GSEP - HAS BEEN ENTERED **

NOTE

 LOAD  30          STOR  31          ADD   32          ADN   33          JUMP  34

 LMJ   35          TZ    36          TG    37          RSC   40          LSC   41

 RSL   42          LSL   43          RSA   44          STOP  45          END   77

LISTING OF RUNNING PROGRAM
          **AAL
          30 1     100*
          30 2     101*
          35 3      22*
          37 1     102*
          34        16*
          45
          30 1     101*
          32 1     101*
          31 1     101*
          34        10*
          30 4     205*
          31 1     200*
          30 1     203*
          31 2     201*
          43 2      43*
          42 2      43*
          31 2     202*
          36       202*
          32 1     200*
          30 5     200*
          43 5       1*
          31 5     200*
          30 2     201*
          42 2       1*
          31 2     201*
          33 4     204*
          31 4     206*
          36       206*
          34        26*
          34    3  000*
          77
          100+        23
          101+         3
          102+       100
          200+         0
          201+         0
          202+         0
          203+         0
          204+         1
          205+         3
          206+         0
          000
          **FIN
```

Tab. 2.1

DUMP OF USED MEMORY

	0	1	2	3	4	5	6	7
000	000000000010	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
010	300100000100	300200000101	350300000022	370100000102	340000000016	450000000000	300100000101	320100000101
020	310100000101	340000000010	300400000205	310100000200	300100000203	310200000201	430200000043	420200000043
030	310200000202	360000000202	320100000200	300500000200	430500000001	310500000200	300200000201	420200000301
040	310200000201	330400000204	310400000206	360000000206	340000000026	340000300000	777777777777	777777777777
050	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
060	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
070	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
100	000000000023	000000000003	000000000100	777777777777	777777777777	777777777777	777777777777	777777777777
110	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
120	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
130	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
140	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
150	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
160	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
170	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
200	000000000000	000000000000	000000000000	000000000000	000000000001	000000000003	000000000000	777777777777

Tab. 2,2

LOC	COMM	REG	OPER	OPTIONAL DUMP OF USED WORDS		
010	30	1	100			
				(R1)=777777777777 (OPER)=000000000023		I. CONF.
				(R1)=000000000023 (OPER)=000000000023		F. CONF.
011	30	2	101			
				(R2)=777777777777 (OPER)=000000000003		I. CONF.
				(R2)=000000000003 (OPER)=000000000003		F. CONF.
012	35	3	022			
				(R3)=777777777777	(CI)=013	I. CONF.
				(R3)=000000000013	(CI)=022	F. CONF.
022	30	4	205			
				(R4)=777777777777 (OPER)=000000000003		I. CONF.
				(R4)=000000000003 (OPER)=000000000003		F. CONF.
023	31	1	200			
				(R1)=000000000023 (OPER)=000000000000		I. CONF.
				(R1)=000000000023 (OPER)=000000000023		F. CONF.
024	30	1	203			
				(R1)=000000000023 (OPER)=000000000000		I. CONF.
				(R1)=000000000000 (OPER)=000000000000		F. CONF.
025	31	2	201			
				(R2)=000000000003 (OPER)=000000000000		I. CONF.
				(R2)=000000000003 (OPER)=000000000003		F. CONF.
026	43	2	043			
				(R2)=000000000000000000000000000000000011		I. CONF.
				(R2)=100000000000000000000000000000000000		F. CONF.
027	42	2	043			
				(R2)=100000000000000000000000000000000000		I. CONF.
				(R2)=000000000000000000000000000000000001		F. CONF.
030	31	2	202			
				(R2)=000000000001 (OPER)=000000000000		I. CONF.
				(R2)=000000000001 (OPER)=000000000001		F. CONF.
031	36	0	202			
				(OPER)=000000000001	(CI)=032	I. CONF.
					(CI)=032	F. CONF.
032	32	1	200			
				(R1)=000000000000 (OPER)=000000000023		I. CONF.
				(R1)=000000000023 (OPER)=000000000023		F. CONF.
033	30	5	200			

Tab. 2.3

				(R5)=777777777777 (OPER)=000000000023			I. CONF.
				(R5)=000000000023 (OPER)=000000000023			F. CONF.
034	43	5	001				
				(R5)=000000000000000000000000000000010011			I. CONF.
				(R5)=000000000000000000000000000000100110			F. CONF.
035	31	5	200				
				(R5)=000000000046 (OPER)=000000000023			I. CONF.
				(R5)=000000000046 (OPER)=000000000046			F. CONF.
036	30	2	201				
				(R2)=000000000001 (OPER)=000000000003			I. CONF.
				(R2)=000000000003 (OPER)=000000000003			F. CONF.
037	42	2	001				
				(R2)=000000000000000000000000000000000011			I. CONF.
				(R2)=000000000000000000000000000000000001			F. CONF.
040	31	2	201				
				(R2)=000000000001 (OPER)=000000000003			I. CONF.
				(R2)=000000000001 (OPER)=000000000001			F. CONF.
041	33	4	204				
				(R4)=000000000003 (OPER)=000000000001			I. CONF.
				(R4)=000000000002 (OPER)=000000000001			F. CONF.
042	31	4	206				
				(R4)=000000000002 (OPER)=000000000000			I. CONF.
				(R4)=000000000002 (OPER)=000000000002			F. CONF.
043	36	0	206				
				(OPER)=000000000002		(CI)=044	I. CONF.
						(CI)=044	F. CONF.
044	34	0	026				
						(CI)=045	I. CONF.
						(CI)=026	F. CONF.
026	43	2	043				
				(R2)=000000000000000000000000000000000001			I. CONF.
				(R2)=100000000000000000000000000000000000			F. CONF.
027	42	2	043				
				(R2)=100000000000000000000000000000000000			I. CONF.
				(R2)=000000000000000000000000000000000001			F. CONF.
030	31	2	202				
				(R2)=000000000001 (OPER)=000000000001			I. CONF.
				(R2)=000000000001 (OPER)=000000000001			F. CONF.
031	36	0	202				
				(OPER)=000000000001		(CI)=032	I. CONF.
						(CI)=032	F. CONF.
032	32	1	200				
				(R1)=000000000023 (OPER)=000000000046			I. CONF.
				(R1)=000000000071 (OPER)=000000000046			F. CONF.

Tab. 2.4

033	30	5	200			
				(R5)=000000000046 (OPER)=000000000046		I. CONF.
				(R5)=000000000046 (OPER)=000000000046		F. CONF.
034	43	5	001			
				(R5)=000000000000000000000000000100110		I. CONF.
				(R5)=000000000000000000000000000001001100		F. CONF.
035	31	5	200			
				(R5)=000000000114 (OPER)=000000000046		I. CONF.
				(R5)=000000000114 (OPER)=000000000114		F. CONF.
036	30	2	201			
				(R2)=000000000001 (OPER)=000000000001		I. CONF.
				(R2)=000000000001 (OPER)=000000000001		F. CONF.
037	42	2	001			
				(R2)=000000000000000000000000000000001		I. CONF.
				(R2)=000000000000000000000000000000000		F. CONF.
040	31	2	201			
				(R2)=000000000000 (OPER)=000000000001		I. CONF.
				(R2)=000000000000 (OPER)=000000000000		F. CONF.
041	33	4	204			
				(R4)=000000000002 (OPER)=000000000001		I. CONF.
				(R4)=000000000001 (OPER)=000000000001		F. CONF.
042	31	4	206			
				(R4)=000000000001 (OPER)=000000000002		I. CONF.
				(R4)=000000000001 (OPER)=000000000001		F. CONF.
043	36	0	206			
				(OPER)=000000000001	(CI)=044	I. CONF.
					(CI)=044	F. CONF.
044	34	0	026			
					(CI)=045	I. CONF.
					(CI)=026	F. CONF.
026	43	2	043			
				(R2)=000000000000000000000000000000000		I. CONF.
				(R2)=000000000000000000000000000000000		F. CONF.
027	42	2	043			
				(R2)=000000000000000000000000000000000		I. CONF.
				(R2)=000000000000000000000000000000000		F. CONF.
030	31	2	202			
				(R2)=000000000000 (OPER)=000000000001		I. CONF.
				(R2)=000000000000 (OPER)=000000000000		F. CONF.
031	36	0	202			
				(OPER)=000000000000	(CI)=032	I. CONF.
					(CI)=033	F. CONF.
033	30	5	200			

Tab. 2.5

				(R5)=000000000114 (OPER)=000000000114			I. CONF.
				(R5)=000000000114 (OPER)=000000000114			F. CONF.
034	43	5	001				
				(R5)=000000000000000000000000001001100			I. CONF.
				(R5)=000000000000000000000000010011000			F. CONF.
035	31	5	200				
				(R5)=000000000230 (OPER)=000000000114			I. CONF.
				(R5)=000000000230 (OPER)=000000000230			F. CONF.
036	30	2	201				
				(R2)=000000000000 (OPER)=000000000000			I. CONF.
				(R2)=000000000000 (OPER)=000000000000			F. CONF.
037	42	2	001				
				(R2)=000000000000000000000000000000000000			I. CONF.
				(R2)=000000000000000000000000000000000000			F. CONF.
040	31	2	201				
				(R2)=000000000000 (OPER)=000000000000			I. CONF.
				(R2)=000000000000 (OPER)=000000000000			F. CONF.
041	33	4	204				
				(R4)=000000000001 (OPER)=000000000001			I. CONF.
				(R4)=000000000000 (OPER)=000000000001			F. CONF.
042	31	4	206				
				(R4)=000000000000 (OPER)=000000000001			I. CONF.
				(R4)=000000000000 (OPER)=000000000000			F. CONF.
043	36	0	206				
				(OPER)=000000000000		(CI)=044	I. CONF.
						(CI)=045	F. CONF.
045	34	0	000,3				
				(R3)=000000000013		(CI)=046	I. CONF.
						(CI)=013	F. CONF.
013	37	1	102				
				(R1)=000000000071 (OPER)=000000000100		(CI)=014	I. CONF.
						(CI)=015	F. CONF.
015	45	0	000				

Tab. 2.6

DUMP OF USED MEMORY

	0	1	2	3	4	5	6	7
000	000000000016	450000000000	000000000071	000000000000	000000000013	000000000000	000000000230	777777777777
010	300100000100	300200000101	350300000022	370100000102	340000000016	450000000000	300100000101	320100000101
020	310100000101	340000000010	300400000205	310100000200	300100000203	310200000201	430200000043	420200000043
030	310200000202	360000000202	320100000200	300500000200	430500000001	310500000200	300200000201	420200000001
040	310200000201	330400000204	310400000206	360000000206	340000000026	340000300000	777777777777	777777777777
050	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
060	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
070	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
100	000000000023	000000000003	000000000100	777777777777	777777777777	777777777777	777777777777	777777777777
110	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
120	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
130	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
140	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
150	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
160	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
170	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777	777777777777
200	000000000230	000000000000	000000000000	000000000000	000000000001	000000000003	000000000000	777777777777

END OF EXEC GSEP

Tab. 2.7

MAX PAGES

```
        *** THE GROWING SYSTEM FOR EDUCATIONAL PURPOSE - GSEP - HAS BEEN ENTERED ***


STORAGE ASSIGNEMENT FOR VARIABLES(NAME,LOCATION)

      ZERO   041          ONE    042          VETT   043          TWO    055          REM    056
             057

      STEP    LOC     OCTAL                    SYMBOLIC                          COMMENT

        1*                                     RES   ZERO,1,0
        2*                                     RES   ONE,1,1
        3*                                     RES   TWO,1,2
        4*                                     RES   VETT,10,0
        5*                                     RES   REM,1,0
        6*                                     RES   X,1,10
        7*                             ANNA    STRT  010                         PRIME NUMBERS
        8*      010   300300000041             LOAD  R3,ZERO      *
        9*      011   300400000042             LOAD  R4,ONE
       10*      012   310400300043             STOR  R4,VETT,R3   *
       11*      013   320300000042             ADD   R3,ONE       *
       12*      014   320400000042             ADD   R4,ONE
       13*      015   310400300043             STOR  R4,VETT,R3
       14*      016   330400000042             ADD   R4,ONE       *
       15*      017   320300000042             ADD   R3,ONE
       16*      020   320400000055     LOOP    ADD   R4,TWO
       17*      021   300100000055             LOAD  R1,TWO
       18*      022   360000100043     TEST    TZ    VETT,R1      *
       19*      023   340000000025             JUMP  NO           *
       20*      024   340000000034             JUMP  YES
       21*      025   470400100043     NO      DIV   R4,VETT,R1   *
       22*      026   310500000056             STOR  R5,REM
       23*      027   360000000056             TZ    REM
       24*      030   340000000032             JUMP  A
       25*      031   340000000020             JUMP  LOOP
       26*      032   320100000042     A       ADD   R1,ONE
       27*      033   340000000022             JUMP  TEST
       28*      034   310400300043     YES     STOR  R4,VETT,R3
       29*      035   320300000042             ADD   R3,ONE
       30*      036   370300000057             TG    R3,X         *
       31*      037   450000000000             STOP               *
       32*      040   340000000020             JUMP  LOOP
       33*                                     END


                        ***END OF GSEP COMPILATION***
```

Tab. 3.1

COPI–II: A GENERAL PURPOSE PEDAGOGICAL SYSTEM

Robert G. Milburn, Jr. and Raymond E. Sandborgh

1. RATIONALE

1.1 *Introduction*

More than anything else, the development of the COPI System was dependent on the potential it could offer for useful research on teaching. This research was held to consist principally of two elements, a) the development of alternative computer-based instructional strategies and b) their evaluation in an environment from which reasonable extrapolation to operational situations could be carried out. Virtually all of the design decisions affecting the development of the COPI System stem from a) and b) above.

1.2 *Functional Requirements*

The former indicates that COPI System was to comprise a portion of the automated interactive, i.e., cybernetic, elements of a computer-based pedagogical system. This was taken to mean that COPI was to provide a facility for a) creating automated instructional material, b) administering and monitoring of the administration of the material, and c) reporting on a) and b). Instructional material was taken to include data which could be stored, retrieved or controlled by a computer along with algorithms, transformational processes and rules governing the storage, retrieval, display and interaction with the material by a learner.

The specific characteristics of the system, including for example, the syntax of the interactive dialect associated with the system, was determined in part by the broad functional characteristics stated above and by the specific design criteria listed in the following section.

1.3 *Criteria*

COPI was designed to facilitate research on teaching. It was expected that the great majority of users of COPI would be individuals who were not highly skilled in the design and use of complex data processing systems. The first criterion was that COPI be oriented toward teachers and researchers who typically are not data processing experts, so that they could maximize their use of the system.

A second major decision resulting in design criteria, related to the system's role as a research facility, was a decision to develop a general-purpose facility which was to be unbiased relative to any specific teaching strategy or discipline area. The implication of this decision was that COPI should have capabilities for both intrinsically logical and adaptively logical tutorial strategies, as well as strategies which were socratic. In addition, provision was made for simulation, games and for learner-controlled strategies.

The third criterion also stemmed from the research orientation. Specifically the system was to be highly modular in order to facilitate change.

The fourth criterion was that the system be a large-scale system in terms of the number and variety of terminals which could be supported concurrently. This criterion resulted in part from the need to reduce the cost of providing researchers with sufficiently large test populations. It also resulted from the need to study the system implications of such large-scale systems. Presumably, in order to evaluate and plan for the implementation of economically viable automated teaching systems a system, which at least approximates in size, the planned system, should be used as a source of data.

The previous four criteria have dealt with software. The fifth criterion dealt with hardware, specifically, the interactive terminals. It was decided that researchers in automated teaching would require terminals which made available a variety of media for output of stimuli and input of response. It was further determined that such terminals must be economic and modular so that the researcher might readily configure a terminal for a particular media mixture.

1.4 *Summary*

The COPI System Software was designed to be tool for research on teaching such that various instructional strategies could be examined in environments which modeled various actual classroom settings. The requirements (in general terms) of such a system are: to provide a means for creating instructional material, administering and monitoring the instructional material and method for reporting on the administering and monitoring of the instruction.

The five principal design criteria for the COPI System are: 1) that the system be user-oriented, 2) that it be general-purpose and 3) modular, 4) that it suppport a relatively large number of concurrent users, and lastly, 5) that the terminals be modular multi-media terminals.

2. DETAILED DESCRIPTION OF COPI-II, LEVEL B1

2.1 *Introduction*

This chapter describes in some detail the specific characteristics of the COPI-II, Level B1 System as it currently operates on the UNIVAC 1108 and 1106 Data Processing Systems. The Chapter is divided into two sections, the first of which deals with the software system and the second with the hardware associated with this particular implementation.

2.2 *COPI-II, Level B1 software*

The COPI-II, Level B1 (hereafter referred to as Level B1) software is divided into three subsystems which parallel the three functional requirements cited in ch. 2.2. These three subsystems include 1) a noninteractive preprocessor or compiler, 2) an interactive transaction oriented

processor, and 3) a noninteractive post-processor or report generator. The interactive dialect in which instructional materials are written in common to all three functional areas and will for convenience be discussed first.

2.2.1 Level B1 interactive dialect

The Level B1 interactive dialect or author language constitutes the medium by which the conversational or interactive data is specified. This data is thought of as a heirarchial data structure made up of individually addressable conversational elements which may be collected into LESSONS, and lessons which may be collected into COURSES. The distinction between lessons and courses has less significance from an educational point-of-view than from the system point-of-view. Therefore, we will concern ourselves at this point with only the conversational element.

2.2.2 Conversational element

Much of the published literature on COPI refers to the conversational element as a FRAME. In spite of an apparent and increasingly common sensitivity to the use of the term frame we shall employ it here with the note that COPI frames are principally understood as delimiting major entry points in a matrix or space composed of conversational elements. The frame is then thought of as a convenient way of describing hierarchical elements which constitute lessons and courses. The existence of secondary entry points into the conversational stream which was inferred above will be described in ch. 4 as „intra-frame" branching.

An example of a frame is given in fig. 1. This example of a simple tutorial frame contains the following basic frame elements: 1) frame label, 2) text, 3) anticipated responses and response processing logic, and 4) reinforcement texts. Should this frame be branched to during an interactive session the result would be the transmission of the text or stimulus to the remote terminal.

In this particular example, the text information beginning on line two and extending to the first double dash delimiter, on line 12, would be transmitted and displayed. For the purpose of this discussion we shall assume that the learner is interacting with the system through either a cathode ray tube display (CRT) or a teletypewriter both of which employ conventional alphanumeric keyboards. The text portion of the example includes a question. At any time after the text is received and displayed the learner may respond. In this example the anticipated response is an alphabetic string which from the learner's viewpoint is simply a word or phrase. In normal circumstances the learner will simply enter his response by depressing one or more keys on the terminal keyboard.

Upon receipt of the learner response, the second portion of the frame is referenced. In fig. 1., the second portion of the frame consists of four lines of anticipated responses and associated response processing logic. The first anticipated response, METEOROID, is followed by a number of commands which are referred to as a Logical Unit.

The logical unit associated which the first anticipated response contains three elements: 1) reference to a matching algorithm, KM, or Keyword Match, 2) a reinforcement call, PRINT, and 3) a branching directive, GOTO 125. This information is interpreted by the system

FIG. 1. An Example of a Frame Written in COPI–II, Level B1 Author Language

```
FRI24
  WE HAVE SEEN THAT ON A CLEAR NIGHT A METEOR CAN BE DETECTED
  APPROXIMATELY EVERY TEN MINUTES.  HOWEVER, DURING CERTAIN TIMES
  OF THE YEAR THEY ARE MUCH MORE NUMEROUS.  AT TIMES IT IS
  POSSIBLE TO FIND METEORS APPEARING AT A RATE OF ONE PER MINUTE,
  OR EVEN ONE PER SECOND.  THESE CELESTIAL FIREWORKS DISPLAYS ARE
  CALLED METEORIC SHOWERS, AND OCCUR WHEN THE EARTH ENCOUNTERS
  A SWARM OF METEOROIDS.

  BEFORE GOING ANY FURTHER, *NAME*, LET'S BE SURE WE HAVE OUR
  TERMINOLOGY STRAIGHT.  WE HAVE, IN THIS LESSON, USED THE WORDS
  METEOR, METEOROID, AND METEORITE.  WHICH TERM REFERS TO THE
  ACTUAL SOLID OBJECT WHILE IT IS STILL IN SPACE?--

  METEOROID--KM, PRINT A, GOTO 125--
  METEORITE--(DM, PRINT B, GOTO A21) (GOTO A28) (DO 247, GOTO 124)--
  METEOR--(DM, PRINT B, GOTO A22) (GOTO A28) (DO 247, GOTO 124)--
  --NM, PRINT C, DO 247, GOTO 124--

ACORRECT METEOROID IS THE TERM THAT DESCRIBES THE OBJECT WHILE
  IT IS STILL IN SPACE.  METEOR REFERS TO THE VISIBLE LIGHT TRAIL
  CREATED WHEN THE METEOROID IS HEATED TO INCANDENSCENCE BY THE
  EARTH'S ATMOSPHERE.  IF THE METEOROID REACHES THE EARTH'S
  SURFACE IT IS CALLED A METEORITE.--

BSORRY, BUT THAT ANSWER IS INCORRECT.  LET'S REVIEW THE TERMS.--

CONE OF US IS HAVING A BAD DAY.  LET'S LOOK AT THE PROBLEM
  AGAIN.--
```

as follows: Meteroid is the first anticipated character string or anticipated response. Test the actual response, as entered at the keyboard by comparing it with the anticipated response. Perform the comparison, or MATCH, according to the Key Word Match algorithm. If the match is succesful (in this case, the string MATEOROID would need be a subset of the actual input string) then display the reinforcement string identified by the label A and branch to the frame whose label is 125. If the match is not successful then test the actual response against the second anticipated using the associated response processing logic.

This process is then repeated until either a specified branch takes place or the anticipated responses are exhausted with no branch. In the latter case a modal convention will result in an automatic branch to the next frame.

The example given in fig. 1. exhibits four additional characteristics of the syntax of the interactive dialect. The first of these is the No Match, abbreviated as NM, found in the fourth line of anticipated responses. NM is said to be another Match Type falling in the same category as the KM previously mentioned. NM implies that if no match occurred in the preceding anticipated solutions, that the response processing directives in the logical unit associated with the NM should be processed. Level B1 match types currently include the two already mentioned as well as the following: Direct Match, a literal character by character comparison, Phonetic Match, a transformation algorithm Long [1968] for determining whether words sound alike and lastly a combination of phonetic match and key match which searches for the phonetic equivalent of the anticipated response as subset of the actual response. The planned extension of these match types is discussed in ch.

3.

A second additional characteristic is the DO statement. The DO statement is a branching command which differs from the GOTO in that GOTO results in unconditional control being passed to the referenced frame, whereas the DO statement only temporarily passes control to a frame or a sequence of frames. Following each DO statement is a related GOTO statement. The GOTO in this case specifies to which frame control will be passed when the sequence called by the DO is exited. The DO statement may then be viewed as a kind of a subroutine call. DO statements may be nested so that one subroutine may call another.

A third characteristic is termed „Multiple Logical Units" and is shown in fig. 1 in the logical units associated with the second and third anticipated solutions. Multiple Logic Units are used to provide for varation in response processing as a function the number of times a particular path is taken through a given frame. Thus in the example, should a student make the response „METEORITE" on an initial pass he will branch to Frame A21. Should he again respond with „METEORITE" on a subsequent pass he will branch to Frame A28.

The fourth and last characteristic to be mentioned is the existence of a special character string in the text portion of the frame. In fig. 1., the eighth line of text contains the string *NAME*. This string is a special case of a general capability which takes the form *a, b, c,....n* where a, b, and c, etc., represent alphanumeric characters which make up a label for an alphanumeric string whose contents have been set in previous interaction. The string is set through the use of an ASSIGN statement which permits the author to assign a symbolic name to a particular input. to then save that input and later use it as he would any other character string. Thus had the author used the ASSIGN to collect the name of the student, the student would see his name in place of the symbols *NAME*. Data entry symbolics may be used as anticipated solutions as well. A use of this type would take the form:
NAME–KM, PRINT A, GOTO A2–.

The foregoing discussion does not constitute a complete description of the syntax of COPI-II, Level B1. A complete description would of necessity include, in addition to what has been given, the following: a discussion of specialized device commands such as for the CRT or slide projector, a discussion of counters which are author set or student set, the method of testing counters, response latency controls, macro definitions, and specific qualitative and quantitative limitations on the use of these commands. A complete description of these and several other capabilities has been published separately, Milburn, et al., [1970a]..[1970b].

This completes the discussion of the Level B1 syntax.We turn now to a higher level view of the software according to the functional areas mentioned previously.

2.2.3 Level B1 Compiler

The Level B1 compiler Bucholtz [1969b] operating as batch processor accepts card image data containing instructional material coded in accordance with the Level B1 author language. The compiler essentially packs the input into fixed format, variable length files suitable for retrieval and use by the interactive system. The printed output includes the following: 1) a source listing organized by lesson within course with flags for control card errors, 2) a concise error listing organized by frame within lesson, within course, listing syntax errors internal to the frame, 3) a listing of frames referenced but not found, 4) a listing of frames found but not referenced, and 5) an index of frames, lessons and courses compiled. In addition, the compiler outputs a special source file for use by the interactive edit.

In summary then, there are three important outputs from the compiler, an object file, a source file, and a diagnostic listing. The compiler operates completely asynchronously from every other element of the Level B1 System and can thus be used to prepare instructional material while an interactive process is taking place involving previously compiled data.

2.2.4 Level B1 Interactive System

The level B1 Interactive System may be discussed from a system point-of-view and user point-of-view. We will first discuss the interactive system from a user's view point.

2.2.4.1 User Functions

The Level B1 Interactive system provides three main classes of user functions. These are 1) Student Functions, 2) Instructor Functions, and 3) Utility Functions.

A student may perform the following actions at a remote terminal under control of the Level B1 Interactive System: a) he may identify himself by name and student number, b) select a lesson or course from a glossary, c) start the lesson he has selected, d) terminate the lesson, and e) define and cause the system to perform mathematical calculations. In addition, the student may determine at sign-on time whether he wishes to begin a lesson or course

at its defined beginning or to begin at the point where he left off in some previous session. Should he select the latter, the system recalls the entry automatically and initiates the lesson at that point automatically.

The calculation mode Relles (1969b) provides the student with the ability of using the computer terminal to perform student defined computations during a lesson. In effect, the student is given a high-level desk calculator which can carry out calculations and sequence through a series of statements involving branching, conditional branching and calculation.

Statements within the calculation mode may consist of constants, variables, operators and functions. Constants may be negative or positive and are stored as floating point decimal numbers ranging from 999 999 999 999.99999 to -999 999 999 999.99999. Variables are identified as single alphabetic characters (A-Z) and are always decimal numbers. Operators for addition, subtraction, multiplication, division and exponentiation are provided, as are functions for sine, cosine, square-root, integer or truncation, and for the obtaining the absolute value.

The calculation mode permits execution of one line statements of the form:

```
R = 2.3
A = (3.1415) X (R**2)
SQT(4-20)
```

In the last statement above, the system would return the result:

4.00000*I, or 4.00000* √ -1.

The calculation mode also provides for the execution of short sequences of statements of the following form:

```
DO    R = 5
      D = 2*R
      C = π*D
      A = ((C/2*π)**2)*π
      V = (4 ÷ 3) (R*A)
      S = 4*A
END
```

Execution of this „program" would result in the following output:

S = 514.159 00, Haupert, et al.,[1970].

In addition to the above, the calculation mode provides the student with Format Control enabling him to cause the statements to remain static on the CRT or to automatically roll up after each output or input sequence. After completing a calculation a student may cause the system to automatically return to the point in a lesson from which the calculation mode was entered.

The Level B1 Interactive System provides the following Instructor Functions:1) all student functions, 2) control over student numbers, 3) privileged review of lesson material, 4) use of the interactive EDIT and 5) system termination. Access to priviledged instructor functions are protected by both sign-on procedure and by special mechanical overlays that operate in conjunction with the keyboard of the CRT.

The EDIT function may be employed only at a CRT remote terminal. EDIT allows the instructor to view selected lessons in source format, make corrections, and cause the revised lesson to be recompiled and stored by the system. At the instructors option the revised lesson or lessons may either be added to the lesson file or replace previously compiled lessons. EDIT operates completely asychronously with the remainder of the interactive system, permitting simultaneous student use and instructor edit of a given lesson without mutual interference.

Lastly, the Level B1 Interactive System provides for the transfer of lessons or courses from magnetic tape to drum or disk. This capability is useful in an experimental environment where it is not desired to maintain large permanent drum or disk files. In an operational environment lesson materials would normally be stored in rapid random-access mass storage and this feature would receive only occasional use.

2.2.4.2. System Functions

The Level B1 Interactive System is from a systems view, a real-time, transaction-oriented processor. It is composed of three principle classes of elements. These are: 1) device oriented handlers, 2) executive and control routines, and 3) applications processors.

The device handlers constitute an open list which at the time of this writing include both CRT handlers and teletypewriter handlers. Both types of devices may operate concurrently or the system may use only one or the other. The student terminals, whether CRT's or TTY's may be operated locally or remotely, on single drop or multidrop, multiplexed or not, or any combination of the above.

The applications processors comprise an open ended list which currently includes the following: a) framehandler, b) calculation mode, c) edit, and d) utility routines. Framehandler is the processor which controls the interaction between the student and the instructional materials. Effectively, each of the applications processors becomes a mode of operation as far as a student or instructor is concerned. The addition of applications processors has been provided for in order to facilitate specifically the addition by the user of specialized simulations or games which might then operate in common with and be selectable from or as a result of tutorial conversation.

The Executive and Control routines serve to link the various terminals with handlers and applications processors as required. Each terminal active in the system may be under control of a different applications processor.

2.2.5 The report generator

The third principal element of the Level B1 Software system is referred to as a report generator HAUPERT (1968). This element will be referred to hereafter as the GRADE Program (General Record Analysis and Data Evaluation). The GRADE Program operates on data which is collected during the interactive process. It is a batch program which collects, sorts, and analyzes the data in order to produce two outputs. The first output is called a „Student Progress Program Listing" and the second a „Frame Listing". Effectively the GRADE outputs describe the performance of individuals and the performance of the curriculum.

The Student Progress Listing contains a complete trace of the activities of every student who was active during the reporting period. The report lists the student's name, number, elapsed time on the system, and according to course and lesson, the sequence of frames or path that was taken. The time spent in the frame, the anticipated response which collected the match, and changes in counter values are also shown.

The Frame Listing gives statistical performance information for the education material on a frame by frame basis. Each frame referenced during the reporting period is listed according to lesson and course. In addition, a

distribution is given of the references associated with each anticipated response, Mean and median response latencies for the frame are listed also.

2.2.6 Summary statistics

The COPI-II, B1 Software System has been implemented on UNIVAC 1106 and 1108 Systems which operate under „EXEC 8". Ch. 3.3 gives a brief description of the 1106 and 1108 computers.

The entire Level B1 Interactive System requires approximately 23000 decimal 36-bit words of core storage for operation. To this must be added approximately 10 decimal words for each terminal and 563 decimal words for each line. The lesson assembler utilizes approximately 15000 decimal words of core and the GRADE Program approximately 25000 decimal words. Each of the three principle elements also makes use of drum or disk storage. The exact requirements for storage on drum or disk depend on the quantity of educational material and the number of students indexed by the system. We may note that requirements for one-to-twenty billion characters of drum or disk storage are not unreasonable and could easily be reached in an operational environment.

It has already been noted that the COPI-II, Level B1 System operates under EXEC 8. The most important implication of this is that concurrent processing of background batch, demand and other real-time processes is permitted. The total computing power available for such use depends on the number of interactive terminals requiring service and their response frequency. At the present time no specific maximum number of terminals for operation under Level B1 has been fixed. No estimate is given here as the maximum will always be a function directly dependent on the mix of terminals, line speed, curriculum characteristics and hardware configuration.

2.3 *Hardware*

2.3.1 Central Site

The COPI-II, Level B1 System is at present implimented on the UNIVAC 1108 and 1106 Data Processing Systems operating under EXEC 8. The configuration of these systems required in order to run the COPI-II, Level B1 System is that minimum configuration required for EXEC 8 operation with the addition of communications facilities. MILBURN, **et al.**, (1970a).

Both of the processing systems indicated above are fast, medium to large scale systems, capable of operating as Unit Processors or as Multiprocessors. Main memory storage is composed of up to 262 000 36-bit words. Read/restore cycle time for the 1106 is 1500 nanoseconds and for the 1108, 750 nanoseconds.

Both systems have optional magnetic drum storage units and magnetic tape units as well as punched card and printer subsystems.

2.3.2 Remote terminals

At present the remote terminals employed by the COPI-II, Level B1 System are limited to CRT's and TTY's. Specifically, either KSR or ASR 33 or 35 Teletypewriters are permitted. UNIVAC Uniscope 300 or Uniscope 100 Cathode Ray Tube Displays are permitted in either a one-per-line configuration or in a multiplexed configuration. The Uniscope 300 and Uniscope 100 are self contained Cathode Ray Tube Displays, utilizing a self-contained core memory capable of storing and displaying 1024 characters or more. Both devices employ extensive editing capabilities and make use of either an overlay or wallet sized card which has a unique identifier code associated with it and which is optionally employed by the system to control access to specialized functions. Both displays are alphanumeric only. The Uniscope 100 offers upper and lower case alphabets.

Operating in conjunction with the Uniscope 300 or Uniscope 100 are two optional random access image file devices. The first is a random access microfiche display, the second, a low cost random-access slide projector. The image file devices may be operated singly or jointly, in conjunction with a particular CRT display. Both devices connect directly to the CRT, through which they are accessed and controlled by the remote processor. Additional media devices under development or evaluation are mentioned in ch. 3.

2.4 *Summary*

In this section Level B1 software was discussed from both systems and users areas view-points. Descriptions of the major features of Level B1 were presented along with the basic hardware requirements. The necessary hardware was discussed for both the central site and student terminals. A comparison of the characteristics of Level B1 with the the criteria in ch. 2 from the basis for extension to the system, which are discussed in the succeeding sections.

3. EXTENSIONS TO THE SYSTEM

3.1 *Need for extensions*

The need for extensions to the system arise as a result of the fact that the design criteria, ch. 1, have not been entirely met, as well as from recent developments in the field that have yielded insights into some possible new directions for such a system. It is felt that a major weakness lies in the difficulty a user may have in writing complex, conditional statements that are related to known student characteristics or student history A second area for concern relates to an inability to easily link student files and curriculum files into the author language in order to provide for flexible control of these or use of them by the author. Recent developments in Drill-and-Practice Systems and Computer-Managed-Instruction Systems have been reviewed and it has become apparent that the current system is not sufficiently flexible to handle these areas effectively in other than an experimental mode.

3.2 *Scope of extensions*

Extensions to the COPI-II, Level B1 System are taking place in two phases. In phase one. Level B1 is being extended to Level B2 with the addition of 1) seven new match types, 2) author defined counters, 3) the ability to display counter values during the interactive process, and 4) the ability to branch within a frame, from one logical unit to another. This last addition will make possible multiple and varied response acceptance and processing by one frame and will greatly simplify coding in certain commonly encountered circumstances.

In phase two of the extensions a new syntax is being defined as COPI-III. Emphasis on COPI-III will be in the area of improved string processing and generative techniques. Specifically, provision will be made for generating instuctional material from elements of the subject matter

according to defined procedures and relationships. COPI-III will also provide for some automatic selection and modification of material or alternate material based on experience by the system.

COPI-III will also incorporate a drill-and-practice subsystem as defined by Suppes [1969] and Jerman [1970], and a Computer Managed Instruction Subsystem for which characteristics are currently under study.

Finally, in the area of terminal systems, additional media devices are under study with emphasis on audio techniques and packaging of the entire student station as a modular system.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the continued assistance of Mr. R.E. Hegler, Director, Support Operations, Federal Systems Division, Univac, who has made possible the effort to develop the COPI System.

REFERENCES

BUCHOLTZ J.M., R.L. ECKERMAN, L.J. RODELL and H.P. LONG, 1969a, Detailed Language Specification for COPI-II, Level C (Univac, St. Paul), TR-0016-ESP-69.

BUCHOLTZ, J.M., 1969b, COPI-II, Level B Lesson Assembler Program Description. (Univac, St. Paul), TR-0037-EDS-69.

HAUPERT, J.H., 1968, General Record Analysis and Data Evaluation for COPI-II, Level B Computer Aided Instruction System. (Univac, St. Paul), TR-0010-ESP-68.

HAUPERT, J.H., J.M. BUCHOLTZ, N. RELLES, and M. CONLIN, 1970, Uniscope 300 and Teletypewriter Remote Terminal Operations Guide for COPI-II, Level B1 (Univac, St. Paul), in press.

LONG, H.P. 1968, Phonetics of the English Language and its application to the COPI Spelling Resolver. (Univac, St. Paul) TR-008-ESP-68.

JERMAN, M., 1970, Computer-assisted instruction in the Bulletin of the National Association of Secondary School Principals. 54,27-40.

MILBURN, Jr., R.G., J.M. BUCHOLTZ and N. RELLES 1970a, Interactive Dialect of COPI-II, Level B1. (Univac, St. Paul), TR-0042-EDS-70.

MILBURN, Jr., R.G., J.M. BUCHOLTZ, N. RELLES and M. CONLIN, 1970b, COPI-II, Level B1 Educational Data and Communicating System. (Univac, St. Paul), TR-0042-EDS-70.

RELLES, N., 1969a, A Preliminary Description of COPI-II, Level D. (Univac, St. Paul), TR-0018-SMSP-69.

RELLES, N., 1969b, An Interactive Calculation Mode for the use with COPI-II, Level B. (Univac, St. Paul) TR-0038-EDS-69.

SMITH, D.N., 1969, COPI-II, Level A Coursewriter's Guide. (Univac, St. Paul), TR-0025-69.

SUPPES, P., 1969, Computer-assisted instruction at Stanford in Educational Technology. January.

„CHIMP"

Kathryn Wolff and R.J. Munn

It is generally assumed in the educational process that learning will be much improved if every student is provided with his own individual tutor. In studying computer assisted instruction we look to the computer to be that tutor and in doing so we assume that the student is learning more and understanding more when he studies lessons presented by a computer which can analyze and respond at once to a student's input. The truth of such an assumption obviously depends on the **quality** of the lessons studied via the computer and their relevance to other studies which the student undertakes. It should be equally obvious that poor quality lesson material will remain poor no matter how it is presented.

While mass-produced curriculum materials may be suitable in certain educational situations, it is not likely that conversational computer programs will be successful, especially at the University level, unless the course material presented by the computer matches what we may call „the local conversational mode." A trivial example of this would be in notation; that is, subjecting a student to two different notations, one in the classroom and another at his terminal may mask the underlying principles being taught. We must hope then that the University faculty will be willing to prepare their own computer materials for interactive teaching. To lure the faculty in doing this two things must be done:

(i) The faculty must be convinced with realistic examples that computer assisted instruction does have a role in the teaching process whether this be in just the simple drill type interaction or in a more fundamental way, and

(ii) The preparation of lesson material must be made no more difficult than the careful preparation of a set of lecture notes.

One might hope that this second condition would prove extremely attractive to the faculty member truly interested in his teaching proficiency. The discipline necessary in sequencing lesson material for computer presentation to a range of students might assist the lecturer in developing the most logical conceptual hierarchy for presentation of a particular topic. One, in addition, might hope that effectiveness of teaching in certain ways could be studied by the lecturer, since the computer provides both a replicable teaching situation and, if desirable, a permanent record of each student-lesson interaction.

The situation at the University of Maryland is similar to that at many other universities and colleges. Typically, the campus has access to one or more digital computers with limited memory, many users, limited time sharing facilities and a computer science center not primarily concerned with computer assisted instruction. The faculty member who wishes to present some part of his course material via a computer rather than by class lecture or reading assignment is thus faced with an extensive programming task totally unrelated to the material that he is teaching – unless the author language assumes the burden for him.

A carefully designed CAI language should therefore have some and preferably all of the following attributes:

1. The language should be easy to learn and obvious in its operation.
2. The often used basic operations should be designed in a way corresponding to the natural requirements of the teacher, not of the computer.
3. Insertion, replacement and deletion of material should be as easy and as natural as any other form of text editing.
4. The language should allow the presentation of material in a natural and efficient way.
5. The language should provide for the analysis of both verbal and numerical responses.
6. The system should be coded in a common, high level language allowing for easy transfer from machine to machine.
7. The system should put as small a strain as possible on the computer system with reference to facilities usage.

The CHIMP language was developed for the easy preparation of CAI lessons in the physical sciences at the University of Maryland. The Attributes of the CHIMP language are outlined briefly in Table 1. A general scheme for the presentation of a simple one question/answer sequence is shown in figure 1. Its implementation in CHIMP is shown in figure 2. It is obvious that it is a sizable amount of coding if we are to handle the sequence in a realistic way. It is also obvious (by the fact that we haven't included explicit text and responses in figure 2) that the same sequence could be used in each such question/answer sequence. It thus is entirely natural to turn over to the computer the task of inserting the variable parts in the sequence wherever they occur. We can do this in CHIMP by the device known as a **macro**. The **macro** corresponding to

the sequences of figures 1 and 2 is shown in figure 3. It consists of primitive CHIMP coding with two additions; the substitution symbol Δ and an end of line symbol\. The **macro** has been given the **name** STRAT 1 which will be used to identify it subsequently. Now an author can use this **macro** by writing his lesson as shown in figure 4. Anything between consecutive backslashes is substituted for the corresponding substitution symbol in the **macro** definition, after the strategy STRAT 1 is invoked. (A number of editing features are available in order to format the text for CRT output but these are not essential to the discussion).

When the language was being developed it seemed reasonable that the typical faculty member would have access to both a secretary and a key puncher. It seemed quite unlikely that he would choose to sit at a terminal and type in a lesson segment of typically 1000 lines when he can have it punched on cards and run as a batch job. Hence, while it is possible to prepare a lesson at the terminal in CHIMP, no particular emphasis has been placed on interactive lesson construction. The language does have a number of interactive editing commands should the author decide to modify his lesson file directly, rather than modifying the lesson deck.

Many of the CAI languages currently available have extremely primitive computational facilities available to both the author and student. This is strange when one considers the computational power potentially available. Rather than utilize the general purpose software available for our computational sequences, it was decided to build a compiler as an integral part of the language. The language chosen was „BASIC" and the implementation allows both the student and author to use it at will. In addition, an indirect communication between student and author is established since they can effectively share the 260 simple BASIC variables and can thus transmit information through them.

The availability of such a large number of counters permits the lesson author to construct logic for making any desired branching sequences; if a large number of exercises of varying difficulty are available in a lesson file each student may have a selection of problems tailored to his demonstrated level of ability (cf. figure 4, incremental counter Z 1 which switches on attaining a value of 5 with the CHIMP command LIF). Access to „BASIC" by the author has increased the scope and redirected the emphasis of the lesson material. No longer is the author limited to examples that are within the analytic mathematical ability of the student.

As CHIMP has developed as a language over the past eighteen months it has become apparent that we have a powerful tool for preparing interactive lesson segments for computer use. One of the reasons for this we feel is that the language was developed in an interactive mode. No attempt was made to present the faculty with a system purportedly anticipating their every need but rather to develop a language that the faculty found useful.

It has become obvious in our limited experience that the **macro** approach to CAI is extremely useful and we are building a library of tested **macro** sequences. Two very important aspects that have emerged in **macro** use are:

(i) They can be used to rapidly prepare a large number of sequences of a similar type with very little effort – for example drill exercises in solving routine chemical calculations, see figures 5 and 6.

(ii) The construction used in the **macro** definition is an explicit definition of the teaching strategy. In the early stages of the education process much needs to be learned about the best way to present a set of simple concept: randomly, linearly, conditionally and with or without reinforcement? Each such „strategy" is easily represented as a **macro** construction and lessons with identical text and varying strategies are easily constructed by teachers to fit their local situation. The generality of the strategy makes available comparable data on student performance.

In the higher levels of education this second aspect of the strategy approach may become less inportant although its utility value in producing lessons easily will remain.

In summary CHIMP represents a language developed in an interactive environment, which has proved extremely popular with users. We realize that the system as it stands has limitations which we are endeavouring to remove. It seems to us that while the power to write interactive lessons is an extremely useful one, the real power of CAI will be in its ability to supply students with the power to manipulate and organize data or concepts and the like and to see immediately the consequence of their actions. In building the language we have allowed ourselves the ability to add to the language quickly and easily with little or no penalty in increasing the facilities usage. At the present time we are designing file interrogating processors to be used in much the same way as BASIC is now used for computations.

ACKNOWLEDGEMENTS

None of the work described in this paper would have been possible without the cooperation of a large number of people. In particular we would like to thank Dr. Joseph T. Vanderslice for allowing us to introduce CAI to the Chemistry Department at the University of Maryland, the Computer Science Center for the gift of computer time and finally the Graduate School which supported the investigation financially.

Table 1

Operation	Command	Example	
Text presentation	PRE	PRE	HI THERE
Accept response from student	ANS	ANS	P
Keyword match with a variety of options	KEY	KEY	1,Y,3,2, JOHN-SMITH, IS, SICK-ILL
Numeric response	LIM	LIM	1.559, 1.6E+00
Exact match	GUD	GUD	CMS
Unexpected response	UNX	UNX	5
Counter manipulation	CTR	CTR	A6 = 5 + A3 + 2 * B9
Unconditional branching	GTO	GTO	*-2
Conditional branching	GTC LIF	GTC LIF	Z9,5,R1,*+2,K1 A3+ 2.EQ.6,1,A3-2
Subroutines	SUB RET	SUB RET	A5 + 2
Editing responses	PAC	PAC	, . $ ' AEIOU
Enter a BASIC program	BAS	BAS	LET A1 = 1 + C3 PRINT INT (A1) END

The CHIMP system is written in FORTRAN IV. It occupies at a maximum 10.5K of core at any instant. The lesson material resides on a random access mass storage device. Only one copy of the lesson is needed for all its users, i.e. the lesson is re-entrant.

Figure 1

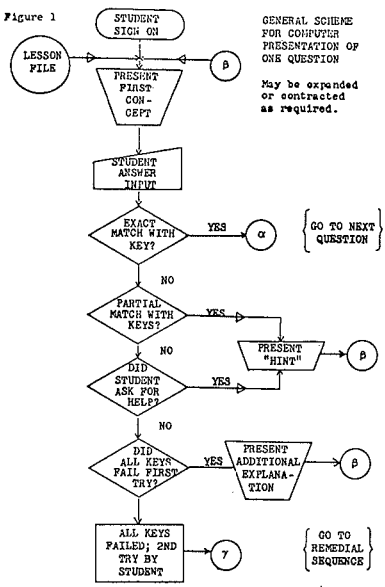


Figure 2

```
      GTO  *+2
A1    PRE  LET'S TRY THIS QUESTION AGAIN:
      PRE  [ TEXT HERE ]
      ANS  P
      KEY  [KEY ATTRIBUTES HERE]
           [KEY COMMENTS HERE]
        *
      LIM  [LIM ATTRIBUTES HERE]
           [LIM COMMENTS HERE]
        *
      KEY  [KEY FOR PARTIAL ANALYSIS]
           SORRY,YOUR ANSWER IS NOT QUITE COMPLETE.
           YOU SHOULD HAVE SAID [ ADDITIONAL TEXT]
      GTO* A1
      PAC  '
      KEY  N,3,1,HINT,HELP, DON'T KNOW
           [HINT TEXT ]
      GTO* A1
      UNX  1
           INCORRECT.  WHAT I WANTED YOU TO UNDERSTAND
           WAS [ ....]
      GTO* A1
      UNX  2
           NO, THAT'S STILL NOT QUITE WHAT I HAD IN
           MIND.  LET'S LOOK AT ANOTHER EXAMPLE.
      SUB  S7
        *
```

Figure 3

```
      MAC  STRAT1
      GTO  *+2
1     PRE  LET'S TRY THIS QUESTION AGAIN:
      PRE  Δ\
      ANS  P
      KEY  Δ\
           CORRECT.  Δ\
        *
      LIM  Δ\
           CORRECT.  Δ\
        *
      KEY  Δ\
           SORRY, YOUR ANSWER IS NOT QUITE COMPLETE.
           YOU SHOULD HAVE SAID  Δ
      GTO* Δ1
      PAC  '
      KEY  N,3,1,HINT,HELP, DONT KNOW
           Δ\
      GTO* Δ1
      UNX  1
           INCORRECT.  WHAT I WANTED YOU TO UNDERSTAND
           WAS Δ\
      GTO* Δ2
      UNX  2
           NO, THAT'S STILL NOT QUITE WHAT I HAD IN
           MIND.  LET'S LOOK AT ANOTHER EXAMPLE.
      SUB  SΔ
        *
      *
```

Figure 4

```
      STR  STRAT1
           \A\
QUESTION   \TEXT HERE, ANY NUMBER OF LINES\
KEY        \KEY ATTRIBUTES HERE, ONE LINE\
COMMENT    \TEXT HERE\
LIMITS     \NUMERICAL RANGE, FORTRAN E FORMAT ALLOWED\
COMMENT    \TEXT\
PARTIAL KEY  \ATTRIBUTES HERE\
COMMENT    \TEXT HERE\
           \A\
HELP TEXT  \ANY NUMBER OF LINES\
           \A\
WRONG ANSWER COMMENT \TEXT HERE\
           \A\\7\
```

Figure 5

```
      MAC  ACID
.     GTO  *+1
Δ2    PRE  LET'S TRY THIS QUESTION AGAIN:
      PRE  IN STANDARDIZING A Δ SOLUTION, IT WAS
           FOUND THAT Δ WAS REQUIRED FOR COMPLETE
           REACTION WITH Δ.  CALCULATE THE NORMALITY
           OF THIS SOLUTION.  CALL BASIC FOR YOUR
           CALCULATIONS.
      ANS
      LIM  .Δ\
           CORRECT.
      CTR  Z1=Z1+1
           *
      LIM  Δ\
           NO.  I THINK YOUR ERROR WAS
      GTO* Δ2
      UNX  1
           NO:  TRY THIS WAY OF DOING THE CALCULATION
           Δ\
      GTO* Δ2
      REM  CONTINUE MACRO AS REQUIRED.
      LIF  Z1.EQ.5,L3
      *
```

Figure 6

```
      STR  ACID
           \A\
QUESTION   \HYDROCHLORIC ACID\\25.8 ML OF ACID\
           \A SOLUTION CONTAINING 1.2588 GM OF NA2CO3 \
           \HYDROCHLORIC ACID\
LIMITS\          \
LIMITS OF ANTICIPATED ERROR\      \
(CONTINUE AS REQUIRED FOR MACRO, FIGURE 5.)
      STR  ACID
           \B\
QUESTION \SODIUM HYDROXIDE\\28.9 ML OF ALKALI\
           \A SOLUTION CONTAINING 1.0043 GM OF
            BENZOIC ACID (C6H5COOH, EQUIVALENT
            WEIGHT=122) \
           \SODIUM HYDROXIDE\
LIMITS FOR CORRECT ANSWER\      \
(CONTINUE AS REQUIRED FOR MACRO)
      STR  ACID
           \C\
QUESTION \SULFURIC ACID\\27.3 ML OF ACID\
           \0.1312 GM OF PURE NA2CO3 \
           \SULFURIC ACID\
(CONTINUE AS REQUIRED FOR MACRO.)
(CONTINUE FOR QUESTIONS D, E, F, G, H, ETC.)
L3   PRE  GOOD FOR YOU, $NAME3$ YOU HAVE WORKED FIVE
           NORMALITY PROBLEMS CORRECTLY.  NOW LET'S
           TRY SOMETHING ELSE.
     REM  LESSON CONTINUES TO SOLUBILITY PRODUCT
           CALCULATIONS.
```

SERA 69, A NEW HYPOTHETICAL MACHINE FOR EDUCATIONAL PURPOSES

W. L. van der Poel

1. INTRODUCTION

Although the basic principles of programming can nowadays be taught better in a higher level language, there is still a need for educating programmers who are able to work on the level of assembly languages. For those jobs where the utmost is required from a machine and especially for many parts of the 'hard' software as for example I/O packages, device controlling routines, interrupt routines, assemblers, compilers and tracers, one still must work on the level of the instructions available in the hardware repertoire of the machine.

Over the past 10 years, a lot of experience has been gained in Holland by using a hypothetical machine called SERA for courses in elementary and advanced programming. (1). The name was jocularly coined and stands for Seemingly Easy Reckoning Automaton (Schijnbaar Eenvoudige RekenAutomaat in Dutch). The main reason for creating such a hypothetical machine was that most real machines have lots of irritating idiosyncrasies and irregularities in the construction of their order codes which, from the standpoint of practical and efficient use, are perfectly acceptable, but, from the standpoint of teaching the fundamentals, real machines are often too complicated and their manuals too thick. It would be a waste of time for a pupil to work himself through such a thick handbook, while he has to change to a different real machine later anyway. A second reason is that on the neutral ground of universities, high schools and other teaching institutions it is very difficult not to make unsolicited propaganda for a particular manufacturer. A third reason is to standardize the machine language for a certain class of state examinations. A similar case is the City and Guilds code, which is in use in Great Britain for the same purpose.(2).

2. HISTORY

The original ideas for creating a hypothetical machine in the style of SERA were developed very gradually by many people among whom notably were: P. L. M. van Berkel, M. Euwe, G. C. Nielen, W. L. van der Poel and D. H. Wolbers. The old SERA (which we shall now call SERA 60 for convenience) was never specified by mutual agreement between lecturers to a great extent. In fact, in the beginning, every lecturer invented new instruction codes on the spot whenever he needed them. Gradually there came some uniformity in the mnemonics and the instruction repertoire. At the same time (1962) the first implementations of SERA 60 were made on Stantec ZEBRA, Telefunken TR4, Honeywell 125, IBM 1401 and Digital PDP8. As the host machines differed in word length and radix some implementations resulted in binary SERA's, others in decimal SERA's. These discrepancies resulted in a certain degree of incompatibility of programs. As long as only arithmetical operations were done with programs written in assembly language no harm was done, but programs explicitly manipulating with bit patterns and depending on word length (e.g., decimal and binary shift instructions) were incompatible. The abovementioned implementations all were pure interpretive systems, although for the Electrologica X1, a compiling version was written. In the latter case it is no longer possible to manipulate with instructions.

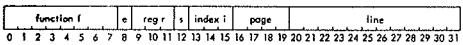
To summarize a few highlights common to all implementations: The instruction word was composed of two parts: an address part in the least significant part and a function code in the most significant part. One bit in the function part was used to indicate whether the address part was to be used as the operand itself or whether the content of the address had to be taken as operand. The unconditional jump had an function part = 0 so that e.g., a return address from a subroutine could be used as a jump to that address. One of the most useful instructions was the MOD, which modified the instruction immediately following by the content of its address. In this way and at the cost of one extra instruction every address in store could be effectively used as an index register. In the same way repeated MOD instructions could be used for multiple addressing. Still when one had to use the same modification over and over again, it could only be done at the cost of two instructions every time. There was no agreed system for interrupt handling. For an abbreviated description see Appendix 1, where the most important function codes are explained in a practically machine free, word length free and radix free ALGOL 68 description, which should be self-explanatory. For SERA 60 a word length free assembler was written which was described in its own assembler languages and which could assemble itself, once a handcoded copy was placed in the machine, irrespective even of radix and exact coding of the different mnemonics for the functions. (3), (4), (5), (6), (7).

3. SERA 69, INSTRUCTION FORMAT

In the last few years it was realized, that the features of SERA 60 were not adequate for representing the structure of present day machines. Especially the inability to deal with interrupt systems, more complicated I/O equipment such as disks, data transmission links, displays, memory protection, relocation, multiprogramming, etc., was heavily felt. Therefore a (small!) committee was set up to find common ground for defining a new SERA. This committee had the difficult task of designing a machine with the least possible number of exceptions, a simple instruction

repertoire and all the abovementioned features. Although it would be interesting to discuss how and why we arrived at the solutions chosen, I have to restrict myself in this lecture to the ultimate outcome alone.

The store of SERA 69 is byte addressed (1 byte = 8 bits). Most operations are performed on words of 4 bytes. Also the instruction format is always one word. Words always start in addresses which are exact 4-folds. The instruction format is as follows:



The function part f can designate 256 different operations of which only 45 are used. The remainder will all be trapped by the non-existent operation interrupt when working in user mode. By the so-called extra-code bit e the user can define another 256 different operations in his own region of the store by a particular kind of subroutine. All other part of the instruction word keep their normal significance and can be accessed in these subroutines. Because of this possibility of creating extra-code many sophisticated instructions have been omitted such as double-length floating-point operations, editing and special conversion instructions and stacking instructions. They can all be easily defined in extra-code.

The register designation r selects one out of 8 registers to act as the active accumulator. However, reg. 0 always contains 0, so in fact there are only 7 registers designated by A, B, C, D, E, F, and G. The empty register 0 is designated by L. These letters appear again as third letter in the function mnemonic as was the case in SERA 60 with A and B alone.

The same 8 registers play a dual role as index registers. The index register used is indicated by the i part of the instruction.

The right hand halfword is the address which can range over 2^{16} bytes directly and 2^{20} bytes indirectly in monitor mode. The calculation of the effective address is done in the following steps:

(1) The address of 16 bits is extended to full word length by adding copies of bit 16.
(2) The content of index register i is added in full word length.
(3) Now the s-bit comes into play. If s = 0 the address obtained in steps 1 and 2 is used as a direct operand itself. If s = 1 the content of the address is used as the operand.

Taking this content of an address is a different process in user mode and monitor mode. In monitor mode the right hand 20 bits are used to range over a full 2^{20} words directly but in user mode only the right hand 16 bits are used. Of these the left 4 bits are used to look up a real 20 bit address in a table of which the starting address is given in a register RELOK. To this 20 bit address the line part is added to give the actual address used. Thus each user can have up to 16 non-contiguous pages assigned to him by the monitor, which will look to him as one contiguous region. For each user the monitor keeps the relocation table. Changing from one user to another only requires refilling RELOK with another starting point. Only the monitor can do this by protected instructions. Furthermore, the relocation tables also contain two protection bits for each page. This gives 4 possibilities: (1) page completely protected (this can mean: unavailable in core. Swap with another page in background store), (2) writing and reading forbidden but instructions can be executed from that page, (3) reading permitted but writing protected, and (4) page is fully accessible.

When the s-bit = 0 this is denoted in the mnemonics by having the 4th letter a D (for direct operand), when s = 1 the 4th letter is (for Indirect). The correspondence between a written instruction in mnemonic code with symbolic address and modifier now looks as follows:

The use of the s-bit makes interregister transfer instructions superfluous; e.g., HPAD B (Take positive in A, B modified but Direct) will take the content of B into A Disregarding the relocation in user mode, the operand for most instructions can be succinctly expressed as:

OPERAND := **if** s = 0 **then** I[i]+ n' **else** M[I[i]] + n' **fi**

where I[i] denotes the content of index register i, n' denotes the elongated address and M[x] denotes the content of location x.

The available repertoire of instructions is summarized in appendix 2. A few instructions on this list merit further discussion. The old property that an unconditional jump is represented by an empty operation part has been kept from SERA 60, i.e. SALD is equivalent to the old SAL instruction. But a complete family of other jumps has been added. All jumps will effectively set a return address in the register, mentioned as the third letter of the mnemonic. If this is an L this has no effect as L is in fact a non-existent register. Furthermore, all jumps are conditional and test the condition register COND on three criteria: < 0, $= 0$, and > 0. Of these three possibilities only one can be fulfilled at the same time. On the other hand, three bits in the jump code are devoted to 8 combinations: all, <or =, <or >, <, = or >, =>, nothing, jumping resp. always, only when smaller or equal, etc.. The SN, jump never, still can be used to set a return address to the next address in some register.

The instructions for take, store, add, subtract are rather conventional. Multiply, divide, right shift and left shift will use two registers, namely the mentioned one and the previous one for the least and most significant part of the product, dividend, or number to be shifted resp.

The compare instruction fills the condition register without disturbing an accumulator and can also be used to set the condition to a switch in store by comparing with L; e.g., VGLI JOHN; SZLD THERE will jump to THERE only if JOHN was zero.

Some instructions have f $\geqslant$ 128. These instructions are protected in user mode. This means that they will cause an interrupt. Whether the monitor still wants to execute the required effect is up to the monitor, but is the normal case with TRAN, TEST and STOP. The transput will be dealt with later.

Some other instructions have an address which is not used as such or as source of an operand, but which points to a control sequence. This sequence usually consists of

more than one word of data. This is used on the TRAN, TEST and VERP instructions and will be discussed under TRAN.

4. THE INTERRUPT SYSTEM

The interrupt system is kept very simple and has no provision for automatic levels of priority. The monitor can, by software, treat some interrupts with higher priority than others. There are 32 channels on which an interrupt can come in. The interrupt channels correspond to the 32 bits in a word and are controlled by two masks. The enable mask EMK contains a 1 in each bit position corresponding to a channel for which an interrupt can be permitted. If the enable bit of a channel is 0, then that channel can never cause an interrupt and the devices on that channel could be regarded as not connected or non-existent. If the enable bit is on, then an interrupt is accepted if also the corresponding bit in the interrupt mask MK is 1. Otherwise this interrupt request is caught in the CATCH register until the bit in the interrupt mask becomes 1, or could be cancelled if the enable bit is made 0. As soon as an interrupt comes through both masks, the following actions take place simultaneously: the instruction counter IC and a second instruction counter PIC are swapped. The second instruction counter could have been prefilled with the address where to go on after the interrupt, and after the swapping it will contain the place where to go on with the interrupt program. The existing interrupt mask MK is saved in a second mask PMK and the interrupt mask MK itself is made zero so that no other interrupts can come in. The mode of operation is forced into monitor mode. (It is quite possible that the system was in monitor mode already. Also the monitor can make itself susceptible to interrupts if it has set some of the mask bits to 1). And lastly a register called the interrupt register INR will receive the information on the kind of interrupt being made. This could be the number of the device on a certain channel plus information on eventual errors.

Some interrupts are caused by the processor itself such as a protection violation in the relocation procedure or the execution of a non-existent instruction. This can only happen in user mode and these interrupts have priority over any of the masked interrupts. The interrupt register is filled with a negative characteristic for such an interrupt.

Interrupt for arithmetical overflow can be normally masked off by bit 31, as well in user as in monitor mode. They will not be discussed further.

The instruction VERP (= move characters) can take a long time. Special provision has been made to allow interrupts in the middle. The effect is properly described in appendix 3 although this effect could be achieved more efficiently in an implementation.

5. TRANSPUT

The input and output is governed by two instruction types: TRAN and TEST. Both instructions use the effective address to point to a sequence of control words. In the case of TRAN this control sequence contains:

(1) as the first word the beginning address in store (relocated to user area) from/to where transport of information takes place. All information is transported in the form of bytes, whether this represents characters or quarter word binary.
(2) The second word gives the number of bytes to be transported. For some functions this piece of data is irrelevant.
(3) The third word contains the number of the channel, the number of the device on that channel and the function (e.g., seek, read, write etc.) which has to be initiated.
(4) The fourth and possibly the fifth word if needed will indicate a device address such as arm positioning on a disk or long address on a word addressable drum.

When TRAN is given in monitor mode it will start transput on a real device. Once the TRAN has been given, the program goes on with the next instruction and the information transport requested will take place on a cycle stealing basis. When the transport is finished, an interrupt will be received from the channel if the corresponding enable bit in the enable mask and the bit in the interrupt mask are one.

TRAN can also be given in user mode. The form of the control sequence can be exactly the same. But as the instruction is trapped as a protected instruction, the monitor can now actually do the action with completely different real TRAN instructions. This gives the possibility to make a flexible assignment of logically interpreted device numbers and functions and the real resources allocated.

TEST will practically only be used in monitor mode to inquire into the progress or the correct finishing of a TRANsport. If the transport is finished and correct, the instruction following TEST will be done. Otherwise the control sequence will come into action. This is composed as follows:

(1) The first word contains an address to where a jump will be made if the transport is not yet complete or if the transport has some errors.
(2) The second word contains the information on the status of the transport. If negative, the transport is not yet complete. The amount indicates how many bytes still have to be transported. This enables the user to utilize the bytes which have been transported to the buffer, while the rest of the transport is still going on. If the word is positive, this is an indication that something has gone wrong. It depends on the kind of device as to what the significance is and what to do as a corrective action.
(3) The third word contains again channel and unit number as in TRAN for identification purposes.
(4) For slow devices this word contains the running buffer count. A fast device could very well have a special hardware register. Then a zero in this location will not be changed during the transport and does not allow for intermediate inspection.

6. USES

SERA 69 can be used in courses of high level as well of low level. If used in elementary courses, many advanced features can be omitted. In fact a mini-implementation has been made already on the PDP-8. In this implementation relocation, protection, paging and interrupt have been omitted.(9) The repertoire of functions has been reduced to the first 31 (not counting bit 0).

On the other hand SERA 69 can be used in courses dealing with the actual construction of multiprogramming monitors and operating systems; e.g., it is not necessary to

have a really big machine having the full complement of 2^{20} bytes. Even in a machine which can provide for only a limited number of pages, the pages not present in core can be flagged with protection bits 00 in the relocation table, so that on interrupt a page swapping can take place and the user still has a virtual store which is much bigger. This can all be programmed within SERA 69 itself and using no machine programming in the host machine.

Some instructions have been left out deliberately; e.g., there are no stacking instructions. These could be created in a number of ways.

(1) One could use an accumulator as stack pointer and write 2 instructions for a stacking operation. (Fast but clumsy).
(2) One could write a macro to generate these 2 instructions. (Fast and elegant).
(3) One could write extra-code instructions. (Elegant and easy but slow).
(4) One could trap some unused functions and return the required effect to the user by the monitor. (Very convenient but not so fast as 1).

In appendix 3, a description of SERA 69 in ALGOL 68 has been appended.

REFERENCES

(1) Van der Poel. W. L.(ed.), Definiërend rapport SERA 69, Stichting Nederlands Studiecentrum voor Informatica. Amsterdam (1969).
(2) Certificate for Computer Personnel (319), Advanced Certificate for Computer Personnel, City and Guild of London Inst., London.
(3) Wolbers, D. H., Principles of Programming in: Working Papers of the IFIP Seminar in ADP, London (1967). Vol. 2.
(4) Van der Poel, W. L., SERA, a Simulated Machine for Purposes of Explaning Compiler Construction. Information Processing Machines in: Proc. of the Symposium, Prague, Sept 7th - 9th, (1964) p. 117 - 131.
(5) Stillebroer, C. H. Het gebruik van SERA op de TR4 (The use of SERA on the Telefunken TR4) Vol. 1
De assembler, Vol. 2. Computing Centre, University of Technology, Delft.
(6) Frielink. A. B., Inleiding tot de Automatische Informatieverwerking. (Agon/Elsevier, Amsterdam 1968).
(7) Roets, P. F., Een rekursiewe makro samesteller voor SERA (Afrikaans, A recursive Macro Assembler for SERA). MSc. thesis, Dept. of Math., University of Technology, Delft (1966).
(8) Van Wijngaarden. A., (Ed.), B. J. Mailloux, J. E. L. Peck, C. H. A. Koster in: Report on the Algorithmic Language ALGOL 68. Num. Math. 14(1969) Feb.
(9) Coumou, J. H., SERA 69 op de PDP-8. Stichting Nederlands Studiecentrum voor Informatica. Amsterdam, May (1970).

APPENDIX 1. An ALGOL 68 description of SERA 60

In this description no consideration is given to some details such as exceeding capacity of the accumulators. Also I/O is very summarily treated.

```
begin int INSTRUCTION COUNTER, INSTRUCTION REGISTER, ACCUA, ACCUB, LINK, TOP,
BOTTOM; ¢ this are the actual registers ¢
int ADDRESS, OPERAND, FUNCTION, STAR ; ¢ some aux variables ¢
int FUNC1 = first bit beyond address, CAP = capacity of word, STARBIT = bit
giving place of direct/indirect operand indication; ¢ constants ¢
[0 : highest address]int M; ¢ this is the store ¢
proc SAL = (:(INSTRUCTION COUNTER := ADDRESS; goto INTERPRET));
bool CONDITION; proc NEG = (: OPERAND := - OPERAND);
START:      INTERPRET;
TEST:       if CONDITION then SAL fi;
INCREMENT:  INSTRUCTION COUNTER +:= 1;
INTERPRET:  INSTRUCTION REGISTER := M[INSTRUCTION COUNTER];
IM:         ADDRESS := INSTRUCTION REGISTER ÷ FUNC1;
            FUNCTION := INSTRUCTION REGISTER ÷ FUNC1;
            STAR := FUNCTION ÷ STARBIT;
            FUNCTION := FUNCTION - STAR × STARBIT;
            OPERAND := (STAR = 1|M[ADDRESS]|ADDRESS);
            case FUNCTION + 1 in SAL, SPA, SNA, SPB, SNB, SOA, SOB, SSP, MOD,
            HPA, HNA, HPB, HNB, BPA, BNA, BPB, BNB, OPA, AFA, OPE, AFB, VMC,
            DLN, HTA, HTB, BTA, BTB, PSH, POP, DOE, INA, UTA esac;
SPA: CONDITION := ACCUA ≥ 0; TEST;  ¢ jump (=Spring) on Pos accuA ¢
SNA: CONDITION := ACCUA < 0; TEST;  ¢ jump on Neg accuA ¢
SPB: CONDITION := ACCUB ≥ 0; TEST;  ¢ jump on Pos accuB ¢
SNB: CONDITION := ACCUB < 0; TEST;  ¢ jump on Neg accuB ¢
SOA: CONDITION := ACCUA = 0; TEST;  ¢ jump on zero accuA ¢
SOB: CONDITION := ACCUB = 0; TEST   ¢ jump on zero accuB ¢
SSP: LINK := INSTRUCTION COUNTER + 1; SAL; ¢ jump to Sub Program ¢
MOD: INSTRUCTION COUNTER +:= 1; INSTRUCTION REGISTER := M[INSTRUCTION COUNTER]+
     OPERAND; goto IM; ¢ modify next instruction ¢
HNA: NEG; HPA: ACCUA := OPERAND; INCREMENT; ¢ take (=Haal) Pos resp. Neg in A ¢
HNB: NEG; HPB: ACCUB := OPERAND; INCREMENT;
BPA: M[ADDRESS] :=  ACCUA; INCREMENT; ¢ Bring to store Pos from accuA ¢
BNA: M[ADDRESS] := -ACCUA; INCREMENT;
BPB: M[ADDRESS] :=  ACCUB; INCREMENT;
BNB: M[ADDRESS] := -ACCUB; INCREMENT;

APA: NEG; OPA: ACCUA +:= OPERAND; INCREMENT; ¢ add = OPtellen, subtract = ¢
AFB: NEG; OPB: ACCUB +:= OPERAND; INCREMENT; ¢ AFtrekken in Dutch ¢
VMC: (int PROD; ACCUA := (PROD := ACCUB × OPERAND) ÷ CAP; ¢ multiply = ¢
          ACCUB := PROD ÷ CAP); INCREMENT; ¢ VerMenigvuldigen ¢
DLN: (int DIVIDEND := ACCUA × CAP + ACCUB; ACCUA := DIVIDEND ÷ OPERAND;
     ACCUB := DIVIDEND ÷ OPERAND); ¢ divide = DeLeN ¢
HTA: ACCUA := M[TOP -:= 1]; INCREMENT; ¢ take from top of stack ¢
HTB: ACCUB := M[TOP -:= 1]; INCREMENT;
BTA: M[TOP] := ACCUA; TOP +:= 1; INCREMENT; ¢ bring to top ¢
BTB: M[TOP] := ACCUB; TOP +:= 1; INCREMENT;
PSH: M[TOP] := BOTTOM; M[TOP + 1] := LINK; BOTTOM := TOP +:= 2; INCREMENT;
POP: BOTTOM := M[TOP := BOTTOM - 2]; LINK := M[TOP + 1]; INCREMENT;
     ¢ PuSH and POP from stack ¢
DOE: INSTRUCTION REGISTER := OPERAND; IM;
INA; ACCUA := input character from input channel; INCREMENT;
UTA; output character to output channel := ACCUA; INCREMENT
end
```

APPENDIX 2. Instruction repertoire of SERA 69

The mnemonics are given in Dutch. They could be changed easily by redefinition or in the assembler. r stands for L, A, B, C, D, E, F, or G.

s=0	s=1	f	action	remarks
SArD	SArI	0	Jump always	⎫
SHrD	SHrI	1	Jump on at most	⎪
SVrD	SVrI	2	Jump on different	⎪
SKrD	SKrI	3	Jump on smaller	⎬ Return address in reg. r
SMrD	SMrI	4	Jump on at least	⎪
SZrD	SZrI	5	Jump on zero	⎪
SGrD	SGrI	6	Jump on greater	⎪
SNrD	SNrI	7	Jump never	⎭
HPrD	HPrI	8	Take positive	
HNrD	HNrI	9	Take negative	
	BPrI	10	Store positive	
OPrD	OPrI	11	Add	Set condition
AFrD	AFrI	12	Subtract	Set condition
VMrD	VMrI	13	Multiply	
DLrD	DLrI	14	Divide	
OCrD	OCrI	15	Add and carry	For multi-length operations
AOrD	AOrI	16	Subtract and carry	For multi-length operations
ENrD	ENrI	17	Logical AND	Set condition
OFrD	OFrI	18	Logical OR	Set condition
XOrD	XOrI	19	Exclusive OR	Set condition
RSrD	RSrI	20	Right shift double reg.	
LSrD	LSrI	21	Left shift double reg.	
VGrD	VGrI	22	Compare	Set condition only. No operation
EXrD	EXrI	23	Execute	Next address in r
HKrD	HKrI	24	Take character	
	BKrI	25	Store character	
KNrD		26	Convert characters to number	Address disregarded
NKrD		27	Convert number to characters	Address disregarded
STPD	STPI	156	Stop	Protected instruction
TRAN		157	Transput	Address points to control seq.
TEST		158	Test transput status	Address points to control seq.
		31	Spare place in first 32	
DOrD	DOrI	32	Floating add	
DArD	DArI	33	Floating subtract	
DVrD	DVrI	34	Floating multiply	
DDrD	DDrI	35	Floating divide	
	HALI	36	Take all registers and condition from 6 successive words	
	BALI	37	Store all registers and condition (condition in word 0)	
VERP		38	Move characters	Address points to control seq.

The following instructions are all protected and are used only by the monitor.

TRKD	TRKI	128	Return to user mode
HVMD	HVMI	129	Take enable mask
HMKD	HMKI	130	Take interrupt mask
HPKD	HPKI	131	Take second interrupt mask
HPOD	HPOI	132	Take second instruction counter for swapping
	BPOI	133	Store swapped instruction counter
	BIMI	134	Store interrupt register
HRKD	HRKI	135	Take relocation table pointer

APPENDIX 3. An ALGOL 68 description of SERA 69

In this description most quantities are declared of mode bits, i.e., a row of boolean, where the bits width, i.e., the number of bits in the row of boolean, is supposed to be 32. Some quantities which play a role in transput are very much implementation bound. They are not properly declared according to ALGOL 68, but the identifiers are chosen in such a way that they are self-explanatory. Of all possible devices only one is described which knows itself under the designations: this dev, and this chan. Advantage is taken of the possibility to describe I/O devices running in parallel with the central processor in ALGOL 68. Of all possible function codes given in Appendix 2, only the essential ~~ones~~ and the most interesting ones are given. The double-length involving operations such as multiply and shifting, the floating point operations, and the convert operations are not given here. For a full description of the "hardware" of SERA the reader is referred to [1].

As a rule the programmer-accessible registers are given identifiers in capitals in the ALGOL programme, hidden variables and constants are written in small letters.

```
begin
¢ Preliminary declaration of all registers, constants, other quantities and
  procedures ¢
[0 : 2 ↑ 20 - 1]int M ; ¢ The store is represented by a row of integers,
ranging from 0 to 255 and effectively used as a byte ¢
priority <- = 2, -> 2 ; ¢ priority of operators shift left and right is low ¢
op <- = (bits a, int b)bits: a↑b; ¢ shift left operator ¢
op -> = (bits a, int b)bits: a↑-b; ¢ shift right operator ¢
op + = (bits a, b)bits: if b = zero then a      ¢ a recursive definition of ¢
       else (a /\ not b \/ not a /\ b) + (a /\ b <- 1)fi; ¢ adding binary ¢
op - = (bits a, b)bits: a + not b + one; ¢ subtr = add compl ¢
sema mutex = /1, ¢ for mutual exclusion of two processes, influencing the
same quantity ¢ sema stop = /1, restart = /0 ;
[1:number of devices]sema activate = (/0, /0, ... , /0); ¢ as many as there are
      devices ¢
[1:29]sema busy = (/1, /1, ... /1); ¢ as many as there are channels for dev ¢
[1:29]int channel information;
[1:number of devices]bits core address, device address, device reloc;
[1:number of devices]int action, number, running count;
[1:number of devices]bool error;
¢ Registers of the machine ¢
bits L, A, B, C, D, E, F, G, ¢ accumulators or index registers ¢
      COMD, IR, ¢ the instruction register, holding the present instruction ¢
      IC, ¢ the instruction counter ¢ PIC, ¢ second instruction counter ¢
      MK, ¢ interrupt mask ¢ PMK, ¢ second interrupt mask ¢ EMK, ¢ enable
      mask ¢ RELOC, ¢ the relocation table pointer ¢ CATCH, ¢ catches the
      interrupts ¢
int INR, ¢ the interrupt register, where source of interrupt can be seen ¢
bool MOVEBUSY := false , int MODE := 0 , ¢ start in monitor mode ¢
¢ Auxiliary variables, hidden registers ¢
bits operand, addr, numb, devad, y, time,
int chan, dev, act, function, i,
bool writelock, readlock, second.ex, ¢ to prevent a second EXEC in user mode ¢
¢ Constants ¢
bits zero = bin 0, four = bin 4, eight = bin 8, twelve = bin 12, sixteen = bin 16,
      one = bin 1, bit0 = bin(2↑31), byte0 = not bin 2↑24,
      byte1 = bin(2↑24 - 2↑16), byte2 = bin(2↑16 - 2↑8),
      byte3 = bin 255, long address mask = bin(2↑20 - 4),
      address mask = bin(2↑16 - 1), index mask = bin(2↑19 - 2↑16),
      reg mask = bin(2↑23 - 2↑20), page mask = bin(2↑16 - 2↑12),
      line mask = bin(2↑12 - 1), prot mask = bin(2↑31 + 2↑30),
      real page mask = bin(2↑20 - 2↑12);
¢ Procedures ¢
proc reg = int: abs(IR /\ reg mask -> 20), ¢ gives nr of accumulator ¢
proc I = (int x)ref bits: case x+1 in L, A, B, C, D, E, F, G esac;
proc effaddr = bits: I(abs (IR /\ index mask -> 16)) + (IR /\ address mask)
      + (17 [] IR | not address mask | zero ),
proc fetch = (bits x)bits: if readlock then INR := -2; INTERRUPT else
      i := abs(x /\ long address mask);
      (bin M[i] <- 24) + (bin M[i+1] <- 16) + (bin M[i+2] <- 8) + bin M[i+3] fi,
proc store = (bits x, y):( ↓ mutex; if writelock then INR := -3; INTERRUPT else
      i := abs (x /\ long address mask);
      M[i]   := abs (y /\ byte0 -> 24);
      M[i+1] := abs (y /\ byte1 -> 16);
      M[i+2] := abs (y /\ byte2 -> 8);
      M[i+3] := abs (y /\ byte3); ↑ mutex fi);
proc relocate = (bits x)bits: if MODE = 0 then  x
      else bits page = x /\ page mask -> 10 ;
           bits line = x /\ line mask ,
                relocword = fetch(RELOC + page),
           int protection = abs-(reloc word /\ prot mask),
           bits realpage = reloc word /\ real page mask;
           case protection+1 in (INR := -1; INTERRUPT) ,
                                (writelock := readlock := true) ,
                                 writelock := true esac;
           real page + line  fi ;

¢ Now follows the parallel start of the central interpreter and all
  peripheral devices ¢

par begin
IC := zero; ¢ start in location zero ¢
MK := PMK := zero; ¢ initially no interrupts allowed, no device connected ¢
¢ Central interpreter ¢
NEXT: ↓ mutex; if (y := CATCH /\ MK) ¢ zero
           then chan := 0; for x to 29 while x [] y = false do chan := x+1;
                CATCH := CATCH /\ not(bin 2↑(31-chan));
                INR := channel information[chan]; ↑ busy[chan]; ↑ mutex
           else ↑ mutex; INTERPRET fi;
INTERRUPT: y := IC \/ (MOVEBUSY|bit0|zero); IC := PIC \/ longaddr mask;
      MOVEBUSY := 1 [] PIC; PIC := y; PMK := MK; MK := zero; MODE := 0;
INTERPRET: second.ex := readlock := false; y := relocate(IC) /\ longaddr mask;
      readlock := false; IR := fetch(y); IC := IC + four;
EXECUTE: writelock := readlock := false; L := zero;
      operand := (13 [] IR|fetch(relocate(effaddr))|effaddr);
      function := abs(IR /\ byte0 -> 24);
```

```
EXTRACODE: if 9 [] IR then y := fetch(relocate(bin function <- 2));
      store(relocate(y), IC);
      store(relocate(y + four), operand);
      store(relocate(y + eight), effaddr);
      store(relocate(y + twelve), IR /\ regmask);
      IC := y + sixteen; INTERPRET fi;
      if MODE = 1 /\ function > 127 then INR := -4; INTERRUPT fi;
      case function + 1 in JUMP, JUMP, JUMP, JUMP, JUMP, JUMP, JUMP, JUMP,
            LOADPOS, LOADNEG, STOREPOS, ADD, SUBTRACT, MULT, DIVIDE, DOUBLE ADD,
            DOUBLE SUB, LOGAND, LOGOR, EXCL OR, RIGHT, LEFT, COMPARE, EXEC,
            LOADCHAR, STORECHAR, CONVCHARNUM, CONVNUMCHAR, NONEXIST, NONEXIST,
            NONEXIST, NONEXIST, FLADD, FLSUB, FLMUL, FLDIV, LOADALL, STOREALL,
            MOVECHAR esac;
      case function - 127 in RETURN TO USER, LOAD EMK, LOAD MK, LOAD PMK,
            LOAD PIC, STORE PIC, STORE INR, LOAD RELOC esac;
      case function - 155 in STOP, TRAN, TEST esac;
NONEXIST:  if 31 [] MK then INR:= -5; INTERRUPT else NEXT fi;

{ The different functions }
JUMP: I(reg) := IC;                                   { store return address }
      if 6 [] IR /\ abs COND < 0  \/                  { jump on condition }
         7 [] IR /\ abs COND = 0  \/
         8 [] IR /\ abs COND > 0   then IC := operand fi; NEXT;
LOADPOS: I(reg) := operand; NEXT;
LOADNEG: I(reg) := zero - operand; NEXT;
STOREPOS: if 13 [] IR then store(relocate(effaddr), I(reg))
                      else (31 [] MK | INR := -5; INTERRUPT) fi; NEXT;
ADD:       COND := I(reg) := I(reg) + operand; NEXT;
SUBTRACT:  COND := I(reg) := I(reg) - operand; NEXT;
LOGAND:    COND := I(reg) := I(reg) /\ operand; NEXT;
LOGOR:     COND := I(reg) := I(reg) \/ operand; NEXT;
EXCL OR:   COND := I(reg) := I(reg) /\ not operand \/ not I(reg) /\ operand;
           NEXT;
COMPARE:   COND := I(reg) - operand; NEXT;
EXEC: (second ex /\ 31 [] MK | INR := -6; INTERRUPT); { no chain of executes }
      IR := operand; I(reg) := effaddr + four; second ex := true; EXECUTE;
LOADCHAR: (readlock[INR := -2; INTERRUPT);
      I(reg) := I(reg) /\ not byte3 + (13 [] IR[bin M[abs(relocate(effaddr))]] |
                effaddr /\ byte3); NEXT;
STORECHAR: (writelock[INR := -3; INTERRUPT);
      M[abs relocate(effaddr)] := abs( I(reg) /\ byte3 ); NEXT;
LOADALL:   COND := fetch(relocate(effaddr)); for j to 8 do
           I(j)   := fetch(relocate(effaddr + bin 4 × j)); NEXT;
STOREALL:  store(relocate(effaddr), COND); for j to 8 do
           store(relocate(effaddr + bin 4 × j), I(j); NEXT;
{ protected instructions }
RETURN TO USER: MK := PMK; MOVEBUSY := 1 [] operand;
      IC := operand /\ longaddrmask; MODE := 1; NEXT;
LOAD EMK:  ↓mutex; EMK := operand; CATCH := CATCH /\ operand; ↑mutex; NEXT;
LOAD MK:   MK := operand; NEXT;
LOAD PMK:  PMK := operand; NEXT;
STORE PIC: store(effaddr, PIC); NEXT;
STORE INR: store(effaddr, bin INR); NEXT;
LOAD RELOC: RELOC := operand; NEXT;
STOP:      ↑stop; ↓ restart; IC := operand; NEXT;

{ Transput }
TRAN: addr := fetch(relocate(effaddr));
      numb := fetch(relocate(effaddr + four));
      chan := abs ((y := fetch(relocate(effaddr + eight)))/\ byte3);
      dev  := abs (y /\ byte2 → 8);          { extract control sequence }
      act  := abs (y /\ byte1 → 16);
      devad:= fetch(relocate(effaddr + twelve));
      if 2↑(31-chan) [] EMK then
      core address[dev] := addr;
      deviceaddr[dev] := devad;              { transfer control data to }
      action[dev] := act;                    { device }
      number[dev] := abs numb;
      devicereloc[dev] := RELOC;
      ↑activate[dev] fi; NEXT;
{ Each device has to store the present value of RELOC to be able to do the
proper relocation to the pages of this user, when control has been given to
another user (with another relocation table) in the meantime }

TEST: chan := abs((y := fetch(relocate(effaddr + eight)) /\ byte3);
      dev := abs (y /\ byte2 → 8); ↓ mutex;
      if ready[dev] ↓ 0 then IC := fetch(relocate(effaddr));
            store(relocate(effaddr + four), bin ready[dev]);
            store(relocate(effaddr + twelve), running count[dev]) fi;
      ↑mutex; NEXT
end , { Now follows an example of one of the devices. All devices run in
      parallel and follow the same pattern }
begin do ( ↓ activate[this device]; ↓ mutex; ready[this device] := -1; ↑ mutex;
      ↓ busy[this channel];
      if action[this device] = read then for i to number[this device]
            while not error[this device] do
            (running count[this device] := i;
            M[special relocate(core address[this device] + i)] :=
            content of(device address[this device] + i) ); { end of do }
            (error | ready[this device] := positive error indication)
      elof action[this device] = write then .... fi;
      ↓ mutex; (2 ↑(31-this channel) [] EMK | CATCH := CATCH \/
      bin 2 ↑(31-this channel) ); channel information[this channel] :=
      this channel + this device + positive error indication;
      ready[this device] := (ready[this device] < 0 | 0); ↑ mutex )end ,

{ The clock is exclusively connected to channel 1 and is also device 1.
It is composed of two parts: the device handler and the clock proper.
The clock proper only ticks every unit of time and registers the number of
ticks. The device handler gives the time when TRAN is reading in the core
address. It sets a timer to number and allows an interrupt after that number
of ticks when TRAN is used with the write function }

begin do( ↓ activate[1]; if action[1] = read then
            store(special relocate(core address[1]), time)
      elof action[1] = write then
      ↓ busy[1]; for i to number[1] do delay one unit of time ;
      ↓ mutex; CATCH := CATCH \/ bin 2↑30; ↑ mutex fi ) end ,
{ core address[1] still contains the address of a subroutine to where the
user wanted to go. It is up to the monitor, whether control is directed there
at real time or after a simulated time }

{ the clock proper }
begin do ( delay one unit of time; time := time + one) end ,

{ The parallel process, simulating the start button }
do ( ↓ stop; L:  if start button pressed then ↑ restart else L fi ),

{ All other devices }
end { of parallel processes }
end { of description }

{ The procedure special relocate will not be specified here. It is almost
exactly as relocate except that special relocate [this device] will be used
instead of RELOC. Interrupt channel 30 and 31 are not trapped in CATCH and
have no semaphore busy. The channel 31 is used for interrupts on arithmetic
overflow. They are not described here }
```

A SUB-SET OF ALGOL AS AN INTRODUCTORY PROGRAMMING LANGUAGE

Geoffrey Riding

1. INTRODUCTION

During recent years the teaching of programming, particularly in auto-code or high-level language, has increased to such an extent that it now includes most universities and technical colleges, and a large number of schools at the secondary level (e.g. BIRKIN, 1969). In the latter two categories particularly, the students are frequently given only an introductory course, without necessarily making formal use of a computer until they move on to higher education or into industry or profession. When they make this move it is quite possible that they will find themselves using a different language than the one they learned at school or college, and this may pose problems for them if their introductory language was carried to too great a depth.

It is therefore desirable that a student should be given a very thorough grounding in a language of about the level of Mercury Autocode, Elliott Autocode, etc. One approach to this problem is to design special-purpose languages purely for teaching beginners, such as BASIC (NIXON, 1969); another approach is to teach actual languages such as Mercury Autocode, etc. It seems a pity, however, to increase yet again the number of programming languages in existence, but the obvious disadvantage with the early type autocodes is that these are rapidly becoming obsolete.

For these reasons the sub-set of ALGOL described in this paper has been defined, and this will henceforth be referred to as the „AGL sub-set of ALGOL", or more simply, as "AGL". This can be described very briefly as „Mercury Autocode written in ALGOL notation", and since it is a strict sub-set of ALGOL it introduces students to programming by means of a simple autocode-type language, but which nevertheless is part of a „living language". This means that subsequently they can move „upwards" to full ALGOL, or „sideways and upwards" in some other language, without the necessity to „unlearn" anything.

A second consideration, in this type of teaching, is the desirability of providing very extensive fault monitoring facilities, since this can reduce considerably the load on the teacher in helping students to get their programs working. Whilst programs written in AGL may of course be run with any ALGOL compiler (subject to the usual input/output considerations), the fault monitoring provided may not be considered sufficient for introductory purposes. Since AGL is fundamentally a simple language, however, it is relatively easy to write a compiler specifically for this purpose, and to include in it rather more monitoring facilities than exist in most ALGOL compilers. Since student programs tend to be numerous but short, some installations may also consider it worthwile to provide for batch processing in such an AGL compiler.

The remainder of this paper will be divided into two parts: (a) the definition of AGL, with additional descriptive notes – this is independent of any particular computer, and (b) a description of the implementation on the Atlas computer – in which extensive monitoring and batch-processing have been included.

2. DEFINITION OF AGL

A formal definition of AGL is given in Table 1, in a notation which is somewhat more simple than the usual meta languages, and which consequently may be given to students for reference purposes, once they have assimilated the main features of the language.

The notation of Table 1 assumes the following conventions:

(a) upper case WORDS are used for defining the language – single upper case letters may, however, be used as alternatives for the corresponding lower case letters.
(b) symbols which have particular meanings:

:-	„defined as"
|	„or"
&	„followed by" (omitted where no ambiguity can arise)
?	„optional"
'	integer type (prime follwing a definition word)
---	„pattern repeated as often as desired"

(c) all other characters are part of the language as shown
(d) NIL has the meaning „nothing"

The following abbreviations will also be used throughout the remainder of the paper:

CHAR	= CHARACTER	INT	= INTEGER
CONST	= CONSTANT	OPND	= OPERAND
OPTR	= OPERATOR	EXPR	= EXPRESSION
COMPAR	= COMPARATOR	STAT	= STATEMENT
COMPND	= COMPOUND	DECLAR	= DECLARATION

Additional Details of AGL

2.1 *Arrays, Expressions, and Assignments*

Arrays are assumed to be of type real, this being the only type allowed.

Expressions are evaluated from *left to right*, except in so far that multiplication and division are carried out before addition and subtraction.

If the variable to which an assignment is being made is of type integer, then the right hand side must be one of the following:

(a) an expression in which all operands are integer variables or integer constants, and which does not contain the division operator,
(b) the „entier" function.

(c) input of integer data (see below).

2.2 *Input and Output*

Although the ALGOL 60 report (NAUR, et al, 1963) did not specify any standard form of input/output, it was considered desirable that a teaching language such as AGL should include a defined form of input/output. As given here, the relevant procedure calls are taken from Atlas ALGOL (I.C.T., 1964), and operate under following conventions:

(a) VARIABLE .= read – this procedure call reads the next number from the data, and assigns it to the given VARIABLE. The numbers forming the data will be in fixed-point form, with an optional sign, and will be terminated by a space or new line – this character will be read as part of the number.

(b) print (OPND, INT, INT) – prints the value of the OPND in fixed-point from if the *first* INT parameter is non-zero – in which case this INT will specify the number of character positions to be output before the decimal point. If however, the first INT is zero, then the instruction will print in standardised floating-point form. In either case the *second* INT indicates the number of decimal places to be printed. The number will be followed by 2 spaces.

(c) NAME' := readch – reads the next single character from the data and assigns its „numerical value" to the given **integer** location. The „numerical value" of each character will be according to some pre-determined list which will be included in the compiler.

(d) printch (OPND') – prints the character whose „numerical value" is equal to OPND'.

(e) write text ({ STRING }) outputs all the characters of STRING. However, if spaces are required in the output, these will have to be indicated in the STRING as ß. The symbols { } are used as „string quotes" in Atlas ALGOL.

(f) space (OPND'), newline (OPND') – prints the specified number of spaces or newlines.

2.3 *Control Statements*

The format defined for labels was chosen to correspond with that for numerical labels in languages such as FORTRAN, whilst retaining the ALGOL principle of identifiers which commence with a letter. The preferred format uses upper case L, since lower case may be confused with the digit 1. To simplify the writing of a compiler, it might be desirable to impose some limit on the value of the INT used in the label (on the ATLAS implementation this must be in the range 1 – 127).

With respect to compound statements, labels are considered to be „global", and all labels in the program must therefore be distinct. Jump instructions within a compound statement may therefore refer to labels either inside or outside that compound statement. Jumping *into* such a statement, however, is not allowed.

In a **for** statement of the form

for i := p **step** q **until** r **do** .....

the values of p, q, and r must be such that

(r-p)/q + 1

is a positive integer.

For statements may, in principle, be „nested" to any depth, but in practice a limit may be imposed by the compiler (on the Atlas implementation this limit is 10).

2.4 *Miscellaneous Statements, etc.*

The delimiter **comment** and everything following it, up to the next semi-colon, will be ignored. Comments must always be terminated by a semi-colon – even when followed by **end**.

The existence of a „dummy" statement implies the following:

(a) **end** may be labelled, since it is a dummy statement in fact which bears the label.

(b) the statement before a *labelled* **end** must be terminated by a semi-colon.

(c) the last statement before an unlabelled **end** may be terminated by a semi-colon if desired.

(d) the **begin** of a compound statement may be terminated by a semi-colon, but *not* the first **begin** of a program since the concept of a dummy does not apply to declarations.

2.5 *General Example*

The following program for sorting 10 numbers into ascending order is written in normal Atlas ALGOL, but within the limitations of the AGL sub-set. It will be seen that the restrictions of AGL are relatively unimportant, whilst the structure of this program in such that it could be easily re-written in an autocode or other high-level language, with few changes other than notation.

```
begin

    real a integer i, j, k, n; array z[1:10];
    for i := 1 step 1 until 10 do z[i] := read;

    for i := 1 step 1 until 9 do
        begin k := i; n := i ½ 1;
        for j := n step 1 until 10 do
        begin if z[j] < z[k] then goto L1;
        k := j;
        L1: end;
        a := z[k]; z[k] := z[i]; z[i] := a end;

    for i := 1 step 1 until 10 do print (z[i], 2, 0)
end;
```

3. IMPLEMENTATION OF AGL ON ATLAS

3.1 *General*

Although an ALGOL compiler is available on Atlas, a separate AGL compiler has been written in order to provide much more extensive monitoring facilities, and to allow for batch processing. This second factor was quite an important one, since it has enabled any number of AGL programs to be run as one Atlas „Job" (HOWARTH, PAYNE and SUMNER, 1961), with a considerable saving of the overheads in cost and time. On Atlas, these overheads include „inter-job gap", time to find and read the compiler from the Supervisor (KILBURN, HOWARTH, PAYNE and SUMNER, 1961), etc., and form a considerable proportion of the total time for student programs using the normal ALGOL compiler.

Whilst input is normally assumed to be on Atlas-coded 7 track paper tape (using a Friden flexowriter), versions of the compiler have been provided which allow for input on Mercury-coded and Elliott-coded 5-track tape. The appropriate version can be called for in the „Job Description" (HOWARTH et al, 1961), and these are known as AGL_5 and AGLE respectively (the normal version being AGL_7).

Special punching conventions, however, must be adopted in these cases, and these are given in Table 2.

3.2 *Method of Implementation*

The AGL Compiler has been written in Atlas Autocode (BROOKER, ROHL and CLARK, 1966), making use of the „phrase structure" facilities (BROOKER, MORRIS and ROHL, 1967) for the initial reading and syntax checking of the AGL programs. (Table 1 is essentially a copy of the phrases and formats used in the Compiler). The analysis parts of the Compiler, however, were written mainly in Atlas machine code, within a framework of Atlas Autocode sub-routines.

Atlas Autocode also allows programs (in this case the AGL compiler) to be defined as a „special compiler" (MORRIS and ROHL, 1967) which may be „called" by means of a Job Description only – this being followed by the data. This facility is particularly useful when the AA program is itself a compiler, and for which the „data" is a program in some particular language – in this case AGL.

The decision to make the AGL Compiler capable of batch-processing indirectly led to a simplification of the method of handling the input/output and fault monitoring in AGL. Since the Compiler is kept in store during the execution phase of each program, it is not necessary to make the compiled program completely self-contained, in the sense that control can be passed back to the Compiler for certain specific tasks such as reading and printing. In fact the Compiler itself makes use of the corresponding sub-routines of Atlas Autocode by calling these in the normal way, control then being passed back to the compiled AGL program by suitable links. The same technique is used for the execution-time fault monitoring also, though in this case control does not return to the particular program which is faulty.

3.3 *Program Headings and Batching*

Each batch of programs, forming one Atlas Job, must be preceded by a Job Description, and finish with the terminator ***Z. The Job Description must finish with a statement requesting the particular version of AGL required; control at this point is passed from the Supervisor to the corresponding Compiler.

Within any one batch of programs, each program must be preceded by a „title" of the form:

title any ∮ string ∮ of ∮ characters;

This title will be printed as the first line of the output, and thus serves to identify each program with its results. The symbol ∮ is used to indicate a space, as in the „write text" instruction.

Each program is allowed a certain amount of computing time and output, but if either limit is reached the program is terminated, with suitable monitoring. The amounts allowed are normally 1000 instruction interrupts, i.e. about 6 secs of computing, and 100 lines of output. The programmer, however, may assign his own limits by either (or both) of the following instructions:

computing INT;

output INT;

These must appear between the **title** and the **begin** of the program.

Finally, each program and its data, except the last, must be terminated by +++T, the last one being terminated by +++Z (followed by ***Z, as indicated above). These terminators enable the Compiler to detect the complete end of each program and its data, and serve to re-set the Compiler when the amount of data called for by any one program differs from the amount given.

3.4 *Monitoring and Fault Diagnostics*

As mentioned in the Introduction, one of the main objects of this implementation was the provision of monitoring facilities which give the programmer as much information as possible. For this reason the Compiler attempts to monitor the structure of the program, and any faults, by means of a „line number". Unfortunately this concept does not exist in any formal sense, and consequently the numbers given out by the Compiler must be considered only as a rough guide. Nevertheless this is normally sufficient, since the given line number normally applies to the line containing the *next* semi-colon.

During the „compiling phase", the **begin** and **end** of the program are monitored, together with the corresponding line numbers. The start and finish of all **for** statements are similarly monitored. Instructions which are syntactically faulty are also monitored, the line number in this case being followed by *. An attempt is made by the Compiler to indicate the type of error in each case, the complete list of monitored compile-time faults being as follows:

ILLEGAL USE OF HEADING
ILLEGAL USE OF „BEGIN"
ILLEGAL USE OF DECLARATION
ILLEGAL USE OF LABEL
ILLEGAL USE OF BASIC STATEMENT

The above 5 faults imply the use of an instruction which is syntactically correct, but which is used in an illegal position in relation to the other instructions.

NAME? HAS NOT BEEN DECLARED
NAME? HAS ALREADY BEEN DECLARED
NAME? HAS BEEN DECLARED AS AN ARRAY
NAME? HAS BEEN DECLARED AS A SCALAR
NAME? HAS NOT BEEN DECLARED AS „INTEGER"

The last 3 faults indicate the use of a name in a context which is not allowed in relation to the declared type of that name (e.g. the use of a **real** name as a „subscript").

RHS IS NOT OF TYPE „INTEGER"

An attempt has been made to assign a general „expression" to an **integer** location.

ILLEGAL USE OF „REAL" QUANTITY

A „real" operand has been used where only „integer" operands are allowed, e.g. in the „space" instruction.

NON-INTEGER IN „FOR" STATEMENT
„FOR" NESTING TOO DEEP
SUBSCRIPT OUT OF BOUNDS
LABEL? ALREADY SET
LABEL? OUT OF RANGE
INSTRUCTION NOT RECOGNISED

This is followed, on the next line, by a copy of the unrecognised instruction.

STATEMENT TOO LONG

Instruction more than 256 characters in length.

INPUT ENDED DURING COMPILING

+++T, or +++Z has been encountered whilst still trying to compile the program.

OUTPUT LIMIT REACHED DURING COMPILING

If any faults are detected during compiling, then the program is terminated entirely by

COMPILED PROGRAM FAULTY

If, however, there are no detectable faults during the compiling phase, then this will conclude with

PROGRAM COMPILED AND ENTERED

and the program will proceed to the „execution phase".

During execution, checks for various faults are applied, e.g. attempts to find the square root of a negative quantity, references to array elements with subscripts outside the declared bounds, etc.. The complete list of execution-time faults is as follows:

DIVISION BY ZERO

EXPONENT OVERFLOW

An attempt to calculate a number greater than 10^{114} (sometimes implies „division by zero")

SQUARE ROOT OF A NEGATIVE

LOG OF A NEGATIVE

EXCESS STORE

This can occur if very large arrays are declared, in which case the „store" requested in the Job Description must be increased. However it is more likely that some other fault in the program has caused this.

NON-INTEGRAL „FOR" LOOP

The condition given in § 2.3 for the parameters of a **for** statement has not been satisfied.

ARRAY SUBSCRIPT OUT OF BOUNDS

May also refer to switch jump.

LABEL NOT SET

SPURIOUS CHARACTER IN „READ"

Data does not comform to description given in § 2.2(a)

ATTEMPT TO READ „REAL" DATA INTO „INTEGER"

COMPUTING LIMIT REACHED

OUTPUT LIMIT REACHED

INPUT ENDED DURING EXECUTION

+++T or +++Z has been encountered during the execution of an „input" statement.

COMPLETE INPUT ENDED

The last program/data was not terminated with +++Z.

When an execution-time occurs, the program is terminated (with appropriate monitoring and line number), and an indication of the current state of the „working area" of the store is given. This is done by printing all scalar names, together with their current contents, and similarly for the first *ten* elements of each array. Locations containing zero, however, are omitted entirely from this list.

Finally, the output for each program concludes with the total computing and output amounts used (the corresponding figures are also given for the compiling phase only). This „logging" may then be used by the programmer as a guide for modifying his requested computing and output limits, as described in § 3.3 above.

3.5 *Example of Output*

As an example, the program given in § 2.5 was run with the AGL Compiler, preceded by

title sort ♯ numbers ♯ example;

and followed by data:

27 5 31 56 8 17 56 24 92 10

The resulting output is reproduced below:

```
                    SORT NUMBERS EXAMPLE


AGL SUB-SET OF ALGOL

    3   START OF PROGRAM
    4     START OF 'FOR' STATEMENT
    4     END OF 'FOR' STATEMENT
    6     START OF 'FOR' STATEMENT
    8         START OF 'FOR' STATEMENT
   10         END OF 'FOR' STATEMENT
   11     END OF 'FOR' STATEMENT
   13     START OF 'FOR' STATEMENT
   13     END OF 'FOR' STATEMENT
   13   END OF PROGRAM
PROGRAM COMPILED AND ENTERED



  5   8   10   17   24   27   31   56   56   92


                          COMPILING      TOTAL
INSTRUCTION COUNT             96           101
OUTPUT COUNT                  28            41
```

4. CONCLUSIONS AND ADDITIONAL COMMENTS

The AGL sub-set of ALGOL has been taught on a number of courses at Stretford (Lancs) Technical College, including courses for teachers from local schools. About five hours of lectures and five hours of practical sessions, taken over a five-week period, seem to be quite adequate to gain a reasonable understanding of programming to the AGL level. As mentioned in the Introduction, the extensive monitoring provided by the Atlas implementation undoubtedly helped the students to get their test programs working quickly, and with general supervision only from the teachers.

Relative to some of the modern high-level languages, AGL may be considered to be over-simplified, particularly in view of the lack of dynamic array declarations and sub-routine facilities. The AGL sub-set of ALGOL, however, is only intended as an *initial* teaching language, with the emphasis on the *first principles* of programming. This will be reflected in the early exercises which the students will attempt, and for which the more sophisticated concepts are unnecessary.

In some respects it may even be beneficial in the long run for the student to learn the basic principles at such a low level. For example, two-dimensional problems can be programmed in AGL by including the necessary instructions to access the data when stored one-dimensionally. Similarly, a simplified form of sub-routine structure can be created by introducing the concept of „return links", and using switch jumps to return to the main program. The experience and knowledge thus gained will undoubtedly help the student when he progresses to the refinements of full ALGOL or other high-level language.

The AGL Reference Manual for Atlas

A reference Manual has been written for people wishing to actually use AGL on Atlas (RIDING, 1969). This gives a more detailed description of the language, with examples of complete programs and results, and more details regarding Job Descriptions. „numerical equivalents", etc. Copies may be obtained, on request, from the University Computing Service, Manchester University.

Acknowledgements

The author wishes to thank his colleagues in the Department of Computer Science, Dr. J.S. Rohl in particular, for helpful comments during the course of this work.

REFERENCES

BIRKIN, F.J. (1969). „Sixth form computing groups in Cornwall", The Computer Bulletin, Vol. 13, p. 80.

BROOKER, R.A., MORRIS, D. and ROHL, J.S. (1967). „Compiler facilities in Atlas Autocode", The Computer Journal, Vol 9, p. 350.

BROOKER, R.A., RHOL, J.S., and CLARK, S.R. (1966), „The main Features of Atlas Autocode", The Computer Journal, Vol 8, p. 303.

HOWARTH, D.J., PAYNE, R.B., and SUMNER, F.H. (1961). „The Manchester University Atlas Operating System, Part II: Users' Description", The Computer Journal, Vol 4, p. 226.

I.C.T. (1964). „A Primer of ALGOL 60 Programming for the Atlas Computer", I.C.T. Ltd., CS 379A.

KILBURN, T., HOWARTH, D.J., PAYNE, R.B., and SUMNER, F.H. (1961). „The Manchester University Atlas Operating System, Part I: Internal Organization", The Computer Journal, Vol 4, p. 222.

MORRIS, D., and ROHL, J.S. (1967). „The Atlas compiler system", The Computer Journal, Vol 10, p. 227.

NAUR, P., et al (1963). „Revised Report on the Algorithmic Language ALGOL 60". The Computer Journal, Vol 5, p. 349.

NIXON, W.L.B. (1969). „Experience with BASIC". The Computer Bulletin, Vol 13, p.4.

RIDING, G. (1969). „AGL Sub-set of Algol on Atlas", Manchester University.


```
FUNCTION:-      sqrt | sin | cos | arctan | ln | exp |
                abs | sign | entier
CHAR:-          LETTER | DIGIT | SYMBOL
STRING:-        CHAR --- CHAR
JUMP:-          goto LABEL | goto NAME [SUBSCRIPT]
FOR STAT:-      for NAME' := OPND' step OPND' until OPND' do
DUMMY STAT:-    NIL
BASIC STAT:-    VARIABLE := EXPR                        |
                VARIABLE := FUNCTION (OPND)             |
                VARIABLE := read                        |
                NAME'    := readch                      |
                printch (OPND')                         |
                print (OPND, INT, INT)                  |
                space (OPND')                           |
                newline (OPND')                         |
                write text (« STRING »)                 |
                comment STRING                          |
                JUMP                                    |
                if OPND ¤ COMPAR ¤ OPND then JUMP       |
                FOR STAT & BASIC STAT                   |
                FOR STAT & COMPND STAT                  |
                DUMMY STAT

STAT:-          LABEL: BASIC STAT | BASIC STAT
STAT LIST:-     STAT; --- ; STAT
COMPND STAT:-   begin STAT LIST end
PROGRAM:-       begin DECLAR LIST; STAT LIST end;
```

TABLE 1 - DEFINITION OF AGL

```
LETTER:-        a | b | c | d | e | f | g | h | i | j | k | l | m |
                n | o | p | q | r | s | t | u | v | w | x | y | z
DIGIT:-         0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
SYMBOL:-        ( | ) | , | * | / | ≤ | < | > | ≥ | = |
                ≠ | + | - | . | ≠ | [ | ] | ; | « | »
INT:-           DIGIT --- DIGIT
CONST:-         INT | INT.INT
NAME:-          LETTER
LABEL:-         L & INT | l & INT
NAME LIST:-     NAME, --- , NAME
ARRAY:-         NAME LIST [0 : INT] | NAME LIST [1 : INT]
ARRAY LIST:-    ARRAY, --- , ARRAY
SWITCH LIST:-   LABEL, --- , LABEL
DECLAR:-        integer NAME LIST | real NAME LIST |
                array ARRAY LIST | switch NAME := SWITCH LIST
DECLAR LIST:-   DECLAR; --- ; DECLAR

SUBSCRIPT:-     INT | NAME' | NAME' + INT | NAME' - INT
VARIABLE:-      NAME | NAME [SUBSCRIPT]
OPND:-          VARIABLE | CONST
OPTR:-          + | - | * | /
SIGN:-          + | -
EXPR BASE:-     OPTR & OPND --- OPTR & OPND
EXPR:-          SIGN? & OPND & EXPR BASE?
COMPAR:-        = | < | ≤ | > | ≥ | ≠
```

TABLE 2 - AGL PUNCHING FOR 5-TRACK TAPE

	Written	Punched
AGL5 . AGLE	real	'REAL'
	;	?
	«	(-
	»	-)
	[	((
	]	))
AGL5 only	:	↓
	≠	ø
	a < b	B > A
	a ≤ b	B ≥ A
	[	≰
	]	→
AGLE only	≠	@
	a > b	B < A
	a ≥ b	A > B
	a ≤ b	B > A
	a ≠ b	A ≠ B

N.B. On AGL5, > is treated as ≥

TEACHTRAN: A TEACHING LOGIC TRANSLATOR LANGUAGE FOR EFFECTIVE MULTI-PURPOSE CAI SYSTEMS

Junji Tsuda, Chikafusa Hirano, Nobuyuki Takeichi and Takeo Miura

1 EDUCATOR'S PROGRAMMING LANGUAGE

1.1 *Assumed hardware configuration:*

Before going into the detail of the programming language, a brief description of the assumed hardware configuration of the CAI system will be given.

The CAI system is an on-line, real time multiprogramming system for individualized education of large numbers of students. The computer to be used in this system has to be provided with on-line, real time processing features including the hardware interrupt structure. In addition, fast-speed, large-capacity memory, such as drum and disc, are also required as on-line auxiliary memory.

Terminal devices for students are assumed to be as follows:
typewriter, random access slide projector, random access audio tape recorder and CRT with keyset and light pen; the CRT display device has character generation and vector generation functions. The large variety of terminal devices can be selectively used in the most effective combination for the educational purpose.

As a means of student response, keyset and light pen are considered. The keys on the keyset are divided into two types: those used by the student for entering constructed response to questions and those used by the student to control his progress through the lesson material. The former includes alphanumeric characters and various special symbols, and also Kana characters which can be used for education in Japanese. The latter includes eight types. Function of each type will be described in detail in the next section.

1.2 *Teaching Methods:*

The current CAI system accomodates a variety of teaching modes:
linear, branching, tutorial and inquiry. Fig. 1 schematically shows a complete set of teaching rules permitted in this system.

The material presented to the student is classified as question and comment according to the content.

The question expects an answer from the student. The comment expects an understand-answer as to whether the student understands it or not. In Fig. 1, Qi denotes the question presentation, and ci comment presentation.

The lesson material is organized into three types of sequences:
(1) the main sequence which is the main route of lessons set by the educator and has a possibility of being passed by all students, (2) the hint sequence which is provided for the students who have difficulty with the questions in the main sequence, and (3) the call sequence which is given to any student who wishes to do extra and more challenging work at will. Within each of three sequences, the same teaching rules apply. In the call sequence, the student is able to call other sequences and, in this way, proceed through the material selected by himself until he is satisfied.

Now, teaching rules will be explained in detail. If the material presented is a question, the student usually makes his answer to it. He constructs an answer message by using the keys on the keyset. When the answer message has been constructed and the student pushes the ANS key, this is read as the answer into the computer. Until the ANS key is pushed, the student is able to correct the answer to his satisfaction. This is important from the man-machine interface viewpoint, and to realize it, an editing program for student input is incorporated into the system software. Moreover, the student is able, if required, to answer by indicating positions on the CRT screen by means of the light pen. Upon receipt of the answer, the computer program judges the content and selects an appropriate next step.

In provision for such a case when the student fails to find the answer, the hint sequence can be provided. The student requests it by means of the HINT key. At the end of the hint sequence, or when the student has found the answer on the way and pushes the OK key, the teaching sequence returns automatically to the original question. Then, the student makes his answer and proceeds in the main sequence. If the student cannot answer at last, a program is available by which the student is able to request the correct answer. To request the correct answer, the student must use the CA key. The REV key can be used to request returning to the previous step in such a case when the student wishes to review the comment presented in the previous step. Since, if the HINT key, CA key and REV key are abused by the student, there may be adverse educational effects produced, the educator is able to specify by a program as to whether the use of these keys should be permitted or not. If the use of any key is not permitted, but the student requests by means of that key, an appropriate notice is given and the teaching sequence does not progress.

The educator is able to limit the response time of the student. In this case, if the student fails to answer within the response time, the TIME UP condition is satisfied and corresponding branching is done.

The CALL function is provided for the following educational methods and is expected to be effective. In the education given by the main sequence, some students may wish to do extra work and the others may wish to obtain a more detailed comment than presented. In the mathematical education, it may become necessary to consult formulae or numerical tables. In these cases, the CALL function provides a means for the student to freely arrive at

the material he wants. For this purpose, at the same time that a question (or comment) is presented to the student, a list of materials available is also presented so that the student is able to select any material he desires from it. The student types in the name of the material desired and pushes the CALL key, and the material is furnished. The call sequence may comprise either a single material as in the case of furnishing the numerical table or a series of materials as in the main sequence. In the call sequence, the student is able to call the material of other sequences. Thus, the student is able to solve the problem given, working with the materials selected at his will from those prepared. To return from the call sequence to the original step in the main sequence, the RETURN key is used.

Unlike the case of question presentation to a comment presented, the student need not answer. When the student has understood the comment well and pushes the NEXT key, he is allowed to proceed to the next step. Since no answer is needed, the ANS key, HINT key and CA key associated with answering are not used. The other keys may be used in the same manner as in the case of question presentation.

Except for fixed destinations such as 'reverse' and 'call', the educator is able to prescribe arbitrary branching by a program for any student response. The conditions for the branching are the content of the student response and the history of his work (scores, counts of incorrect answers, etc.). Time-up is considered also as a kind of student response.

It is an outstanding feature of this CAI system that, by making use of these teaching rules, effective education can be realized such that the text written by the educator is not forced on the student, but various requests arising on the part of the student in each situation can be accepted. Although the requests of the student are acceptable, the work of the student must still be under the direction of the educator and necessary materials be prepared and arranged to ensure effective use by the student.

2. DESCRIPTION OF TEACHTRAN:

2.1 *Statements*

TEACHTRAN has been developed as an educator's programming language in which the educator can write the teaching rules mentioned above easily. A special consideration is given so that the teaching sequence prepared by the educator can be directly transferred to the computer program in an intuitively easy to understand form. A general teaching flow chart for CAI is shown in Fig. 2. Four stages are included: (1) question (or comment) presentation, (2) student response to it, (3) judgement of student response and branching, and (4) evaluation on each branching (for instance, loss of scores due to incorrect answers, count of incorrect answers, etc.). Correspondingly, statements in Teachtran are divided into four types: presentation statements for (1) and (2), branching statements for (3), assignment statements for (4), and some miscellaneous statements.

2.1.1 Presentation statements

The presentation statement specifies the content of the material presented to the student and the conditions for answering by the student. It is slightly differentiated between question presentation and comment presentation. The statements for the question presentation are as follows:

(1) Qi SLIDE: j
(2) TAPE: k
(3) (C) TYPE: "ABC...." (m, n) / (m′, n′)
(4) (C) DRAW: (x_0, y_0) (x_1, y_1) ... (x_n, y_n) / (x_{n+1}, y_{n+1}) ... (x_m, y_m) / ...
(5) TIME: 1 SEC
(6) ANS: "ABC ..." / "A′B′C′ ..." / ...
(7) ANS: $(x_0 \sim x_1, y_0 \sim y_1)$ / $(x_2 \sim x_3, y_2 \sim y_3)$ / ...
(8) REQ: HINT/CANS/REV

Qi on line (1) is the statement number. It is given at the head of a series of statements which specify the question to be presented, and shows the number of the question. The statement at line (1) specifies that the jth frame should be presented by the slide projector. The statement at line (2) specifies that the kth voice should be presented by the tape recorder. The statement at line (3) is either TYPE or CTYPE, and specifies that the message braced by quotation marks should be displayed by the printer or CRT. In the case of CTYPE, the previous display is cleared before a new display is given. (m, n) shows that the typing should be started from the mth line nth row position on the screen. (m′, n′) shows that a display of student input message should be started from the m′th line n′th row position. The statement at line (4) specifies that a polygonal figure formed by connecting coordinate points specified in the subsequent list should be displayed. These statements are concerned with the content of the material to be presented, while statements at lines (5), (6) and (7) deal with the conditions for student response to the question. The statement at line (5) is used to limit the response time of the student. The statement at line (6) lists up all student answers expected by the educator and coincidence of the student answer with the expected one is judged. Expected answers are called ANS1, ANS2, ... in the order in which they are arranged. The statement at (7) is used when the expected answer is a position on the CRT screen, and $(x_0 \sim x_1, y_0 \sim y_1)$ means a square region with $x_0 \leqslant x \leqslant x_1$ and $y_0 \leqslant y \leqslant y_1$. If the position indicated by the student falls within this square region, it is considered that the student answer is in coincidence with the expected answer. The statement at line (8) specifies whether the hint request, correct answer request and reverse request by the student should be permitted or not for the question.

Presentation statements for comment presentation are the same as the statements given above, except that ANS and REQ: HINT/CANS statements are not used because the student need not to answer, and Ci is the statement number.

2.1.2 Branching statements

Execution of the presentation statement is completed when any response condition specified is satisfied. Usually, it is followed by branching statements which select the material to be presented subsequently. The branching statement is a compound statement which is able to specify both conditional jump and evaluation functions simultaneously. This avoids the program becoming complicated if complex branching is required. Branching statements are as follows.

(9) (i) IF ANSn → Destination/Assignment statements (=D./A.)

(10) IF TIME UP → D./A.
(11) IF HINT → D./A.
(12) IF CANS → D./A.
(13) IF SWn → D./A.
(14) IF ACCn ≧ ACCn′(orN) → D./A.

(15) →D./A.

The general form of D./A. is as shown below.

Ci→ Statement number/assignment statement/assignment statement/ .....

i on line (9) is the statement number. IF ANS ri →D.A. means that, if the student answer coincides with the nth expected answer, a jump to the destination specified by D.A. is made, and at the same time, the branching is evaluated in accordance with assignment statements. ANS 0 is a special one assigned when the student answer does not coincide with any expected student answer at all. Statements at lines (10), (11) and (12) specify jump conditions of time-up, hint-request and correct answer request, respectively. In this CAI system, switches and accumulators for recording the history of the work of the student are provided. Statements at lines (13) and (14) specify jumping to be made with reference to this record. On (13) jumping starts when SWn is set, and on (14), the jump condition is determined by comparing the content of ACCn with ACCn' (or integer N).

The statement at line (15) specifies unconditional jump.

A detailed description of D./A. will be given below. The final destination is specified by the statement number part. Ci is called the inserted comment, and before going to the final destination, this comment is presented. The inserted comment can be used conveniently to insert routine sentences such as TRY AGAIN if the student makes incorrect answer or TIME UP when the response time is up. The part for the assignment statements is used to change the contents of switches or accumulators and contain an arbitrary number of assignment statements separated by slants (/). For instance, reduction of scores due to incorrect answers from the accumulator which holds the score is specified here. The form of assignment statements written here is as follows.

2.1.3 Assignment statements

Assignment statements are used to change the content of the switch or accumulator in the evaluation of student response or recording of the history of the work.

(16) #SWn: = 1 (or 0)
(17) SW: = 1 (or 0)
(18) ACCn: =± A ± B

A and B denote ACC n' or integer N. The statement at line (16) specifies the nth switch to be set (or reset). The statement at line (17) specifies all switches to be set (or reset). The statement at line (18) specifies that the value calculated on the right-hand side should be assigned to ACC n.

2.1.4 Miscellaneous statements

(19) # lesson name
(20) remarks
(21) END
(22) HEND
(23) RECORD
(24) SPRECORD

The statement at line (19) defines the name of the lesson, and is given at the head of the program. By this name, the student calls the lesson to receive program services. Line (20) provides a means for the educator to put any comment in writing the program. It is independent of teaching. END is a control statement which shows the end of the program. HEND represents the end of the hint sequence. Upon execution of this statement, returning to the original question for which the hint was given is made.

A significant feature of this CAI system is the capability of completely recording the student response. This record includes the number of the question (or comment) presented, the key pushed by the student, the time required for the student to respond, the total time required, and the content of the switch and accumulator. These data can be processed at any time when the statistical information becomes necessary. Whether or not such a record should be made or not can be controlled by the program. RECORD at line (23) starts recording and SP RECORD at line (24) stops it.

2.2 *Example programs:*

In TEACHTRAN, the following complex teaching logics can be programmed easily.

```
(1)          ACC 1: = 0
(2)    Q10   SLIDE: 5
(3)          ANS: "BOOK"
(4)          ACC 1: = ACC 1 + 1
(5)          IF ANS 1 →10
(6)          IF ACC 1 = 1 →C10 →Q10
(7)          IF ACC 1 = 2 →C20 →Q10
(8)          →C30/ACC 2: = ACC 2 - 3
(9)    10    IF ACC 1 ⩽ 2 →Q11 / ACC 2: = ACC 2 + 5
(10)         →Q11
(11)   C10   TYPE: "NO, TRY AGAIN"
(12)         TIME: 2 SEC
```

As the statements at lines (2) and (3) show, the 5th frame slide is presented to show the content of the problem. BOOK is the expected answer which is assumed to be correct. ACC1 is used to count the number of times of student response, and ACC 2 is used to count the score of the student. Before presenting the question, ACC 1 is cleared (line (1)), and when the student response comes, the content is increased by 1 (line (4)). At line (5) if the student answer is correct, jumping to the statement at line (9) is made. If the student answer is incorrect, control proceeds to line (6).

At line (6), in the case of the first time of incorrect answer, control proceeds to C10 where TRY AGAIN is printed out. Two seconds later, Q10 is presented again. In the case of the second time of incorrect answer, at line (7), control proceeds to C20 where an appropriate comment to aid the student in finding the correct answer is given. Then, Q10 is presented once again. The detail of C20 is not shown above. In the case of the 3rd time of an incorrect answer, at line (8), control proceeds to C30 unconditionally, and at the same time, the score is decreased by 3. When the student makes the correct answer, control proceeds from line (5) to line (9). If the correct answer is obtained by the time of the second try, the score is increased by 5 and the next question Q11 is given. Otherwise, the score is unchanged and control proceeds toQ11 (at line (10)).

3. FUTURE EXTENSIONS

As shown above, such sophisticated educational programming that has complex branching sequences can be realized easily by means of TEACHTRAN. For further replenishment, however, the following points must be investigated.

The first problem is to enable the educator to program processing of the message invoiced by the student. With the current design of TEACHTRAN, the basic match type

prepared by the system is a direct match which is a one-to-one correspondence between the students reponse and the educator's anticipated response. To facilitate other match types than direct match, TEACHTRAN has AID function. The statement AID n causes CAI to call and execute a machine-language subroutine residing in the system. This function permits the educator to add new match types as well as other new capabilities to the system. The only disadvantage is that AID subroutines must be written by a person familiar with the internal structure of CAI and machine-language programming technique. If the educator wants to have direct access to the student answer for examining it, statements for processing character strings should be provided, together with reinforcement of logical and arithmetic operations. As a means of referring to the character string input by the student, a fixed name, for instance, ANS should be prepared. In this connection, it will also be necessary to adopt variable names in place of the fixed names such as SWn ACCn in current TEACHTRAN, so as to enable the educator to define the variables freely. A question arises here however, as to whether the educator is able to take full advantage of these functions.

Next, at present, the record of the work of the student is not transferred between lessons. In some cases, however, it may be effective to refer to the results of preceeding lessons up to the last time.

Furthermore, with the assignment statements now available, it is difficult to carry out such complicated calculations as retrieval of statistical information from the result of the work of the student and to control the students' lesson course on-line by the results. At present, however, it cannot necessarily be said effective to use such a detailed educational information on-line.

4. SYSTEM IMPLEMENTATION

We have developed a concrete CAI system based on the above mentioned idea. This system utilizes a small scale computer Hitachi's HITAC-10 as the central processor unit. The computer has a core memory of 32,768 words (16-bit word, 1.4 US cycle rate). A disc is used as an on-line auxiliary memory. In spite of small scale, this computer is designed to have excellent on-line, real time processing functions and is extremely low in cost, and therefore, is quite suitable for the experimental CAI system.

The current CAI system has four student terminals, each of which consists of typewriter and random access slide projector. Random access audio tape recorder and CRT of teaching machine-oriented type and low cost are under development and will be added in the near future.

The computer capacity is enough for about 50 students to be instructed simultaneously and individually. The response time of the computer in this case is considered to be within 3 seconds.

4.1 *Operating systems*

As shown in Fig. 3, the software system of this CAI system consists of an on-line lesson monitor and several off-line utility routines such as compiler, record output routine, etc., which are controlled by the system monitor. The computer operator can control the entire system by typing in commands on the console typewriter. The system monitor, receiving the command from the operator, loads the core with the corresponding routine to execute it. In addition, it has an interrupt control routine for various on-line control services.

The educator's program is read into the computer, converted to an object program by the compiler, and registered on the library. The student will select the desired lesson from the library. When the lesson is over, the record output routine outputs a record of the work.

The CAI system is an on-line multiprogramming system giving simultaneous and individualized instruction for large numbers of students. Thus, inside the computer, many different programs for different students will be run simultaneously. To accomplish this with only small core, such a method is used in that each lesson is stored on the disc in the form of a library, and only the needed part is taken out at a needed time to the core for execution. The lessons are compiled into object programs of 512 word blocks. The disc-to-core transfer of the program is carried out by each block. The object program is relocatable to permit effective use of unoccupied area in the core and also, it is re-entrant so that a plural number of students can use it simultaneously. In this connection, a data area called status block is provided for each student. This area accomodates I/O data, switches and accumulators and is used for storage of various status information. Various queues generated inside the computer are constructed as chains of status blocks. The lesson monitor is a program for job scheduling, I/O control, core area maintenance, interrupt control, etc., required to ensure smooth progress of lessons. Recording the work of each student is another task of the lesson monitor; upon receipt of the student response, it checks whether recording is necessary or not and, if the necessity exists, the record is made on the disc.

4.2 *Using the CAI system*

Now, the procedure for operating the CAI system will be explained. It is necessary to minimize the procedure required for the student to initiate a lesson. The student first pushes the CALL key to call the computer. Then, as requested by the computer, he keys in the desired lesson name and his name. Thereafter, the lesson is started automatically. The end of the lesson is effected automatically by the educator's program. If the student wants to stop on the way, he is only required to key in END and push the CALL key.

The educator can make preparation for a lesson by inputting the command through the console typewriter. When the educator calls the system, the system responds by TYPE IN COMMAND. Then, he keys in COMPILE to call the compiler to the core. The TEACHTRAN lesson is read into the computer. In the course of compilation, checking for syntax errors is done and, at the same time, a list of various pieces of information on that program is printed out. For any perfect program which has been checked out for errors and does not require the output list, the command NLCOMPILE is used. At the end of compilation and with no error detected, COMPILATION OK is printed out and the program is registered in the library. Then, as LESSON is input, the lesson monitor is loaded on the core and the system is ready for starting a lesson. Now, the student can make a lesson call as described previously. When the lesson is over and the educator types in ORECORD, a record of the work of the student is output. Moreover, using the command such as PLIBRARY, DLIBRARY or DRECORD, the educator can take listing of lessons registered in the library, cancel the existing lessons, or cancel the student's record.

5 CONCLUSION

The CAI system presented in this paper is characterized by the fact that is uses a small scale computer, and is designed to permit experiments on as much variety of educational methods as possible with a view to research purpose. The educator's programming language described here has been developed along these lines. The opinions of field educators regarding this language is that 'at present, the technical level of the educator regarding the programmed instruction is not so high and it is a question of whether such a high branching function as prepared in this language can be utilized fully'. In this respect, a need is felt that, along with the development of such a CAI system, the study of educational algorithms including the computer teaching should be pushed in the field of education.

Our subject in future is to collect the data on actual experiments with this system and to make efforts for improvement and optimization of the system. Urgent problems to be solved are: (1) techniques for teaching program preparation, (2) cost reduction, (3) evaluation of effects of the CAI system and (4) establishment of the preparedness on the part of educators.

REFERENCES

(1) Gustave J. Rath, 'The Development of Computer-Assisted Instruction', IEEE Transactions on Human Factors in Electronics VOL. HFE-8, No. 2, June 1967

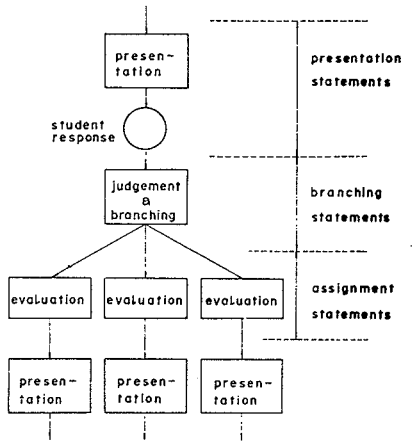


Figure 2 General flow of teaching sequences

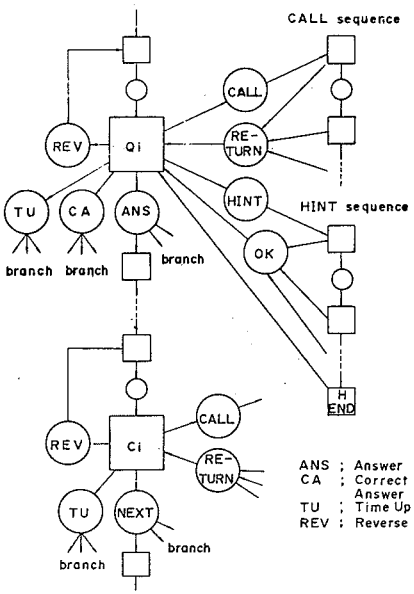


Figure 1 Flow diagram of teaching rules

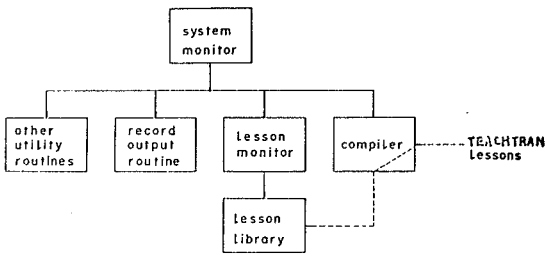


Figure 3 Software system configuration

PROGRAMMER COMPATIBILITY

A.J.A. Vermijs

1. THE PROBLEM

It is well-known that there is practically no computer program readily understandable to programmers who have not taken part in its design. Even if these programmers belong to the same team and are familiar with the type of coding used, it is often found that only the author himself can grasp it. Sadly enough, the information in the program appears to be thoroughly disguised. The author is the only person who knows his way about this maze and none of his colleagues will venture into it in an attempt to retrace his steps.

Actually more effort goes into the familiarization with an existing program than into re-designing the same program from scratch. Minor functional changes to be implemented in programs made by others have often led to complete re-programming of the original problem. An unknown number of duplications arises from the fact that programs written by other people for similar problems are not understood.

One important cause of this unfortunate situation is, in many cases, the lack of program documentation. Program listings are, by their very nature, not ideal program descriptions, yet the additional documentation so badly needed is failing as often as not.

However, the principal cause, in our opinion, is the fact that programs are not organized systematically. To accomplish our goal of understandability, standards have to be created in data and program structure and these standards have to be enforced upon all people in the group.

It would be unfair to say that nothing is being done about it; flexibility and modularity are treated more or less as quality standards; what we want is not to reject such standards but to add new ones in order to promote program interpretability and achieve compatibility between programmers.

It is evident that this compatibility has its limitations. To understand a program one needs knowledge of the conventions used as well as of the type of coding and of the machine. Yet the principal step towards understandability is to have conventions defined and adhered to by the programs actually written.

We propose to contribute to this by evolving a philosophy covering some problems of data and program structure.

2. EVOLVING A PHILOSOPHY

The system which will be described here will yield its full profit only if it is set up at the beginning of the development of the software for a new computer. The correct time to agree on basic conventions which can work through into all the software that has to be developed, is before any program whatever has been created for the particular machine. If this point is followed in practice, a high degree of uniformity will be achieved in all sorts of programs, both in support programming for program design and in the making of test programs and application programs.

It is essential that the structure to be enforced embraces both data and programming. With the detailed hardware data of the new machine as the point of departure, the first goal will be to structure that hardware data which influences the forthcoming build-up of programs. Typical cases in point are the command codes for peripherals, configuration data, etc. There is hardly any program which does not use this data. On top of this, conventions have to be defined on data processed or generated by the programs. This is an endless task, it is true, but the vital thing is that the data structure for a given problem be modelled at an early stage. All too often the definition of the data structure is a gradually evolved by-product of coding so that the final result lacks system and consistency. The same goes for the programming structure. It is necessary to define quite generally what the program structural elements are and how they hang together.

For the purpose of illustrating these general considerations, we shall work out an example in the following chapters. A few problems will be dealt with which concern input, output and program coordination for a given computer system. The example will try to produce conventions which remain within the assembler symbolic code. Without pretending to have worked out procedures which are of universal application, the author is yet convinced that the standards laid down below provide a foundation for a progamming system which will yield readily understandable programs within its own limitations.

3. GENERAL SYMBOL DEFINITIONS

Before any description can be undertaken we have to give some general definitions for system labels.

For ease of presentation it is assumed that the assembler used can handle labels of at least 8 alpha-numeric characters. It is important that system labels are standardized for all programs. This prevents the same kind of information from being labelled differently by different programmers, thus doing away with the first cause of misunderstandings. A typical set of conventions as meant here is now given.

The first character position is reserved for identifying the type of memory address:
P for absolute addresses
T for relative addresses
K for routine start addresses
S for subroutine start addresses.

Character positions 2 and 3 are allocated to a mnemonic symbol for an I/O system or a program, e.g.
CP for card punch
CR for card reader
DR for magnetic drum unit
DS for disk store
KB for keyboard
MC for master control program
PP for paper tape punch
PR for paper tape reader
TW for typewriter, etc.

Positions 4 to 6 are available for mnemos to identify functions, data fields, etc. such as:
CHN : channel number
CTR : control
PRG : program
RST : request status
SRG : subsystem register
SSN : subsystem number
TST : test
WRT : write, etc.

Positions 7 and 8 are reserved for entry and exit specifications in routine and subroutine labels.

4. A COMMAND AND DATA STRUCTURE FOR I/O OPERATIONS

4.1. *I/O Commands*

The majority of computer designers do work along logical and well-defined lines. Their systems are modular and many of the system modules have a general purpose structure. I/O channels, for instance, are suitable for connection of control equipment for a large variety of I/O devices. The I/O addressing and command structure also have similarities for different types of subsystem.

In this light it is amazing to see how often this hardware system concept is ignored by programmers. When programming I/O routines for the different types of peripherals, they do so in different ways, thus ending up with a multi-coloured set of I/O programs.

To demonstrate how it is possible to avoid such divergence from a pattern, let us use the I/O concept illustrated in Fig. 4.1.

A Device Controller and the associated Device are jointly designated as a subsystem.

Generally speaking, the following elements are of importance for I/O operations with any subsystem:
- type of operation
- configuration data of the subsystem, such as channel number, interrupt entry, etc.
- what kind of status of a channel or subsystem equipment is required before the operation is performed
- what kind of status is correct after the operation is performed
- additional specifications such as time-out settings, interrupt request expected, etc. etc.

These elements differ for different types of operations involving various types of subsystems. What they have in common, however, is the basic sequence for commanding the various units of the I/O structure. The nature of the channel/subsystem configuration dictates a logical series of commands for performance of a certain I/O operation. The actual content of a command depends on the individual type of subsystem and operation involved but there is a general layout fitting all program activities for I/O operations.

All these facts provide us with the fundamentals for the creation of a **universal I/O function**, using subsystem-dependent command tables and subsystem registers.

4.2. *Command tables*

Any I/O operation can be defined with a table of appropriate commands and additional information. The general layout of the table may be as follows:
- Command code for status request to channel control
- Correct channel status code before operation
- Command code for status request to subsystem
- Correct subsystem status before operation
- Command code for channel operation
- Command code for subsystem operation
- Time-out setting for channel
- Time-out settings for subsystem
- Ready interrupt expected indicator
- Correct channel status after operation
- Correct subsystem status after operation.

The content of this table is formed by the command codes for output to the channel and subsystem, the correct responses of the units involved and the additional indicators and data.

For each type of operation for each type of subsystem a command table is generated. The tables are all composed in accordance with the standard format, and together they are a systematic reflection of all I/O hardware specifications. It is easy to indentify a particular table, using the conventional system labels; for example:

PMTWRT for magnetic tape write operations
PDRRED for magnetic drum read operations
etc. etc.

All items in the command tables are labelled, relative to the table head address P..... as exemplified below:

PMTWRT: head address of table for MT write operation
TMTCRS: request status command for channel
TMTCST: correct channel status
TMTSRS: request status command for subsystem
TMTSST: correct subsystem status before actual write command
etc. etc.

4.3. *Subsystem registers*

Data related to each physical subsystem are brought together in subsystem register. This comprises configuration data such as subsystem number, interrupt request entry, actual subsystem status, data on the transfer being executed like block length, start address, etc. The registers are labelled per system as, for example, PMTRSG1, PMTRSG2, etc., and consequently each item in a register is labelled, relative to the head address:

TMTCHN = channel number
TMTSSN = channel connection number (subsystem number)
TMTITC = interrupt connection of this subsystem
TMTCAR = start address in memory for current data transport
TMTBLC = length of current data block.

The assigment of the data to the registers follows the rule that comparable data for different registers are given the same line distance from the head address (T label). An

anonymous designation is then sufficient. Typical cases in question are the numbers of the subsystem-to-channel connections and the number of the subsystem itself (since each subsystem requires such numbers).

An example will make this clear.

TMTCHN = TDRCHN = TDICHN = ... = TSSCHN, where SS offers a label applicable to all the subsystems in the left-hand member of the equation. Please note that this equating function extends only to address positions relative to the head address. The contents of the various memory locations will inevitably vary.

4.4. *The I/O handler*

The careful preparation of the I/O data structure described above will yield a result as shown in Fig. 4.2. Per type of subsystem a set of command tables is available, one per operation, as well as a set of subsystem register, one per physical unit. For the actual execution of the I/O operation, all we now need is a single universal I/O function. This I/O handler is started by a higher order I/O routine and is given the head address of a command table and of a subsystem register. The I/O handler now inputs all the command codes of the table in the correct sequence and checks for correct operation by the use of the response codes listed. The remarkable thing about this is that we now have only a single I/O routine universal enough to handle all types of subsystem. All the specific data of the different subsystem types is available in well-defined tables and is no longer hidden in pages of forbidding code.

It is clear that a programmer familiar with this basic structure can easily grasp the logic in I/O operations for different subsystems and will have no difficulty in adding I/O operations or subsystem types: it is merely a matter of adding new command tables and subsystem registers.

This philosophy can be carried through readily to higher order I/O functions. The basic logic for such functions is organized in operation tables listing command table labels. Another universal I/O routine is now required to execute the operations listed in the new table, thus performing the higher order function. In this case, however, the tables have to contain also logic information on the type of action to be taken if a basic operation is not performed in the correct manner. The I/O function described first can make use of a universal error output, but this is of no use in the higher order routine. Even so, the kind of logic meant here is amenable to structurizing no less than the normal operation.

5. DEFINING THE ELEMENTS OF THE PROGRAM STRUCTURE

By elements of a program structure are understood here to be those sub-areas into which a problem area is broken up. The breaking-up of a problem area takes place in stages, giving rise to a number of levels. There is a hierarchical arrangement of levels, the top level presenting broad functional specifications of the highest order sub-areas whilst the lowest level accommodates the detailed specifications of the smallest functional units (routines).

Alongside this vertical structure, there is a horizontal one in the form of functions common to more than one sub-area; typical functions of this type are:

- master control program (monitor)
- I/O handler
- subroutines.

A number of representative definitions of the various sub-areas will now be given:

System – A complex of programs and data which together realize the specification laid down for a particular application field.

Program – A complex of routines, subroutines and data which together realize part of the specification laid down by system analysis.

System data – Permanent and/or dynamic data kept in registers and/or tables for use by all programs of the system.

Program data – Permanent and/or dynamic data kept in registers and/or tables for use by all routines of the program.

Routine and **subroutine** – A routine or subroutine performs a single logical function by means of a restricted number of instructions.

A **routine** is initiated at an entry by the monitor and returns via an exit to the same monitor.

A **subroutine** is initiated at an entry by a routine or subroutine and returns via an exit to the position defined by the initiator.

Routine label (start address) – The first character of a routine label is K. Positions 2 and 3 form the program mnemonic, positions 4 to 6 the function mnemonic.

Example: KAABBB.

Positions 7 and 8 identify a routine entry or exit.

For an entry the numbers 00 to 49 are reserved, for an exit the numbers 50 to 99.

Example of an entry: KAABBB02.

Subroutine label (start address) – The first character of a subroutine label is S. Positions 2 and 3 form a process mnemonic,

e.g.: TI= translation subroutine
SG = sorting subroutine
AC = arithmetic subroutine.

Positions 4 to 6 form the function mnemonic.

e.g.: STLBTO = translate binary to octal.

Positions 7 and 8 identify a subroutine entry or exit.

For an entry the numbers 00 to 49 are reserved, for an exit the numbers 50 to 99.

The functional breakdown of a program into routines and subroutines should be done as consistently as possible according to the functions required, and a clear distinction should be made between control functions (CTR) and executive functions (EXC).

6. MONITOR

In addition to the pains taken to structurize the data, it is essential to determine in advance how the programs are to be structured internally for optimum programmer compatibility. And this cannot be done unless the problem of program-to-program coordination is solved at a central point by a master control or monitor program.

For the conception of such a monitor, two objectives have to be kept in mind:

- The monitor should ensure optimum efficiency in the use of the available process time.
- The interplay between a great number of programs or program parts should be coordinated with due regard to the priority rules applicable to the system.

Let us assume that we have a computer system with a number of hardware priority levels which are controlled by a special microprogram. The tasks devolving on this microprogram are:

- To control progress autonomously with due regard to the various priority levels.
- Autonomously to control the saving operations for programs to be interrupted and the restore operations for programs to be started.

We shall further assume that each of the priority levels contains a number of programs which may differ in priority between them.

This interplay of programs should, as a logical whole, form the realization of an imaginary application field. The way in which the monitor coordinates this becomes clear from the following basic functions:

- The monitor presents a program with an information package (queue cell) picked from the associated queue.
- The monitor starts the program.
- The program takes over control and will, if not impeded, complete a program run.
- During a run the monitor may be called for work involving other programs. If the priority level of the program invoked is higher than that of the current program, the micro-program will instantly interrupt the running program, save the information so far acquired for a later re-start, and start the new program. If the called program is of lower priority rating, the monitor returns to the running program.
- Any program run is terminated by a jump to the monitor. The monitor will then allocate another program in the same priority level for activation, and the process described is repeated. If no other programs are on hand in the same level, the level in question is cleared and the monitor scans progressively lower levels until eventually the machine comes to rest in the lowest level.
- The process is started again by external activation.

When a program has been started on an information package, it may be unable to complete the entire work in one run because at some intermediate stage it has to initiate other programs for the purpose. The consequence of this is that the total work will be completed in another program run.

When a program is interrupted, the saving of the current data for a subsequent re-start may be implemented either in hardware or in software, but the hardware solution is preferable because it demands less central processor time.

The order of the activities in time may vary from one moment to the next, and there is no question of any sort of timetable to control activities. Accordingly, as the occupancy of each priority level changes, the monitor decides which program is to be activated first, giving precedence to priority rules over the 'first come, first served' principle. In this way several programs can be accomodated in the machine, even if they should have to work together on a purely *asynchronous* basis.

With this approach there is no problem in making a variety of programs cooperate with an input/output program, which is a situation that will crop up very frequently in practice. In this arrangement, each program may be considered as if it and the other programs it has invoked are the only programs at work in the system. This is possible because, after it has initiated an activity in a different program, it will receive a specific signal indicating that this activity has been completed.

When designing programs for this system, there is only one extra function to be kept in mind: the fact that each program continues to bear responsibility for the activities it has itself elicited.

This typical control function is now centralized per program in what we may call a control routine.

Calls which are filed in a queue for a particular program will, therefore, be of two types:

- initial calls
- report-back calls

From all that has been said above it is now possible to summarize the main functions of the monitor as follows:

- accepting sources of activity from a program at any moment — call
- storing sources of activity in program input files — function
- control of program priorities — initiate
- the starting of programs — initiate

Up to this point we have shown how the various programs are coordinated. There remains the question of laying down the cooperation between routines of one and the same program before it is possible to make the definitive draft of the data structure of the monitor.

6.1. *Routine linking*

During the building-up and test stages of a program, the routines of that program refer to one another not directly but via entry-exit tables. For each routine entry and each routine exit an entry or exit item has to be defined. The entry item should contain the start address of the routine entry whereas the exit item will be filed by a keyboard command with the address of the entry item of the routine to be started from the said exit.

The entry/exit items together with the mnemonic are defined in a table per routine. For each exit two instructions have to be defined, one to transfer the address given by the corresponding exit table to the program data, and one to effect the jump to the monitor. This extra data, the exit instructions and the entry and exit tables, is generated automatically during the assembly of the program and serves, at it were, as function test points during the program run.

All this enables the monitor to create an on-line flow trace function because, when it receives a report-back signal from an exit, it consults the exit table to find the mnemo of the function and the particular exit number, and next uses the same exit table to start the next routine. During the run of the program, this on-line function can be initiated by external command so that a document can be printed afterwards showing the successive functions performed. This – it will be appreciated – is a facility which greatly simplifies fault tracing.

The entry and exit tables together may be regarded alternatively as elements of a central distribution function for routines, the distribution instructions taking the form of KAABBB50 – KAACCC00. The instruction exemplified here has the effect of connecting exit 50 to entry 00 of function AACCC. With such instructions concentrated, it is quite easy for the monitor to modify a sequence of routines either during a test or during a program run. All this is illustrated in Figs. 6.1 and 6.2.

6.2. *Program linking*

A program is started by the monitor on an entry of the

control routine for that particular program. A special format is required for the entry table associated with the entry that is called by the monitor when the program is started for a run, for it is from this table that the monitor has to be informed of:

- the priority level of the program (TMCLV)
- the queue administration block (TMCQA)
- the start address of the program (TMCSAP)

in order that it can perform the call function.
This information is compiled in the program entry item.

The program initiate function of the monitor is worked out separately per hardware priority level.

All queue administration blocks of the various programs are linked together per priority level (QLA). This information is sufficient to determine which program must be initiated in accordance with the perference rules.
The monitor data and program structures are illustrated in Figs. 6.3 and 6.4.

6.3. *Subroutine linking*

We like to conclude this report on programmer compatibility by showing how subroutine linkage has been worked out in the program structure described.

A subroutine may have several entries, but it has only one exit. Because of this limitation, the various possible results are coded in a fixed index register; the routine or subroutine which invoked the subroutine under consideration collects the various results with a standard jump table. The transfer of data between subroutines is effected by putting the address of the data in another fixed index register prior to invoking the new subroutine.
A call for a subroutine takes the standard form of two instructions as examplified by:

1. SPC XMTUNL – save area
2. JMP SMTUNLOO – subroutine start address
3. REP – re-entry point.

SPC stands for: save program counter; this instruction saves the program count of position 3 by transferring it to an area outside the subroutine. Which area this will be is determined during assembly.

JMP refers to the actual jump instruction which specifies the address of the required entry label.

Please note that return jump addresses have not been located within the subroutine itself. There are two reasons for this:

- inclusion in the subroutine would lead to unnecessary memory loading, which is unacceptable particularly with very elementary functional subroutines;
- in addition, it would solve the problem of subroutine nesting only partly, viz., for one-level subroutines. By a general subroutine we understand a multi-level one, i.e., a subroutine which occurs singly in the memory and can be called from diverse priority levels, thus running the risk of multi-interruption. This implies that all the variable internal storage space of the subroutine has to be addressed differently for the different levels.
  This can be effected in an efficient way during the assembly so that the amount of core space is just sufficient to accommodate the most demanding nest of subroutines.

All elementary functions are implemented as subroutines. Of course the problem analysis will, in addition, give rise to functional subroutines which have a meaning within the framework of the application program or the programming system as a whole. Consistency in method will then lead to routines which are, to all intents and purpose, just logical concatenations of subroutines. For the implementation of the program this means that each routine requires only a very restricted number of variations in instructions. The bulk of instruction variation is then concentrated in the variety of subroutines.
All this makes for better readability of programs.

7. FINAL CONSIDERATIONS

7.1 If attempts to bring system into the activities of a programming department are crowned with succes, a major step forward has been made in simplifying a number of tasks inherent to the elaboration of an application program.
7.2. Systematizing the procedure in working out a program requires, as a starting point, a number of general models with a minimum of restrictions.
One is inclined to accept the request „Make a program which exactly meets these requirements" as logical, for such an approach promises optimum speed and efficiency in completing the task. There is a fallacy in this way of thinking, however, for a quite simple reason: there is hardly ever the certainty that the requirements laid down for a program will cover all contingencies and will need no modification. This fact forces one to conclude that all program development is built on shifting sand unless the ability to deal with this situation is kept in mind from the start. If the factors which affect the program parameters can be determined as early as in the problem analysis stage, the program can be built so that a possible source of variations can be rendered relatively harmless. Experience shows that a program built along these lines can easily compete in its demands for time and storage space, with a solution that deals only with the problem on hand.

APPENDIX

1. An overall diagram (fig. 7.1) is given to illustrate in a simplified manner the interplay between various part programs and the I/O handler (left-hand section).
2. This diagram will be used to visualize the principles laid down in the previous chapters. Let us, therefore, follow the routes indicated by the block diagram.
3. Program AA is started by the monitor. Input for program AA is a participant (queue cell) in the associated program queue. Program AA is started and discovers that an I/O operation has to be executed. To this end, it forms output data in a unique memory location for the running priority level. A subroutine (call other program) starts the monitor.
4. The monitor copies the output data into a queue cell (dynamic memory block) and next consults the queue administration from a program entry table. The queue cell is inserted into the particular queue chain and the level indicator for the I/O program is activated.
5. When this happens, the next event depends on the priority difference between program AA and I/O (micro-program of the priority circuit).

5.1. If the priority level of the I/O program is higher, there follows:
1 – interrupt of program AA
2 – start of I/O program.

5.2. If the priority level of the I/O program is not higher, the following sequence of events takes place:
1 – a return of the call subroutine to program AA.
2 – program AA completes the run for the participant and goes to a ready exit which brings us back to the monitor (initiate function).
3 – the monitor will next deal with all participants in all programs within the current priority level; when this is finished,

the level in question is cleared and the priority circuit (micro-program) can decide which lower level is to come next (In the situation under discussion this will be the I/O program).

6. The I/O program performs the following simplified functions:

6.1.– check if any one of the channels is free; if not, go to 6.6.

6.2.– if yes, find out if there are participants for the free subsystem; if running work, go to 8;
if new work, go to 6.5.

6.5.– identify the command table and instruct the subsystem in accordance with the table; go to 6.1.

(We now assume that the subsystem and channel take some time to complete the order and that each of them will indicate by an interrupt request that the order has been executed).

6.6.– go back to monitor; the monitor now performs again the same activities as described in 5.2.3.
After some time the interrupt request comes in from the subsystem (see „interrupt handling").

7. The interrupt handling function distinguishes between two types of interrupt:
1 – interrupts from the channels
2 – interrupts from the subsystems.

7.1.Channel interrupt:
1 – determine if the interrupt is expected;
if yes, clear the channel, report the I/O level and go to 7.1.2.
2 – go back to monitor; the monitor now performs again the activities described in 5.2.3.

7.2.Subsystem interrupt.
1 – determine if the interrupt is expected;
if not, call for alarm print-out and go to 7.1.2.
if yes, check from the command table whether any more commands have to be issued;
if yes, issue the commands and go to monitor;
if not, call for I/O level and go to monitor.

8. The I/O program will now be re-started and will, when analyzing the work in this situation, find that all I/O operations have been completed. The I/O program will now first clear the channel and the subsystem, after which it will report to the initial program that all I/O operations are completed.

9. Program AA will be started again at some later time, which completes the information cycle.

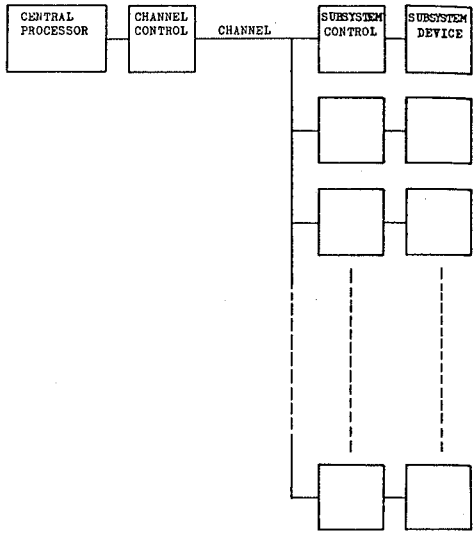


Fig. 4.1

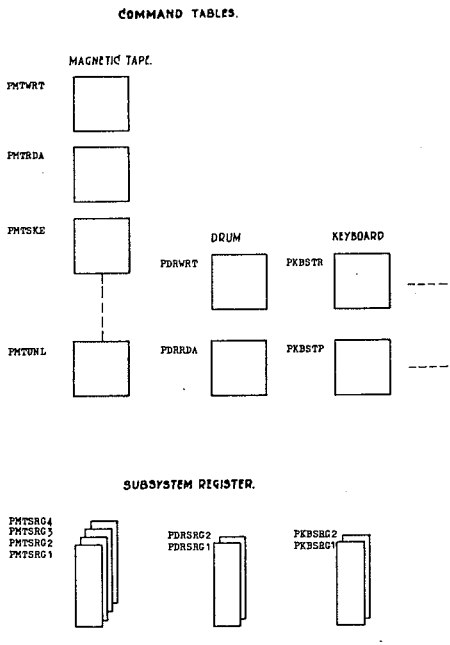
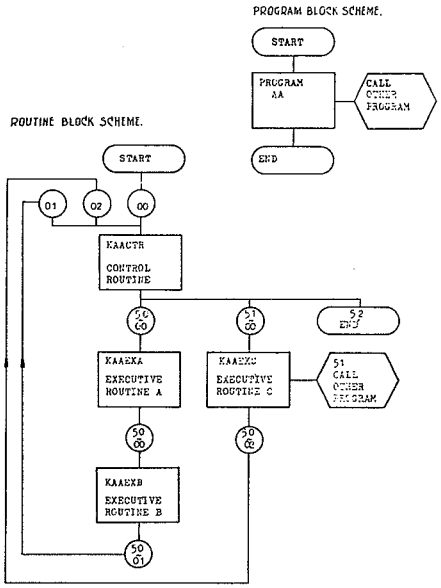


Fig. 4.2



- ORGANISATION FUNCTION

1 EXIT - ENTRY ON LINE FUNCTION TRACE
2 ROUTINE LINKAGE ON LINE.

Fig. 6.1

ROUTINE EXIT-ENTRY TABLE OF PROGRAM AA

		BEGIN OF TABLE/REFERENCE TO OTHER TABLE
	AA	PROGRAM MNEMO
	CTR	ROUTINE MNEMO
	0	NUMBER OF ENTRY 00 BINARY
PROGRAM ENTRY		REFER ADDRESS TO PROGRAM ENTRY TABLE
	1	NUMBER OF ENTRY 01 BINARY
D	KAACTR01	START ADDRESS ENTRY 01
	2	
E	KAACTR02	
	50	
	A	REFER ADDRESS TO CORRESPONDING ENTRY
	51	
	C	
PROGRAM READY-EXIT	52	
	0	BINARY ZERO REFERENCE PROGRAM READY EXIT
	EXA	ROUTINE MNEMO IN ASCII
	0	
A	KAAEXA00	
	50	
	B	REFER ADDRESS TO CORRESPONDING ENTRY
	EXB	ROUTINE MNEMO IN ASCII
	0	
B	KAAEXB00	
	50	
	D	REFER ADDRESS TO CORRESPONDING ENTRY
	EXC	ROUTINE MNEMO
	0	
C	KAAEXC00	
	50	
	E	REFER ADDRESS TO CORRESPONDING ENTRY
PROGRAM SUBEXIT	51	NUMBER OF EXIT BINARY
		REFERENCE TO ROUTINE ENTRY
		END OF TABLE

NOTE: This definition work is done in two phases:

1) Building up entry-exit tables during the assembly phase by means of a program
2) Connect exits to entry by means of keyboard commands on-line.

Fig. 6.2

Example routine entry-exit table

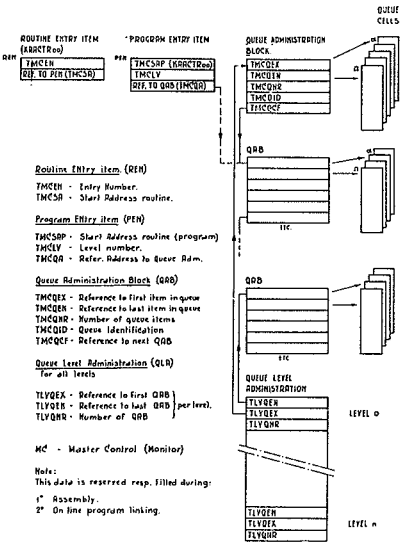


Fig. 6.3

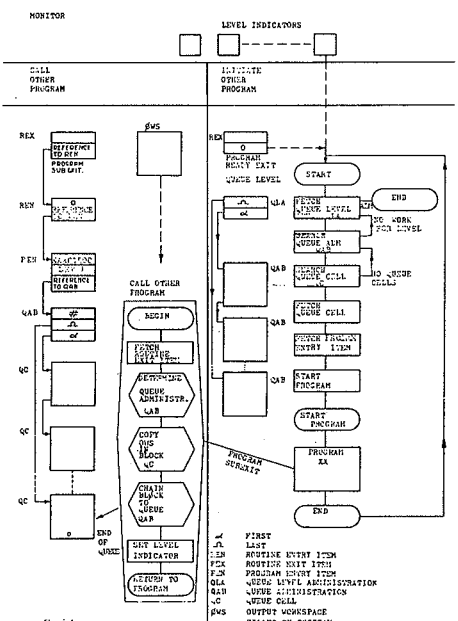


Fig. 6.4

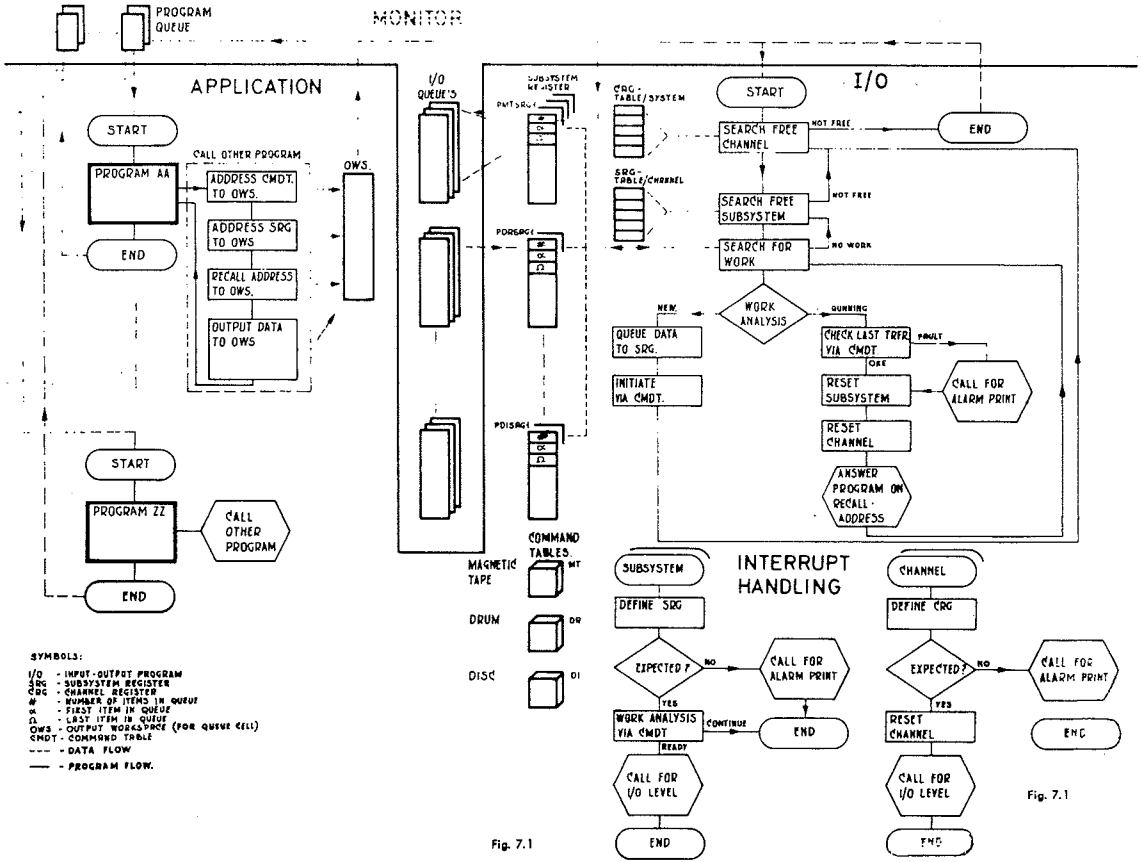

PROGRAM QUEUE
MONITOR
APPLICATION
START
PROGRAM AA
END
CALL OTHER PROGRAM
ADDRESS CMDT. TO OWS.
ADDRESS SRG TO OWS
RECALL ADDRESS TO OWS.
OUTPUT DATA TO OWS
OWS.
START
PROGRAM ZZ
CALL OTHER PROGRAM
END
I/O QUEUE'S
SUBSYSTEM REGISTER
PART SRG
PDRSRG
PDISRG
COMMAND TABLES.
MAGNETIC TAPE
MT
DRUM
DR
DISC
DI
I/O
START
SRG - TABLE/SYSTEM
SRG - TABLE/CHANNEL
SEARCH FREE CHANNEL
NOT FREE
END
SEARCH FREE SUBSYSTEM
NOT FREE
SEARCH FOR WORK
NO WORK
WORK ANALYSIS
NEW
RUNNING
QUEUE DATA TO SRG.
INITIATE VIA CMDT.
CHECK LAST TRFR VIA CMDT
FAULT
ORE
RESET SUBSYSTEM
CALL FOR ALARM PRINT
RESET CHANNEL
ANSWER PROGRAM ON RECALL - ADDRESS
SYMBOLS:
I/O - INPUT-OUTPUT PROGRAM
SRG - SUBSYSTEM REGISTER
CRG - CHANNEL REGISTER
- NUMBER OF ITEMS IN QUEUE
α - FIRST ITEM IN QUEUE
Ω - LAST ITEM IN QUEUE
OWS - OUTPUT WORKSPACE (FOR QUEUE CELL)
CMDT - COMMAND TABLE
--- - DATA FLOW
—— - PROGRAM FLOW.
INTERRUPT HANDLING
SUBSYSTEM
DEFINE SRG
EXPECTED ?
NO
CALL FOR ALARM PRINT
YES
WORK ANALYSIS VIA CMDT
CONTINUE
END
READY
CALL FOR I/O LEVEL
END
CHANNEL
DEFINE CRG
EXPECTED ?
NO
CALL FOR ALARM PRINT
YES
RESET CHANNEL
END
CALL FOR I/O LEVEL
END

Fig. 7.1

Fig. 7.1

INSTRUCTIONAL USE OF COMPUTERS; A CRITICAL EXAMINATION WITH RECOMMENDATIONS FOR ACTION

Karl L. Zinn and Susan McClintock

1. INTRODUCTION

1.1 *State of computer use in instruction*

Many different kinds of computer and software systems are being used by instructional research and development projects in the U.S. today, and over a dozen are offered commercially to institutions that dare to venture into operational status in this new field. Some small machines have been programmed for use by a single student or a small number simultaneously; larger systems handle up to 200 users spread throughout a city or across the country. Many of the research and development projects use general-purpose computer systems, imbedding the instructional applications among other software packages available to users, and the student may use the same terminal and operating system for a variety of teaching and learning aids.

Programming languages and systems show even more diversity than the hardware. Over 50 languages and dialects have been developed specifically for programming conversational instruction. However, many of their differences are superficial and some obvious needs of users still have not been satisfied.

The costs of using these operational or experimental systems and languages in education vary considerably. Figures reported range from $.25 to $35.00 per hour of student use at a learning station. One reason for the wide range is the variety in assumptions made about how many effective student hours can be scheduled in a month or a year, whether the equipment is rented, leased or purchased, and how much time will be spent in utility jobs, preventive maintenance, or repair. The mode of student use is a big factor in cost also, but the prospective buyer is not likely to be fully informed about this.

Computer-based learning exercises have been written in almost all subject areas and for many age levels: from drill in preschool reading to exercises in regional planning and resource management. Much of the material makes little use of essential computer contributions, and some of it would be as effective and certainly less expensive in another mode, for example, a book or film cartridge or two-person game.

1.2 *The need for the study*

In general, there is a great deal of duplication, and also much variety and rapid change. The costs of development and use of this new technology of education can be considerable, and improved communication of relevant, interpretable and current information is needed.

For example, one of the current trends is away from the programming of successive questions to be delivered by the

computer under strict control of the author's sequencing rules; and yet manufacturers joining the field are repeatedly producing their own version of the 'easy-to-use' author language for any classroom teacher to program just such applications.

Another trend is toward generalized, even 'generative' procedures which assemble curriculum material (questions and answers) as they are needed for each individual student; and yet educational technologists pursuing this course are not well acquainted with the accomplishments (and disappointments) in the related specialty of 'artificial intelligence' in computer science.

Project CLUE, along with other studies of current status and trends, is intended to aid those planning computer use in education.

1.3 *Objectives and intended audience*

The final report of Project CLUE is arranged in distinct sections to meet the different needs of three audiences: 1) students of educational technology and interested laymen, 2) educational administrators, teachers and instructional system managers, 3) educational psychologists and computer scientists.

To the student and layman, the report offers an introduction to the present performance, problems and potential of computer uses in education. Technical terminology has been minimized, in the summary and interpretive sections. Much of the specialized vocabulary for the field is clarified by familiar definitions and examples in a glossary. The administrator or teacher, already familiar with the debate about computers in educational technology, will find recommended directions for computer use in education, and background information on which the base project proposals and new programs. For the specialist in research and development of instructional uses of computers materials given in support of the recommendations provide detail on differing opinions regarding computer instruction, dimensions underlying computer use in education, and comment on system design. The index should assist in location of material on specific subjects, or opinion by certain writers.

2. SCOPE OF COMPUTER USE CONSIDERED

2.1 *The domain and a problem of names*

Project CLUE has considered the use of computers for instruction, primarily as a tool for learning and secondarily for research on learning and teaching. Information was assembled from primary sources into five major categories of the instruction-learning system: operations and costs; curriculum development; programming languages and instructional strategies; evaluation; and documentation and dissemination. The recommendations of the study can be applied to five areas of computer application: learning tools; management of records and materials; presentation of instruction; research; and exploration of new information processing techniques.

Administrative uses of the computer for record keeping, scheduling, resource allocation, and data processing were not included in survey and review activities and are mentioned only incidentally. Although these adminstrative uses are extremely important to educators, critical review of this area was not as urgent as for the instructional domain, and particularly for the on-line interactive uses, which are the subject of some controversy.

The scope of Project CLUE includes non-interactive (e.g., batch mode) uses of the computer for instruction. On-line interactive uses of the computer are often quite costly to implement at their present stage of development, and occasionally the same function can be served just as effectively and less expensively off-line. Some systems are arranged to provide results in 5 to 30 minutes from a 'remote job entry' terminal. The user does not have to carry his punched cards to the computer center but is able to interact with the machine using a card reader and printer located for his convenience.

The documents of Project CLUE should not be considered a report on just CAI or CMI or both. The uses of the computer as an instructional aid have been represented by a number of acronyms including CAI. CBI, CMI, CAL, CAE, and many others. These labels tend to become associated with various uses of computers or individual projects or vendors. As a result any single acronym can mean different things to different readers.

'Computer-assisted' or 'computer-aided instruction' (CAI), adopted by IBM in its early writing about instructional systems, is now the most commonly used acronym. System Development Corporation began with the label 'computer-based instruction' (CBI), which was later adopted by Stanford University and RCA. 'Computer-assisted learning' and 'computer-augmented learning', it has been suggested, are more descriptive of the variety of uses of the computer in education. However, the resulting acronym (CAL) is already associated with two computer languages.

If the computer's role is to assist the teacher in managing instruction by handling performance records and curriculum files, the label used is 'computer-managed instruction' (CMI). In this use, the student does not receive instruction directly from the computer. Rather, non-computer media are 'managed' or scheduled for him by his teacher with the assistance of automatic data processing. Perhaps a better label is 'computer-assisted instruction management' (CAIM).

Because acronyms tend to become associated with one manufacturer, computer system, or research and development project, they do not adequately label the entire field. For some observers the CAI label implies author control to the exclusion of the many possibilities for student initiative and direction of the computer processing during instructional activities.

Authors of Project CLUE interpretations have avoided all acronyms and when the CAI label is used it refers to the narrower definition of computer use, e.g., drill and tutorial modes. More comprehensive phrases are intended to indicate the wide scope of computer uses for instruction, for example, 'computer-based learning,' or the 'computer as a learning tool,' or 'interactive uses in instruction.' These place emphasis on the learning process rather than the more controlled instruction process; they point toward the opportunity to extend capabilities of the user when the machine is programmed intelligently.

Nevertheless, adoption of any acronym such as CBL or CLT has been avoided since it would only add to the confusion of initials and meanings.

2.2 *Dimensions of use*

Numerous categories have been devised to classify

interactive uses of computers: drill, tutorial, dialogue, socratic, simulation, learner-directed, etc. However, criteria used for such classification, or the underlying assumptions from which the categories have been derived, have not been clearly expressed. Therefore, it may be helpful to identify several dimensions of use and their relationships to each other, without attempting to be exhaustive or theoretically comprehensive.

2.2.1 Program control

Computer-based lessons differ in the control the writer has over the student's course of study. At one extreme the student can only follow the program; typically he finds himself more restricted working at a computer terminal than he would be with a textbook or set of drill exercises. That is, the writer of a tutorial exercise has not provided the computer instructions which would permit the student to look back, review or skip ahead as he does with printed materials.

Control is desirable in some situations, and curriculum designers have used this facility of the computer to reduce inappropriate skipping about or other distractions. Control is also an advantage when the lesson designer is testing his materials and wishes to know exactly what each trial student has seen and when.

At the other extreme, characterized by almost complete user control, the computer is programmed to serve the student as a tool in the management of the information necessary for solving a problem or achieving some other goal. The most common use is as a conversational computer; that is, for calculations and other immediate processing in response to directions from the student. A lesson designer can also give the student access to large files of information to retrieve and rearrange facts as useful for his study.

2.2.2 Diagnosis and prescription

Another dimension of computer use concerns the extent of data recording and interpretation in the interactive program structure (e.g., the program's facility in diagnosing difficulties and prescribing new tasks on the basic of student performance and attitude).

At one extreme the computer follows student instructions for problem solving or presents simple linear programs for practice exercises without paying any attention to the student's problem solutions or to his response to questions. At this zero level there is no interpretation or prescriptive feedback determined by the author's procedure to be displayed to the student or saved for future use by the program procedure itself.

In typical tutorial uses the computer program processess student responses and tries to act upon them during the instruction sequence by selecting for the student appropriate remedial or new materials. Such a program could also give interpretation of response measures directly to the student, e.g., telling him why he needs more practice with or without allowing him control over future action.

Problem solving environments, typically unconstrained, also may include interpretation of student actions, e.g., presentation of the correct form for an ambiguous instruction, suggestion of a solution procedure along a direction already taken by the student, and evaluation of his final solution for efficiency in terms of number of instructions or computer time used.

For any level of diagnosis and prescription, the extent of control over the direction of instruction may vary from nearly zero (where the student has almost complete control) to maximum control by the computer program. In other words, the program may tell the student why or on what basis he is being led through an exercise whether or not he is permitted to change it. Similarly, any level of control allows the full range of prescription.

However, if the computer is to have a justifiable role when programmed to maintain maximum control, it must be acquiring and interpreting information for use in the controlling procedure. Although the author may wish to have the computer record and store the maximum of relevant information justified by potential use in guiding students, he must design a procedure for diagnosis and prescription which passes along to each student only as much as promotes learning and favorable attitude, and avoid the distraction of too much information or irrelevant items.

2.2.3 Variety of functions available to user

The scope of opportunities for a user (whether student or author) in an instructional system may vary from a completely pre-programmed and limited exercise to the full capabilities of a richly-equipped and general-purpose system. The systems dedicated to 'drill and practice' are exemplars of a limited variety of functions available to the user: authors assemble pairs of items for presentation according to preset strategies, and students work on the exercises in sequence and within the interactive capabilities of the system. Perhaps authors may be allowed to adjust criteria for triggering a standardised-computer response of 'wrong, try again' or 'too long, try again', and students may choose among two styles of drill exercise.

Actually a drill-oriented system may allow for considerable 'variety' in the sense of programming capabilities or functions believed to be worthwhile for the instructional use of a system. A package of drill exercises may include alternative ways to achieve the same performance goals, and the author may write a complicated procedure to select among them on an individual basis using observed student learning characteristics and current performance.

Although drills of limited purpose measure high on 'program control,' they can also measure high on the dimensions of variety and prescription.

Variety is an obvious dimension in consideration of the content to be utilized in a dialogue, the programming languages available for solving a problem, the alternative models to be used as a basis for a simulation, or alternative techniques available to the student to reprogram the model underlying the simulation.

2.2.4 Amount of interaction

As a dimension underlying computer use, 'interaction' refers to a computer operation in which particular attention is given to the continuing exchange between user and program. Typically the user types some instructions, the computer types back partial results, the user modifies his original statement or continues with new instructions, and the computer responds again, etc. 'Interaction' refers not only to the speed or rate of exchange between user and machine but also to the extent of dependency of computer reply on user input and vice versa.

'Interactive' has been used as a label for certain types of programming languages and problem-oriented systems

which have special features to facilitate 'conversation' between user and machine. When the lesson (or system) designer knows the student (or other user) will be there to answer questions when the procedure doesn't know what to do next, or to interrupt when it begins typing out more than he finds necessary, the designer can use this to considerable advantage in automating only parts of the process and leaving the remainder to human interpretation and judgement.

Interaction may also imply 'dynamics,' e.g., changes in parameters and substance to 'generate' new content or rules or procedures for the student presently being aided, but in a fashion not anticipated in detail by the lesson designer. If full prescription could have been given in advance, delivery could have been made without a generalized information processing device, i.e., with some less expensive device such as a slide projector or book!

2.2.5 Role of the computer for the individual serviced

The student may see the computer program as a device which does scoring, analyzing, or retrieving for him; in general, those things he would rather not do himself.
Or he may see it as a test imposed on him for an exercise he must complete.
If the student does see the computer as a servant doing his bidding, and a study aid doing what he asks, he might give the results of processing more attention and thereby better use the device to further his learning objectives.

The teacher, on the other hand, may see the computer system as a proctor of required exercises and a validator of performance in testing situations.
Of course, the individual students still must be proctored by some teacher or student aide if the system is to protect against one student sitting in for another, or using inappropriate aids of whatever kind (books, notes, another computer, etc.). Someday machines may automatically check fingerprints or 'voiceprints,' and various written and procedural aids will be assumed available as part of the performance task.

Human attitudes toward the machine cannot be as predictable or as easy to determine as perhaps the degree of program control or amount of diagnosis and prescription. The role of the computer will be viewed from as many different perspectives as there are people using it, and the user's view of the computer will interact in varying degrees with the other dimensions.

2.2.6 'Naturalness' of the communications between learner and system

At one end of a dimension of natural to formal (or pre-coded) language, the computer is programmed so that the student may respond with terms and conventions natural to him, and little if anything about system operation need be learned to use it. For language-oriented topics the computer program must be capable of performing complicated textual analysis in order to conduct a dialogue with the student. The computer's ability to process natural language, and subsequently to diagnose and prescribe on the basis of statements made by the student without any constraints, approaches the realm of 'artificial intelligence' and presently is very much in a theoretical and experimental stage of development.

At the other extreme the user must learn how to give messages to the system to recognize what has been interpreted successfully and when to call on human aid if the system does not respond understandably. This is no hardship for topics of study in which the conventions of computer use are consistent with the objectives of instruction. Formality of communication is desirable in the sciences and engineering where the computer and its programming conventions provide appropriate constraints on theorizing and model building.

2.2.7 Summary

These six dimensions can be viewed as defining a space or domain of computer use, and the modes usually mentioned as simple categories (drill, tutorial, dialogue, socratic, simulation, learner-directed etc.) are more appropriately described as filling some part of this domain. The documents of project CLUE have used this conceptualization in a tentative way to establish a broader prespective on computer use, and to open up new possibilities for computer service to learners.

3. BRIEF HISTORY OF THE PROJECT

A framework for discussion and review was extracted from primary sources during the early months of Project CLUE (January through March, 1969). Staff members searched for current and relevant materials in order to interpret what was being done, what had been planned, and what was judged to be most needed and important. Project directors and persons active in research and development throughout North America responded to requests for pertinent materials relating to their current work. Staff members visited sites of particular interest and attended meetings and conferences.

Drafts of five statements of points of view were assembled during the summer and mailed to all persons quoted directly and to other interested and knowledgeable specialists in the field. Revisions were made by CLUE staff during the fall and winter, and a preliminary draft of the final report was distributed for review and comment in February, 1970.

Throughout the project, CLUE staff members coordinated their efforts with those of other projects reviewing the use of computers in education, among them: a study of the computer in higher education at Rand Corporation; a survey of computer uses in college teaching at MIT; program planning by the National Council for Educational Technology, London; and numerous single conferences and symposia.

4. HIGHLIGHTS FROM STATEMENTS OF POINTS OF VIEW

Volume II of the CLUE report consists of chapters on five factors in computer use. Each chapter juxtaposes the differing opinions and predictions of experts. A number of concise and sometimes controversial statements have been selected for printing here to provide a summary of tentative findings, although baised by selection, and to provoke discussions at the Conference.

4.1 *Operating procedures and costs*

A tremendous discrepancy exists between the ultimate promise and the immediate possibility of the use of the computer in instruction. Schools must drastically reorganize and be appropriately funded to absorb the necessary

reform. Planning must anticipate obstacles which arise when innovative changes occur.

Administrators rather than technologists must lead the way in preparing for the new educational technology, and preparation must begin at the lowest „management" level, that is, in the classroom. A critical shortage of qualified personnel can be remedied by efficient retraining programs which should in particular promote a new role for the teacher as manager of instructional processes and tools.

In the funding of new technological systems, institutions will not go beyond instructional budgets for conventional instruction unless federal or private funds are designated specifically for instructional systems. Publishers and other private investors should induce initial demonstrations and continuing diffusion effort, perhaps through national commissions.

Although progress in the computer software field will reduce the cost of producing instructional programs, progress in developing instructional strategies may result in more sophisticated, responsive, and individualized lesson material and, therefore, more cost to the user.

Reported costs per curriculum hour developed cannot be generalized since the figures depend on goals, manner of computer use, appropriateness of the system, previous curriculum development in the area, and the accounting method used.

4.2 *Strategies for developing effective uses of computers in instruction*

Ultimately the value system of the educational community determines the worth of the various instructional technologies and methods hopefully with reference to actual learning and attitudinal outcomes.

The use of a computer as a learning tool is ignored too often even though it has more immediate potential than any other method of use. Critics of automated tutors ask: Should the computer control the child or the child the computer? Probably the managers of future computer-based learning resources will make the primary sources of knowledge directly available to students through organized files of information and procedures.

Explicit models are needed for instructional design. One of the major problems faced by those developing instructional strategy and content is the lack of prescriptions for design. Nevertheless, some professors appear to be more productive without the constraints of an explicit model for instructional design.

Hierarchies of knowledge and learning tasks provide one foundation for the design of individualization and its description in flow-chart form. The hierarchial analysis procedure seems readily applicable to any cumulative subject matter (such as math or science) but less applicable in highly verbal areas.

Documentation, that is, readable descriptions of content and strategy, are essential to economical maintenance and effective use of instructional programs. Widely adopted guidelines for documentation will broaden the base of users of effective instructional programs.

In the future, authors of substantial curriculum packages will not be able to afford the luxury of individually shaping or tailoring each line of text in each frame for each kind of student as has been true of nearly all tutorial applications to date. Therefore, the first successful development on a large scale will probably be for well-structured subjects such as elementary foreign language which can be taught by stylized procedures.

Computer science eventually will provide a kind of „artificial intelligence" to be applied in the teaching-learning process in all disciplines.

4.3 *Programming languages and instructional strategies*

No one language can be right for all. Each user has a different background, goal and level of access to the system to be considered: student, instructor, lesson designer, researcher, etc. Each mode of computer use has special requirements, and different styles of program preparation benefit from different language features, from simple formats for describing curriculum material to be presented by stylized procedures, to system features with which to delimit a problem-solving environment.

A language and its implementation shape the strategy of instructional uses to a great extent. It is better to have real needs of the learning environment determine the procedures or information processing used. Some considerations are: data available; processing capability; adaptability to specific tasks; generality of procedures; and manipulation of files.

Universality cannot be established by a group organized to set standards. Nor will any small number of languages (as justified by different requirements) be standardized effectively and widely adopted. Translation among languages of similar purpose is possible (and economic for some curriculum designs). Equipment differences, especially in the terminals (learner stations), cause most of the problems.

Documentation (according to „publication" guidelines) facilitates preparation and revision of instructional programs, review by referees and potential users, and communication to programmers who are to implement instructional procedures and content in some other system and language. Natural language is not the ideal mode for describing learning exercises; some formalization of language appropriately requires the user to reflect on what he instructs the machine to do.

System features impose definite limitations on language (and strategy). Hardware items, such as clocks, interrupt control, memory protection, and terminal controllers, limit what actions an author's procedure might take. Software for handling files, editing text, linking precessors, and connecting simultaneous users of the system are important factors for some uses. Communications capacity, measured in bandwidth, elapsed time and charges, sets bounds on conversation rates especially when diagrams and other graphic representations of knowledge are involved. Flexibility is important in language design to allow adaptation to new ideas and opportunities. Economics is a primary consideration for a processor (or its compiled program) which will be operated for many students. Modularity is an essential part of the design of systems (language processors) which are to be modified later or translated to another operating system. Documentation of any useable system (language) is essential for program maintenance as well as for user reference.

Interactive mode of computer use should be justified by some essential contribution to learner and author. The program's reply should be immediate and responsive, and usually something more than „wrong, try again".

A curriculum development group should remain free of the constraints of specific languages even to the extent of leaving the computer when the available software and

programmers cannot implement the important features of an instructional strategy. A software system should be adapted to specific uses and users; the managers should maintain clear subservience of language maintenance and system operation to project goals.

4.4 *Computer technology for instruction and research on instruction*

Benefits for research on learning and teaching are relatively greater than benefits for operational instruction today, but some warnings are in order. Many research studies do not require the sophisticated equipment of computer-based systems. Many computer-based systems do not provide essential data even for rather simple studies. A learning exercise, whether for research or for operational instruction, should not be computerized simply because the machine would otherwise stand idle.

Contributions of CAI to the development of a theory of instruction (and vice versa) have been questioned. Learning theories are not relevant to school learning and instruction theories have only begun to be developed. Specific theories of CAI distract the developer-researcher from the range of instructional resources available.

Today's educational problems will not be solved by introduction of CAI on a massive scale. Some small number of functions computer do better than other media (including humans). However, the financial expenditures and complexity of management required by computer use today are not necessary to teach most students most subjects. Alternative media and techniques should always be examined.

Guidelines for developing instructional systems to respond to individual differences in an efficient manner have not yet been developed. Many computer-related instruction programs only give the illusion of individual tailoring.

Research should be conducted to better the understanding of student learning and self instruction in semi-automated environments. Administrators must look to less expensive and more flexible facilities both for conducting the research and for administering instructional materials.

4.5 *Documentation and distribution of information, learning materials, and teaching resources*

The role of publisher and manufacturer is not decided. The trust and cooperation necessary between educators and manufacturers is difficult to achieve since the educator's open environment and tradition of free exchange of data are contrasted with the manufacturer's concern for protecting proprietary information and/or not releasing defective products too early.

Distribution of new materials and research findings must be improved. Authors are not provided sufficient incentive to document, distribute, and revise computer-based exercises. Potential users cannot choose wisely without more complete information, reviews, and evaluative studies.

Guidelines for broadening the base of users of effective learning exercises have been suggested. First, formalize and standardize procedures of journal review and dissemination; issue a periodic review of computers in education. Second, provide better incentives to the author: academic credit through professional review; copyright protection; and reimbursement by the institution and/or by user fees. A third guideline concerns improved documentation of learning exercises. Authors are advised to: describe essential information with a standardized notation for the purpose; derive a description from records of interaction between „typical" students and the computer program; and perhaps incorporate „annotations" for the reviewer within the interactive program itself available for examination on his request.

Curriculum development resources and training programs might be centralized in order to diffuse common practices and increase likelihood of exchange. Competing curriculum packages will be distributed through different time-sharing services, and decisions based on quality and economics will be made in the marketplace.

Cooperation between the universities and industry in the production of programs is almost essential. Of course, the major responsibility for what is taught will continue to be with the publisher and author rather than with hardware manufacturers.

5. SUMMARY

This paper has described the framework for a critical review of computer uses in instruction, and put forth some tentative findings, but it is not possible to summarize the conclusions and recommendations at this time because the study is still in progress. Nevertheless, it does appear from the work to date that one general operating principle for persons planning projects in this area is to select carefully among possible applications of computers those which best exploit the computer role and to match computer contributions to the real needs of learners. Furthermore, the development of information processing tools for learning can better be pursued with a subject or discipline orientation than within the special area of instructional technology.

Information processing and computers will be used widely in all subject areas, and these new tools for learning and performance will interact in an exciting way with new goals and means for education.

COMPUTERS AS A TOOL FOR RESEARCH INTO THE CONTENT OF EDUCATION

H. Bestougeff, M. De Vos, and A Valette.

1. INTRODUCTION

The laboratory of the O.P.E. was established for the purpose of elaborating experiences and checking knowledge in the field of electricity with students at Bachelor's level. This first experience led us to install, using an IBM System 360 model 30, a system which would allow teachers who are not informatics experts to define a pedagogical strategy at an adequate level of presentation and for chaining of control and educational items. In practice we have thus created, on the one hand, a timesharing system with twenty teletypes and, on the other, a source language which is sufficiently simple and general to be used possibly for other disciplines.

At present, various groups from disciplines which are as far apart as mathematics, physics, biology, and English language, are using the system for three different applications.

Students of mathematics are able to learn how to write numerical algorithms through use of a conversational programming language 'BASIC'. For the other disciplines, a tutorial mode selected from a documentation system results in two types of questionnaires being drawn up:

- checking of knowledge on the basis of an external course; examples taken from biology
- education: presentation of a training course through a ramified questionnaire (type Crowder), illustrated via a course in English linguistics.

In our two examples we have selected for presentation the two modes which are, in fact, mixed to the extent that the test questionnaires constitute part of the educational questionnaires controlling comprehension as the presentation of various items progresses.

2. CONCISE DESCRIPTION OF THE SOURCE LANGUAGE

With both types of questionnaire the mode QCM has been entirely reserved for free response. Analysis by character, being too restricted, has been dropped. It has been replaced by analysis using keywords; characteristics being as follows:

each item comprises:

- one question block of 700 characters
- one answer block of 700 characters which contains up to 10 answers – type (RT), each one being associated with a specific comment.

Each RT consists of a certain number of keywords. Students' answers coincide with this RT if they contain exactly the same keywords, taking into account other words which may be added to the answer within the limits of the 56 characters assigned to the student. This allows for a wide variety in the phrasing of the answers.

Each foreseen answer is associated with two types of comment: comment true (CT) and comment false (CF); alternatively, one may also ask for coincidence in order to disorder, and foresee various equivalent answers at the level of an identical RT, using the functions OR and AND. Furthermore, negations are systematically examined. All negations which were not foreseen by the author cancel the student's answer, even if all keywords correspond. Specific characters have been recorded as word separators. The list may vary with the users' needs. Finally, an automatic loop testing system prevents the student from taking the same way twice.

3. TEST OF KNOWLEDGE OF BIOLOGY

3.1 *Principal characteristics of questionnaires:*

The questionnaires involved are those for self-checking by students of the P.C.E.M. who are also following courses and making both specific and practical studies. Questionnaires are compiled by a group of six biologists. Presently, two questionnaires are being used: the first, short and linear, is a simple test of knowledge; the second includes 31 questions, 4 of which are ramified. Its content is an analysis of the course of the cycle in the animal world; the questions relate to the training course, during interrogation; however, the essential concepts are brought up either through foreseen answers from students, or in the text of the questions or comments, these concepts being: mating, homologous chromosomes, chromatic reduction, chiasm, crossing over, autosomes and heterochromosomes, etc..

It is completed with the concept of re-establishment of the diploid state due to fertilisation, as well as the importance of the mixing of the genes. Due to this fact, and to the fact that this is a simple check, the procedure is also a process of revision for the student. This aspect is enhanced through the possibility of appealing to the documentation.

At all stages throughout the interrogation, students may ask definitions of terms or expressions which were previously recorded in storage by the teacher. This, therefore, is a test of comprehension and reasoning rather than one of memory, since it does not ask for an automatic restitution of the course.

3.2 *Examples*

By way of illustrations we give some abstracts of listings and student – machine dialogue.

The results obtained from examinations taken by several hundreds of students are now the subject of statistical studies, both quantitatively and qualitatively. Furthermore, at the level of the analysis of contents, the

biology group was obliged to reconsider the structure of education itself, thanks especially to the feedback from the students.

4. APPLICATIONS IN TEACHING ENGLISH LANGUAGE AT THE INSTITUT D'ANGLAIS CHARLES V (Faculty of literature, Sorbonne, Paris).

4.1 *General objectives*

To date, 5 questionnaires have been edited by a group consisting of two teachers and three advanced students (Master's level). The students are entirely engaged in the conceptual work, since, having themselves already received the relevant education, they are better qualified than the teachers to foresee reactions from the students on whom the experiments have taken place. It should be emphasized that this co-operation has provided excellent results. The 'guinea-pig' students are volunteers selected from first-year students, and also fourth and fifth year students who are planning to teach.

4.2 *Choice of linguistics*

Among the subjects called 'literary', linguistics seems to lend itself best to programming. A theoretical reflection on the English language, rather than practical training in the language itself was chosen, since the dialogue with the machine is performed only in writing, contrary to the principle of teaching modern languages.

In fact, we have noticed that computer-assisted instruction should be included in a three-stage system: practical jobs on the computer, practical jobs in the language laboratory, and specifically oriented jobs allowing for discussion and intensification.

4.3 *Example of the possibilities of such a system as is now in practice at the Institut d'Anglais Charles V.*

Students at O.P.E. receive questionnaire Q12, Q13, relating to the breakdown of the verbal group in English. The central role of the auxiliary, simple or modal, is shown, as well as the following concepts: pausing aspect, intonation and modality, through the breakdown of a simple sentence such as: 'the delegates might have been chosen'. The role of the 'locomotive' of the verbal group is assigned to the lead auxiliary which leads us to study the application of the auxiliary as a leader of the verbal group: tag questions, resumption during dialogue, open or closed answers, etc..

When students have been induced to a reflection of the **theoretical** type, the road has been prepared for practical applications which will be carried out naturally at the language laboratory. Thus, emphasis can be placed on intonation, which is impossible with C.A.I..

Finally, the specific studies are used for compiling files of examples chosen during the lectures and their comments. In order to strengthen his experience, the student who is awarded less than 15 out of 20 marks at the final test is invited to perform a series of supplementary drills, both written and oral.

4.4 *Analysis and structure of material*

The object is education and not control. The course is not dependent on any previous knowledge. However, students, all English language experts, are supposed to have mastered the language well. Performance is not observed; therefore, questions are simple, and progress is rather slow. The aim is not to trap the student, but to induce him to ponder the language. Information is presented in a concise and abstract way, relating to distinct language elements. The student is thus led to show linguistic systems as e.g., time, aspect, number, etc., and next to indicate the relationship between the various systems. One sees that the language is presented as an autonomous system by separating the linguistic levels (the apparent language elements) from the extra-linguistic levels (reality). The student is frequently invited to stop to think and carefully read the questions. In certain cases the student can actually find the answer simply because he can understand English, but then, he may be tempted to take a short-cut through the reflection path, which naturally has a bad influence on the remainder of the questionnaire. One of our major concerns is to convince the students, conditioned by years of education based on the checking of performances, that they should proceed at their own pace, and to take their time for it.

4.5 *Example*

We are now showing abstracts in the form of listings of a questionnaire on numbers. (Fig. 4)

The automatic loop test, created in order to facilitate the teachers' task seems, in some cases, to be a limitation for the author who is to construct complex and superfluous ramifications, during course writing, if he wants students to take the same way twice.

It appears, therefore, to be necessary to have two types of language: the first, a very simple one, leaving the author little freedom but allowing him to become rapidly familiar with the system; the second is undoubtedly less easy to handle by ADP non-experts, since the various steps should be specified by the author, but have the advantage of being flexible.

4.6 *First results*

The machine produces surveys of students' results containing statistical results of each session. These results are elaborated with the co-operation of psychologists. With respect to linguistics, it should be stated that the small number of students tested and their high level of motivation does not allow us at this stage to make definitive conclusions as to the value of the system from the viewpoint of acquisition of knowledge.

The essential profit obtained from this experience, until now, lies at the level of material analysis for teachers and, in this field, research is only starting. But already the demands of programming have influenced the approach of many teachers, making them familiar with pedagogical group work and, further, with applying more scientific rigour in a field where ambiguity of terms and the course of reasoning are commonplace. Also the 'test' students are invited to express their criticism and appreciation with respect to the questionnaires which makes them play a more active part, transforming the pedagogical relationship by allowing the student to control the education he receives.

5. CONCLUSION

All experiences obtained until now at the O.P.E. are enhanced by the importance of basic research at the secondary educational level rather than of mass exploitation of a CAI system. We feel, in fact, that it would be

premature to make conclusions from the first results obtained, based exclusively on relative performances. The essential problem is still that the results cannot be completely analyzed in terms of the actual knowledge of learning by experience.

In this respect it is essential that computers be considered as simple tools and that the languages used be adapted to the requirements of education.

Substantative modifications, requested by users, are being made, developing the tutorial mode into an interrogatory mode; increasing the quantity of information exchanged between student and machine; developing the documentation, experiment simulation.

Fig. 1

14 K19 QUESTION 10

The 46 chromosomes of the human species include 44 autosomes (22 pairs) and 2 heterochromosomes, or sex chromosomes.
The general chromosome formula of a diploid cell is 2N = 2A + 2 heterochromosomes.

The two heterochromosomes of the woman are X and X, the formula, therefore, is 2N = 2 x 22 + XX.
In the case of the man, the heterochromosomes being X and Y, the formula is 2N = 2 x 22 + XY.

Then, what are the chromosome formulae for human spermatozoids?

R10J+

K19 ANSWER 10

1 The formulae are 22 + X and 22 + XY.
CJD Yes
Q11

2 22+X//22X//22A+X//N=22+X// N=22X//N=22A+X
22+Y//22Y//22A+Y//Y//+Y//N=22+Y//N=22Y//N=22A+Y

CJD Right
Q11

3 22//22A//22+//22A+//N=22A//N=22+//N=22A+ X//+X
Y//+Y//22+Y//22Y//22A+Y//N=22+Y//N=22Y//N=22A+Y
CJD Right.
Q11

4
ARF+Carefully read question again, and give both formulae exactly and completely.
Q10

Fig. 2

The 46 chromosomes of the human species include 44 autosomes (22 pairs) and 2 heterochromosomes, or sex chromosomes. The general chromosome formula of a diploid cell is 2N = 2A + 2 heterochromosomes.

The two heterochromosomes of the woman are X and X, the formula, therefore, is 2N = 2 x 22 + XX.
In the case of the man, the heterochromosomes being X and Y, the formula is 2N = 2 x 22 + XY.

Then, what are the chromosome formulae of the human spermatozoids.
22+ X

Carefully read question again, and give both formulae exactly and completely.

22 + XX
22 AUTOSOMES + Y and 22 + X
Right.

All ovulae have, of course, the same formula 22 + X. According to the previous data, how many tetrads will form, during the forephase I, in the case of the woman?

TWENTY-THREE
Right.

In a diploid organism there exist N chromosomes of paternal origin and N of maternal origin.
During the anaphase I. do all N chromosomes of the same origin move to the same pole?
THIS IS RATHER IMPROBABLE
Indeed.

Fig. 3

9 – With a female, how are the diploid cells called, which are ready to undergo the monthly cycle?
OOGONIES
The monthly cycle is not performed in the gonies. Answer again.
PRIMARY OVOCYTES.
Yes, ovocytes I, or of the first order.

10 – Same question, for the male.
*DCM:SPERMATOCYTES
Male germinal cells in which the monthly cycle is performed.
*DCM SPERMATOCYTES I
Spermatocytes of the first order (2N Chromosomes).
THESE ARE THE SPERMATOCYTES I
Yes

In a general way the ovocytes of the first order and the permatocytes of the first order are regrouped under the term cytes of the first order.

11 – With a female, how does one call the cells at the end of the monthly cycle?
EGGS
No, that is a language error. In principle, an egg is a diploid cell. Answer more exactly.
OVULAE
Yes, ovulae or ovotids.

12 – Same question, for the male.
SPERMATOZOIDS
No, this denomination is premature. The monthly cycle does not directly produce spermatozoids. You have omitted one phase (sperm being produced at the end of the monthly cycle). Answer again.
*DCM SPERMATID
Haploid male cell at the end of the monthly cycle.
THEREFORE THESE ARE SPERMATIDS.
Very good.

Fig. 4 ABSTRACT OF K24-26

Q20 – The zero article may be replaced before 'books' by the following adjectives: the, these, those, some, any, no, few, many.
Now reduce this list to those which only preceed the enumerative plural.
R20 – Inexact
CJD *Q21

RT – the//some//any//no//my
CFD – state only those adjectives which occur only before the enum. plural. Answer *Q20 again.
RT – these//those//few//many
CJD – Good *Q21
RT – these//those//few//many
CFD – incomplete answer. Answer *Q20 again.
ARF – Verify your typed answer, and answer *Q20 again.

Fig. 1 ABSTRACT OF LISTING OF QUEST. K19 (question block and answer block 10).

The usage of the functions **'OR'** and **'AND'** permits us to foresee various representation of one and the same answer.

Each **character set**, in which characters are not separated by spaces, is globally processed by the machine as one single **'word'** (the 'separator' characters: **comma, period** and **apostrophe** are excepted). Therefore, it is necessary to foresee such individual answer as e.g. **N = 22 + X**, or **N = 22 + X**, or **22A and X, 22 + X**, etc.

In Nr. 3 two **'tabulations'** occur (function **'AND'**); one is the **space** between **N = 22A+** and **X// + X**, the other is the space between this latter group and **Y// + Y//22 + Y//....** The student's answer should, in this case, include one of the terms **22**, or **22A**, or **22+**,.... AND **X** or **+ X**, AND **Y** or **+ Y**, or **22+Y**, etc.

The code **CJD** (D = disorder) here permits us to accept these terms **irrespective of the order** in which they occur in the student's answer.

Fig. 2 Abstracted answer from quest. K19 (questions 10-12). The question 10 is the one presented in the listing shown in fig. 1.

Fig. 3 Abstracted dialogue from quest. K18 (questions 9-12).

*DCM: the student, before answering the question presented to him, requests a definition from the 'dictionary', an automatic documentation system (DCM = documentation). This concise (a maximum of 100 characters) definition may help, the student without providing a complete explanation, to understand the meaning of the question, or to be sure of the exactness of an answer he is planning to give. The dictionary may be

Some typical erroneous answers which are frequently noticed are presented in this abstract: 'ovogonies' or 'oogonies' for question 13, and especially 'spermatozoids' for question 16. Following the comment given by the machine, students answer more exactly in the majority of cases.

Fig. 4 Abstract of listing of quest. K 80-Q 20.

The first RT is the one given at the end of the loop. Normally one puts: 'False, the answer was.....' But, in the present system, after two questions the associated comment is not given anymore. Students do not know where they have gone wrong. It was preferred, therefore, to give them the comment 'inexact' and to have them return to the normal sequence (Q 21). Q 21 starts with the comment on the answer to Q 20 which is, therefore, received by all students. For those who had given the right answer, this serves as an enhancement, limiting the risk of haphazard answering. In a general way, even if there is no loop, each comment which may prove of interest to all students, irrespective of their answer, should be given at the beginning of the next question in the normal sequence, hence the economy of characters in the blocks.

RT2 The machine compares the answers in the order; one therefore places everything one wishes to eliminate in front. In this case, all terms of the list to be excepted.
RT3 Key-words to be found, separated by tabulations.
RT4 Thanks to the function OR one can foresee the situation where the answer, without being false, is incomplete.
ARF In this case, the false elements having been removed in RT2, one supposes that no typing or writing error is involved. Students hardly appreciate getting the comment 'false' if they have just put down two words.

Fig. 5 EXAMPLE OF STUDENT-MACHINE DIALOGUE K24-26 (question block and answer block 6).

The answer to the question Justify the plural in "my experimental group are learning fast" is foreseen in the listing, as follows:

QO6 The word 'public' refers to the plural.
There may be two types of forms:
'The public don't understand...' which reflects the extra-linguistic reality, and
'The public doesn't understand...' which is strictly a grammatical form.
Justify the plural in
'My experimental group are learning fast'.
A group does not consist of one student only, hence the plural.
Good
Mind you, the single form is also possible.

We have noticed that the single/plural form partly covered the contradistinction unique/multiple.

RO6 MULT//MULTIPLE//EXTRA LING//EXTRA-LING//EXTRALINGUISTIC//EXTRA LINGUISTIC//SEVERAL//SEV//MANY
CJD Right *QO8
RT NO//NOT GRAM//GRAMMATICAL//LING//LINGUISTIC//UN//UNIQUE
CJO Right *QO8
ARF Here is the answer: group refers to the multiple *QO7.
QO7 Either you have made a typing mistake, in that case type, tm, or you haven't understood, in that case type hu.
RO7 tm
CJO Sorry *QO8
RT hu
CFO Group indicates several students, hence the plural form *QO8
ARF What have you been making up now... Let's be serious. *QO8.

SIMULATION IN THE CLASSROOM

William Russell Broderick

1. INTRODUCTION

There are many roles the computer can play in the broad area usually referred to as "Computer Aided Instruction" (CAI). The National Council for Educational Technology in the United Kingdom has listed a number of these in their Working Paper No. 1 "Computers for Education". This paper concentrates on one of these, namely, the role of simulation and game playing in the classroom. The author feels that this area has a particularly high potential in the English educational system.

2. STUDENT INVOLVEMENT

Whilst one normally thinks of simulation and gaming in conjunction with the teaching of sciences, we must not limit our thinking to this curriculum area: perhaps one of the most significant advances in this field has been the development of educational drama in the UK over the past 20 years. The aims of educational drama have been to give students an understanding of themselves, their personal potential and their potential in society through deep personal involvement in dramatic activity, this deep involvement usually being achieved through exercises in concentration and free improvisation. The effects of this and the students' behaviour are quite remarkable and have to be seen to be believed.

The Nuffield Science Projects in the United Kingdom are also geared to student involvement in the study of science and have moved away from the more formal methods of science teaching which were prevalent in the 1950's. The aims of the Nuffield Science Projects are to give students a better understanding and appreciation of scientific method; they are encouraged to conduct experiments themselves, to formulate their own hypotheses to devise tests for these hypotheses, and as a result of carrying out these tests, modify the hypotheses in their search for scientific truth. In this work one sees emphasized the importance of the students using discovery methods and their personal involvement with investigating a situation.

3. CONSULTATION WITH TEACHERS

An awareness of the above work caused the author to discuss with teachers the significant contributions they felt a computer could play in teaching. As a result of these discussions, the broadest area of agreement was found to be in the areas of CAI normally referred to as "Computer Managed Instruction" and as "Simulation and Game Playing".

There is little doubt that television simulations of the Apollo 11 moon landing did a great deal to open the eyes of the teacher and the general public to the potential of this technique. Furthermore, more locally, The Royal Liberty School takes part in a national competition organised jointly by International Computers Limited and the Institute of Chartered Accountants in England and Wales whereby a group of students forms a board of directors of a company and plays a business game in a competitive environment. At this stage it is appropriate to define the term "simulation" and I define it as follows:

"The creation of a realistic model of a situation which the student may study, interact with, or control, to provide him with a lifelike problem solving experience".

This model however does not have to be computer based, indeed one of the most imaginative computer science lessons the author has watched was with a group of nine year olds who were acting the roles of the various elements of a computer's central processor and input/output systems and it was quite evident that these students were really learning about the fundamentals of computer architecture.

In devising any simulation for educational work, be it computer based or not, it is important to define the educational objectives to be attained. Furthermore, should these educational objectives not be properly defined,then the use of the simulation becomes very restricted and follow-up work after the activity tends to become unrelated to the students' learning and behaviour during the game.

In our study of simulation and gaming with teachers, the teachers themselves emphasized the importance of simulation in providing, within the classroom, models which could not otherwise be achieved and studied: two of these are mentioned below (See Section 5).

4. A SIMULATION IN GEOLOGY

It was also discovered that some teachers had been developing simulations of their own, and a report by Keith Lovatt, at the time the Head of the Geology Department at The Royal Liberty School and now on the research staff of The Royal Liberty School Computer Department, is given below for it shows how ideas for simulation are developed by teachers and illustrates the limitations placed upon both the teacher and the class because a computer system was not available for this work.

4.1 *Simulation of Geological Prospecting*

"The student of geology in schools has little opportunity to apply his knowledge of geological principles and techniques in "real" situations because of the restrictions of time and money. This state of affairs is particularly true in the field of geological mapwork and the student is prevented from enjoying the excitement and intellectual

stimulation of original discovery and interpretation which is the basis of geological mapping.

In an attempt to remedy some of these deficiences, it was decided to devise a prospecting simulation, and,to add spice to the activities,it was constructed as a competitive game. Basically, the rules of the game are simple. Each student is given a topographical map of what is described as a geologically unexplored area. However, it is suspected that huge quantities of oil could be present and so a number of commercial companies deem it worthwhile to send in geological teams to map the area and search for suitable oil traps. Each student acts as the chief geologist responsible for planning and co-ordinating the mapwork for a company.

The original map has a grid overlay and the student can obtain geological information from any grid intersection. This information is the usual information which any geologist might record in a surface investigation and the student plots the information on his topographical map. Each student takes his turn to receive geological information and is limited for each turn to the information from five intersections. The five intersections must form a set – being either a block or string of adjoining intersections.
As the game proceeds the students build up geological maps of the area and construct sections to help in the interpretation.

When the student thinks that he has enough information he stakes a claim for a particular group of grid squares.
These claims are published to the rest of the competitors who are not allowed to claim or even prospect in these areas.

A scoring system based on the oil traps discovered and the amount of time taken to make the discovery completes the competitive aspect of the game.

Ideally a computer should be used to store all the geological information about the area. In a trial run of the game the teacher acted as the store and although this slowed the game down considerably the students still enjoyed the exercise and gained a sound understanding of some of the joys and difficulties of geological mapping.

In the future it is planned to include geological specimens as part of the information available at each grid intersection which should increase the interpretation element considerably."

5. SOME SIMULATIONS BEING INVESTIGATED

The science teachers in particular emphasized the importance of interaction between the students and the simulation and stated that without interaction the use of computer based games would be a virtually indefensible use of expensive resources. However, they saw a considerable potential for creating mathematical models of physical situations on a once only basis and recording these on to video tape and using them during their normal teaching to provide a dynamic model of the behaviour of the situation under consideration.
The teachers also emphasized the importance of interaction when related to students' involvement in the activity being simulated. They felt most strongly that this approach is likely to have the maximum educational benefit.

It will be beneficial here to describe briefly two games of a competitive nature and two simulations:

5.1 *Business Game*

In the business game which is played in The Royal Liberty School, and for which there is a Fortran program on our Marconi-Elliot 905 Computer, a model is created of three competing companies marketing a domestic appliance in four sales areas. Each company is represented by a board of directors formed from amongst the students themselves. The companies each have a sales advantage in their home sales area and compete on equal terms in the fourth area.
The boards of directors are then presented with the initial situation of the game and must decide how they are to plan their use of capital so as to maximise their profit.
Amongst the decisions they have to make are the financial allocations required for transport costs, advertising, administration, purchasing of additional plant to increase production capacity and of course they must decide pricing policies in the four sales areas. It will be seen that the board has to make an inter-disciplinary approach to this work and when it has made its move details of all three boards' decisions are fed into the computer system which produces, as output, a report of the state of the game after that period of play: the periods are normally reckoned to be of three months duration. It has been our experience that during the first session in which the students use the game, they are inclined to make judgements based on guess-work but gradually they tend, of their own volition, to recognise the importance of basing their judgements on facts and they then start to apply more scientific methods to the study of strategies for the game.

5.2 *The Investment Game*

An investment game has been specified (though not yet implemented) to help in the teaching of mathematics where a considerable amount of attention is paid in some schools to the study of stocks and shares. Alas! In the classroom this often becomes yet another unrealistic application of the unitary method of proportions. The purpose of the game is to create a more realistic environment along the lines of a business game where the student will in fact manage a fairly large investment fund and will plan the investment of this money in a wide range of securities.
The objectives here are not only to generate a stimulus in the teaching of this aspect of mathematics in the curriculum and to provide motivation for the students to perform the mathematical analysis involved, but also to integrate this work with economics taught and thus bring more realism to the subject.

5.3 *Genetic Simulation*

In Biology considerable emphasis is given today to the teaching of genetics. Yet because of the time factors involved, it is impossible to study this subject practically in a laboratory and the teaching tends to be descriptive rather than heuristic. At the request of the Biology Department at The Royal Liberty School, a simulation model of genetics is being developed and this is to be used interactively by the students as if they were conducting a scientific experiment. It is anticipated that the students will be able to put into the simulation various configurations of dominant and recessive genes and through studying the behaviour of a simulation with respect to an accelerated time scale, they will be able to study the changing behaviour of the population and also to interact with the simulation at given time intervals and change input parameters, this way discovering the effects of introducing different genetic characteristics into the population. It is part of the design specification to make a model completely general and to

allow the teacher to specify the biological area to be studied.

5.4 *Chemical Reactions*

It has been strongly represented to us that a lot of the students' basic misunderstandings in chemistry come from not really appreciating what happens at the molecular and atomic level in a chemical reaction. It is possible to view an actual chemical reaction at the molecular level and therefore one is forced to consider the possible role of simulation in teaching this. This model has not yet been specified and the objectives have not yet been defined by the chemists. In their discussions the teachers outlined the aim as being to present a dynamic model of a number of molecules interacting. It will be necessary to allow the student to change the parameters of this model so that he can study at first hand the effects of the presence of too many or too few molecules. The model should also enable the student to study and understand why some molecules react and others do not.

The above four examples are selected from a number which are being analysed for future development and have been chosen to illustrate curriculum areas in which simulation and gaming can play a part in bringing into the classroom models of real life situations which cannot be studied in a practical context in any other way. It is important in the development of these models to ensure that their behaviour can be replicated and to ensure that they are so designed as to be capable of being integrated with either normal teaching techniques or in the future with other forms of CAI.

6. SUMMARY

In case too glowing a picture of simulation has been painted, it is appropriate to list some of the advantages and disadvantages of simulation.

6.1 *Advantages*

6.1.1 Simulation may provide experience in a wider range of educational objectives.

6.1.2 Simulation provides the learner with a sense of immediacy and involvement.

6.1.3 There may be a greater transfer between the learning situation and real life although this has not yet been proved.

6.1.4 Simulation can provide experience at a lower cost than can be possible in real life (in areas where real life costs are exorbitantly high).

6.1.5 Simulation can speed up a lengthy process.

6.1.6 Simulation can provide an ideal situation for practice of good scientific method.

6.2 *Disadvantages*

6.2.1 It is difficult even with sophisticated visual display units to create real life situations.

6.2.2 To develop simulations is expensive.

6.2.3 Validation of a simulation model may be difficult.

6.2.4 The study of the effectiveness of the simulation in the classroom is usually difficult to conduct in a scientific manner.

There is little doubt that much research must and will be carried out in this field to discover how best simulation can be used in the classroom. The author is convinced that such expenditure on research will be expenditure in a field with a great educational potential.

COMPUTER EDUCATION FROM A GENERAL PURPOSE DECISION MAKING ENVIRONMENT[1]/

James B. Cowie

1. INTRODUCTION

The past twelve years have seen the development of many management decision making exercises[2]/ (alias 'games') which allow business students to 'practice' decision making by organizing themselves in teams to make decisions in a simulated competitive environment. Some games are based on particular industries, others on special situations. They differ in complexity and level of decision making. An extensive amount has been written on the subject,[3]/ including advice on how to organize such activities and about the potential benefits and costs of using them. In most cases, the designers and users of these exercises seem to have been primarily interested in providing better insight into conventional business topics such as organizing for, planning and implementing decisions. A computer is often used for fast processing of the decision inputs but it is incidental to the interests of the designers or faculty administrators.

This paper describes the design of a general purpose desision making environment where the role of the computer is more crucial. Greater use is made of the capabilities of the computer for the benefit of the administrators and the players. The level of detail of the data is chosen to provide an acceptable data processing case study with the associated problems of file organization, processing and control. It is difficult to give students experience of typical business data processing applications. In this model the students see their decisions processed against master files over successive periods. This involvement generates interest in the underlying procedures. The enviroment also provides a convenient setting for modest data processing projects which should add to its future utility.

In summary, the distinctive claims for this effort are the emphasis on the effective use of the computer to support a flexible design structure and the attempt to provide insight into typical business data processing opportunities and problems.

2. BRIEF DESCRIPTION OF THE GAME

Fig. 1 shows the decision areas which are included in the model. For reasons of space and to allow emphasis on the data processing aspects, this description will not include a detailed statement of the rationale behind each decision area, the decisions which are permissable or the formulae by which they are processed. This section is intended to provide only a general understanding of the features of the game. In simplest terms, teams produce a single product and compete to market it in several districts. However, that sixteen different versions of the game have been developed. For any series of plays, the administrator chooses the model number and the corresponding decision areas he wishes included. The other decision areas need be of no concern to him. The following description will emphasize the fullest version of the model.

In the full model, a maximum of six teams operate in up to fourteen sales districts. Each team takes over a company which has been established for two years. Files exist to describe the current resources of each team and the current values of decision variables, some of which are adjustable by the teams and others only by the administrator. Teams have four opportunities per simulated year to change as many or as few values of their current decision variables as they please. These changes are processed by the four primary runs each year. It is acceptable to make no adjustments, but it would normally be unwise to continue such a policy over several quarters. There are also four secondary runs which alternate with the primary processing. These are used exclusively to redistribute unemployed personnel. Those hired can then be assigned in the next major run, which avoids any delay in their utilization.[4]/

Each team has a factory containing a number of work stations where line workers (direct laborers) work on machines under the supervision of foremen. Support labor must be maintained at suitable strengths to prevent a reduction in productivity. Different categories of workers will be recruited from, and lost to the labor pool. Each of the three labor unions will negotiate annually new wage rates and may cause production losses through strikes. Teams may expand production within their factory or in new facilities. This requires them to carefully time phase the acquisition of resources and select from capital budgeting alternatives.

In summary, in a quarter, total production depends on available raw materials, line workers, their foremen, machines and support personnel, all employed in plants large enough to accommodate them. Additional production can be gained from various overtime alternatives and from successful research and materials handling add to costs.

On the marketing side, attractiveness of the product is affected by price, advertising, salesmen, displays and special effects which include research and development. The markets are illustrated in Fig. 3. The consumers in these districts respond differently to the various marketing influences. The total market attractiveness of teams competing in a district generates a number of orders which are distributed among the teams. These orders become sales only if sufficient finished goods are available. Trucks deliver sales to the districts. Operating plans are funded by profits or by various types of capital financing.

The computer outputs from each major decision period

contain results in each decision area, with a summary of all categories in the finance section. This section also contains a cash flow analysis, a profit and loss statement and year-to-date information for the period just completed.

The game is based on a one-year cycle of activity. Raw materials are ordered in quarter N; they are delivered, paid for and used in production in quarter N+1; finished goods are sold in quarter N+2; and revenue is collected in quarter N+3.

3. DESIGN FEATURES

Six aspects of the games' design will be discussed:

3.1 Requirement for decision changes only.
3.2 Repetitive processing.
3.3 The level of complexity chosen by the administrator.
3.4 The dynamic nature of the game.
3.5 The capacity to reorganize the game.
3.6 Open ended design.

3.1 *Requirement for decision changes only*:

The concept that only decision changes need be input is useful not only because it is realistic but also for practical reasons. It avoids the danger of the complexity overwhelming the teams in the early stages and, at any time, allows them to select the decision areas they wish to concentrate on. At various times they may wish to 'rest on their oars', but there is always a wide range of possibilities if they wish to explore them. Students are encouraged to think in terms of critical decision variables which need frequent attention. This leads them naturally into 'management by exception' and consideration of those decisions which can be left to decision rules within a computer algorithm.

3.2 *Repetitive processing.*

One may question the desirability of having as many as fourteen markets (a home market for each team, 4 in the city and 4 in the country). However, this concern seems more based on human distaste for excessive detail than any technical difficulty, for the program can handle a multitude of markets with an insignificant increase in time or space requirements. This repetitive capability is the computers forte. It gives the administrator the advantage of greater flexibility of market structure and the teams more opportunity to carry out market experimentation with less chance of interference from the adjustments of other teams. Note than a team need not, and perhaps cannot, pay close attention to all districts simultaneously.

3.3 *The level of complexity chosen by the administrator.*

The decision blocks shown in Fig. 1 are programmed as a series of subroutines. The main program selects or bypasses subroutines according to the chosen model number which is input as a datum. Sixteen different routes are currently defined. The extreme alternatives are Model 10 described earlier, in which all decision areas are included, and a model where only pricing and advertising policies are relevant. In the latter case, marketing funds for advertising are provided, and the output consists simply of market attractiveness generated in each district by each team. These special inputs of funds and outputs of market attractiveness would not appear in the full model since advertising allocations would be made from available funds and market attractiveness would be converted within the program into orders and sales.

3.4 *The dynamic nature of the game.*

Some games suffer from a lack of incentive for teams who are doing well to change a winning policy while teams who start badly have difficulty in recovering. It is desirable to build flexibility into the environment on some consistent basis so that strong positions will deteriorate if unattended, but teams will not have to deal with haphazard changes. In this game, there are over seventy values in a cost table which can be adjusted by administrator. These costs range from as little as the cost of interviewing support labor to the cost of purchasing a new plant in an expensive district. A team will have more influence on other costs such as: new wage rates which are negotiated periodically with different unions; bids for and offers to sell capital equipment; and salaries offered to present or sought after personnel, etc. To offset rising costs, the market is willing to pay higher prices as time goes by. Increases in costs and prices would be more likely but the administrator can manipulate these variables any way he sees fit. These changing conditions require each team to be alert for adjustments on a continuing basis, partly to meet competitive pressure and partly to exploit further opportunities.

3.5 *The capacity to reorganize the game.*

It is reasonable to expect a considerable amount of informal information flow between students involved in successive uses of the same game. The administrator should counter this, if necessary, by reconfiguring the enviroment before the start of a new game. This capability is illustrated by reference to Figs. 4 and 5.

Fig. 4 shows seven representatives from a family of curves which express market attractiveness (measured in points) as a function of price. Different responses to prices and price changes are evident. The points from price are amalgamated with points from other sales influences to determine overall market attractiveness (total points) of a team in any district. The four parameters for each of the seven price curves are on file in a table. A second table contains the allocations of these curves to particular districts (e.g., price curve 5 to market district 1, etc.). The administrator can easily make changes in the shape of the price curves (by inputting parameter changes in the first table) or he can reallocate price curves to different districts (using the second table). This can be done during the play of a game but it may be undesirable to modify the environment too much while play is in progress.

Before leaving Fig. 4 it should be noted that the model must protect itself against extreme strategies such as ridiculously high prices. This can easily be coped with by having a rapid decline into negative points allocation (to the right of Fig. 4) which will generate zero sales if the points sum over all sales influences is negative. Similar arrangements are made for the family of curves which determine potential orders as a function of the mean number of points of teams competing in an area. Fig. 5 shows the effect of a parameter value on one such curve. The family of curves used represent different market populations and total reation to market influences. Here again the choice of parameters and allocation of curves to districts are contained within tables which can readily be changed by the administrator.

3.6 *Open ended design.*

Not only does the general design encourage expansion

by the modification or addition of decision subroutines, but, throughout the program, systematic arrangements have been made to allow special inputs which may reward or punish teams for actions taken. For example, an external analysis and report on a production problem may be rewarded by improved productivity and a marketing campaign may earn temporary or permanent inprovement in specific districts. The design is analagous to house wiring where a number of plug points are provided without prior commitment to their specific use.

4. ADVANTAGES FROM THE DESIGN FEATURES

The game structure has been designed to offer considerable flexibility to the administrator. Decision blocks may be bypassed (see 3.3. above); they may be included but in a passive state (i.e., no changes are allowed in the current values of the decision variables); they may be passive by preference of the teams; or they may be active. At different points in an academic course, model segments may be omitted, passive or active. Extensions, or alternative decision blocks, can be incorporated without fear of overburdening the players with excessive complexity.
Similarly the administrator can select the number of active and passive teams he wants to play in a particular game. Examples would be: one active team only; one passive team only; one active team and one passive team; six active teams, etc. He can keep a game interesting for the teams by judicious changes (see 3.4); he can readily select the market structure which suits his purpose (see 3.5) or he can couple the game with other course work (see 3.6). **For example, a** team could present a report on an approach to evaluate employee morale within its company with some ideas for improvement from the real world. After a suitable time lag, the estimated costs and benefits of such an effort could be introduced into the game. The need for this type of activity may be questioned but if a complex game is to be effective, it should be supported by faculty from a number of disciplines. Those whose subject material is primarily qualitative should not be excluded. The game can provide a forum for the contributions and interacting effects of different disciplines. Its usefulness will diminish if it concerns itself exclusively with quantitative elements. Students react favorably to attempts to integrate material from different courses and have additional incentive to undertake qualitative assignments if they see 'tangible' rewards such as improved productivity in their company.

For the student, the most important feature of the design is the wide range of personal involvement he can choose from. Even in the most complex game, he can do almost nothing or he can spend a considerable amount of time and still not exhaust all his alternatives. This will remain true over time even if his team has gained a very strong or a very weak position. The computing possibilities will be discussed later.

As a specific example to illustrate this section, an administrator might wish to use Model 11 where only the variables price and advertising are relevant. He sets up the price structure (see Fig. 4) by assigning his choice of curves (or variations) to the 14 districts (or fewer). He similarly allocates to districts the responses to advertising. He chooses the number of teams which are to be active and passive in this game. During the play he may wish to change parameters, to reallocate curves and number of active teams, etc. He may later include other variables by switching to a different model number. His purpose might be to have teams estimate the underlying relationships and detect when they have shifted. He can illustrate the impact of complexity from more teams, variables or market instability. He can simulate a monopoly, an oligopoly, the impact of active and passive competition, etc. These alternatives are available within the present program. If he wished to combine the marketing variables in some different fashion, the program would require a simple modification. This version would then be available as an option.

5. COMPUTER ASPECTS

The next portion of the paper will be concerned with aspects of the project. This can be broken into four parts:

5.1. Implementation considerations.
5.2. Sample outputs.
5.3. Potential for student computer effert.
5.4. Post game discussion.

During the play neither the administrator nor the teams need concern themselves with the underlying computer processes. Of course, it will be necessary to have the output checked and any errors corrected, but most of the discussion of the data processing aspects takes place after the game is over.

5.1. *Implementation considerations.*

This will be discussed in three categories: 5.1.1. general, 5.1.2. files, and 5.1.3. error control.

5.1.1. General Considerations

For each simulated year, four primary and four secondary runs are processed. As stated earlier, the secondary runs deal exclusively with the hiring of personnel; all other decisions are handled during the primary runs. The full game has been run with 60 students through two simulated years of play (i.e., 16 cycles). This experience will be described more fully later.

The main program is written in FORTRAN IV, requires over 4,000 source instructions, compiled in about 4½ minutes and runs in object code in about 27 seconds for a primary run. [5] It requires, at present, 365 K bytes of core storage and reserves ten cylinders of disc capacity for the data sets.

The first choice to be made in designing a game such as this was between writing a single general purpose program for all runs and all models and providing a separate program for each alternative. The advantage of this second approach is the reduction of space required for the simpler models, but it complicates the program development and maintenance problems and was not used. Care must be taken in planning storage use. The general approach is to branch from the main program to successive subroutines, each designed to process a separate area of decisions (e.g., machines or plant or salesmen, etc.). Each subroutine should accept input from the data set on disc and decisions from cards, process these accept input from the data set on disc and decisions from cards, process these to the main program. Extensive use is made of COMMON the overlay arrays and to pass necessary values between the main program and various subroutines. The sequencing of subroutines is important, since some are interdependent for

data. In such acase, careful consideration of subroutine A followed by B or vice versa may save a significant amount of storage. Segments of program develop for one subroutine may be useful for others. For example, there is similarity between some of the processing requirements for line foremen and salesmen, and between machines and trucks. This parallelism should be planned for before either subroutine is developed. Useful experience can be gained of the gains and losses of adapting a program segment for a somewhat similar purpose, versus developing a new segment. In areas where the processing is identical, it is of interest to see if a team's policy seems to be consistent in these two areas.

5.1.2 File Matters

The data in the files are of four types:

(1) Data values available to the administrator (e.g., cost tables).

(2) Current decision values for each team (e.g., prices).

(3) Current resources of each team and the administrator (e.g., teams' personnel ant those in the labor pool).

(4) Data values from an earlier run which need to be retained for a later time period (e.g., raw materials ordered last period).

The data are oganized in a sequential stream in the order of the subroutines which use them. Sequential processing is efficient for the primary run but less so for the secondary run since only the personnel portions of the file are effected by the latter. At one stage, the file was segmented with all the personnel data in one subfile and other subfiles chosen to suit the different models available. This approach worked but was abandoned because of complication in the file control statements and an excessive number of operator responses required to authorize the processing of each segment of data. In the present form of one segment, the operator responds three times: to allow the reading of the input data set, the writing of the output data set and the rereading of the output data set to confirm its correctness.

The program processes through all subroutines (computations for most subroutines will be skipped if it is a secondary run) and then passes through each of them again to read back and list the output data set as a check. A key value tells if it is the processing or read back pass through the subroutines.

In a typical batch processing application, an important consideration is the number of successive sets of data values preserved before the oldest set is overwritten. Here four data sets are maintained on disc but the initial data set (arbitrarily given reference number 8) is never written over. Processing cycles through regions of the disc labelled with reference numbers 9 through 12.[6] If a simpler model is chosen, only the data associated with the active subroutines are needed. As illustrated in Fig. 6, two ways to deal with this are 1) to condense the initial data set down to only the portions needed (shown as version 1) or 2) to carry passive data portions forward in their original form (version 2). Version 1 is superior for processing efficiency but does not allow a later switch to a more complex model without restarting. Version 2 would allow such a switch. The game is currently designed using version 1 but the administrator can choose a more complex model than initially desired and disallow decision making in certain areas. [7]

It is also possible to use this file structure to play more than one game at the same time.[8] If N+1 regions are used, N-1 games could be played. Care would be needed to process the games in strict order from left to right across each row. Each game could be a different model number if desired. It was explained earlier that the main program selected or bypassed subroutines according to the model number. It would be technically feasible to number each subroutine and to allow the administrator a completely free choice to select any combination of subroutines he wished included. However, a number of possible selections would be illogical (e.g., omission of price) and the present program forces a choice from the set of sixteen. This is not restrictive since any decision area can be declared passive. From one run to the next the only input that varies are two control cards and the data deck. These control cards select the input and output data sets which are appropriate for the period of processing.

5.1.3 Error Control

A sample from the input forms is shown for a team in Fig. 7 and for the administrator in Fig. 8. The input alternatives are amplified for line foremen in Fig 9. Note that data fields can be multipurpose – for example – the work station can be genuine (value in certain ranges) or it can transmit a code (99=fired). The formats in the program may require:

(1) An optional input from a team (e.g., 1 of Fig. 7).

(2) An input with data from all teams (e.g., K of Fig. 8).

(3) An optional input from a team (e.g., 3 of Fig. 7).

Additional cards are punched by the administrator and the deck is assembled systematically using all the administrator forms. The data deck can become quite large and it is desirable to sift out data errors before disturbing the data set. A listing program points out the assembled data deck by team, with a complete copy for the administrator. A sample is shown in Fig. 10. This list of decision is approved by each team if correct. The data deck is then transferred to the main run without adjustment. Each team retains the listing as a record of its submitted decisions. It quickly becomes clear that some errors will invalidate the listing run (e.g., a mismatch between the stated number of decisions and the decision cards provided), while others are valid but not intended (e.g., a $500 salary increase when $50 was intended). The former error should be picked up as the data deck is being assembled, while the latter should be detected by a team's review of its listing. The forms themselves try to reduce the incidence of the latter error by indicating column boundaries for fields. These should be used to help verify keypunching.

During the first actual use of the game, after two processing cycles where there were occasional errors of the second kind (encouraged, in part, by ambiguity in one of the forms), these procedures were successful in generating correct input data decks. Teams should be protected from catastrophic blunders in their input which are legal but clearly not intended. For example, the order card for raw materials may be blank. In such cases, the program defaults to the values for the previous quarter.

5.2 *Sample Outputs*

The output of each subroutine is normally printed just before returning to the main program. In most cases it is printed all at one time and consists of results for a team and duplicate information for the administrator augmented by additional game data and some diagnostics to confirm the proper functioning of the program.

Figure 11 shows an example for non-line (support) personnel. The output for the administrator may involve a

separate page for each team (e.g., Fig 11 upper half) or may be gathered on one page for all teams as is done elsewhere in the model.

Other considerations become relevant when we deal with subroutines using files. Figure 12 shows a typical file for line workers for one team. The level of detail is intended to be illustrative of a small business data processing file and is set to discourage manual processing. This file contains items known to the team (first three columns) and items on ability factors for which a team only has estimates. The latter factors are not typical of real world personnel files. They are utilized here to support the game -- for example -- the foreman potential becomes relevant only if a man is promoted. It helps to determine his ability as a foreman. The processing of these elements is typical, however. Figure 13 shows an output produced in the run that generated the file shown, in part, in Fig. 12. Here again the output a team receives is duplicated and amplified for the administrator. However, in this case, a number of output messages are generated at the time of processing of transactions against the master file and these must be disposed of immediately or stored for later output. The simplest method is demonstrated in Fig. 13, with the output cut down the middle. Other alternatives include storing the messages as written or using codes for messages which would be put out later in separate pages at the end of the subroutines. The output could also be placed in seven regions in temporary storage on disc for the administrator and the six teams, and these regions could be dumped at the end of the run. This approach would eliminate the time spent at present in separating the output (about 40 minutes per primary run) at the cost of processing time and temporary disc space.

Figure 13 illustrates different types of output messages. Message A is 'advisory,' i.e., it confirms something which a team could compute for itself -- in this example, the availability of a group which has completed its training requirements. Message **B** is 'potentially recriminatory'. It might be prefaced by 'do yoe realize that...' MessageC is generated by the program and cannot be predicted by the team. The procedure for losing people assumes that some turnover is unavoidable but the probability of losing others will be influenced by how well they are being treated.

For each group of workers tested, a measure of satisfaction is computed from comparisons of quantifiable variables such as ability and rewards within a team and within the industry. It is well recognized that qualitative factors may be of greater significance in satisfaction in real life situations. Such 'inadequacies' in the game might stimulate more vigorous discussion of equivalent problems in real life, particularly if there is a deliberate attempt to dicuss afterwards what significant real life attributed cannot be represented adequately in models. The algorithms for turnover can be adjusted by the administrator to his taste. He can increase it significantly or eliminate it completely.

If, due to error or desired changes, a decision period is rerun, such probabilistic events in the model may result in some differences in the outputs in all areas following the first section which is modified. This is due to the effect of the changes on the random number stream. It can cause confusion if the erroneous run has been distributed, since a team may find output changes which apparently are not directly affected by the modification that caused the rerun. (e.g., an error in the machine subroutine, when corrected, may later cause salesmen who has 'left' to stay and vice versa.) This problem of replication is familiar to those engaged in simulation experiments.

The last kind of message stems from illegal procedure. This is not illustrated in Fig. 13, but if this team attempted next primary period to assign L-MEN No. 17 to further training, the output would indicate that no such line group existed and the order was ignored. Similarly, if a line group was ordered to further training for two months but this group only had one month of eligibility, the output would be that the order had been adjusted to one month only. It is essential that all such illegalities be anticipated and the action taken be published.

It can be seen from Fig. 12 that the loss of line groups 17 and 57 are indicated by a zero in the first column. The other details in these rows have not been removed since it is useful to retain them temporarily but periodic housekeeping is necessary. Figure 14 shows that the two line groups who left team 5 have duly turned up in the pool of available linemen. The accurate details, as they are on disc, are shown at the top, while the lower half gives the estimates of abilities which would be available to the teams. Summary outputs for a decision period are shown in Figs. 15-19. These values are built up in a table throughout the exercise and are printed for each team at the end of the finance subroutine.

5.3 *Potential for Student Computer Effort*

Some projects can be carried out by students solely on the basis of the data available to them during play; others require the collaboration of the administrator. For example, in the first use of the game, one student wrote a PL/1 program to evaluate salesmen's performance, since he was responsible for decisions in this area, and he was also learning this language. This was carried out independent of the adminitrator, in contrast to a second project where a student was allowed access to the data base to reorganize part of it into more useful forms. This saved teams some clerical effort and he intended to sell or license this product to the other teams, but it was not available in time to be of interest.

Perhaps the simulation possibilities are of the greatest interest. It was mentioned earlier that the number of successive data sets before overwriting is important. This is normally thought of in terms of protection against processing errors. However, in this gaming environment, the data set structure also allows simulation forward for up to three decision periods without disturbing the current data values (since four data sets are preserved).

If the administrator allowed it, teams could use this feature to test strategies incorporating anticipated counter strategies of the other teams if they so desired. A maximum of three periods forward is quite restrictive since intermediate processing cycles have to be counted. This capability can be upgraded by using more disc space and if N+1 regions (including initial) are used, N-1 forward cycles can be processed without overwriting. The multi-game possibility mentioned earlier would compete for disc space with these simulation options. Similarly an administrator might allow a team to replay a couple of earlier periods with different inputs provided the current values were undisturbed. More generally, the past decisions from the current and previous games provide a fertile source of experimental data.

Access to the program for replays or simulation forward, as discussed above, is equivalent to being granted the 'real world of the game' for experimentation.

Information from this could be compared to data from models of this 'real world' developed by the students. The administrator must also decide if he will release to successive teams the efforts of their predecessors, or whether more would be gained by repeating some work, perhaps with different results.

5.4 *Post Game Discussion*

Normally, post game analysis of a gaming experience tries to strengthen the insight into aspects of management. A separate attempt is made here to reinforce understanding of the computer's role in support of business applications. The discussion can be keyed to the design and implementation of this game, to computer projects developed for it, and to any interesting occurrences during play. The important advantage is that all the students are quite familiar with the environment, having been involved with if for some time.

They can be shown what needs to be considered to get a significant application working and to support it in operation. They can reflect on the generality of the flexible features of this model. In all aspects, there will be alternatives with advantages and disadvantages. The implementation considerations discussed earlier will be of interest. The need for error control and good documentation can be stressed. Separate attention should be given to errors in input, processing and the program (requiring program maintenance).

A number of cycles may require to be rerun. An understanding of the consequences of these types of errors and ways of minimizing them or their effect, is of value to a student, particularly if he is to be involved with data processing in his career. If students have developed programs for the game, they should be required to document their efforts in a standard manner. These standard requirements should be discussed with all students, using game exhibits as examples. The students can consider the various types of outputs from this experience and relate them to the underlying processing. They can study the reports, the different kinds of messages available as described in the text, and the alternative layouts that are possible. Evaluation can be made of projects, both for their worth and for their technical aspects -- for example, the potential of a particular language. Out of all this, students should understand more clearly what is involved to use a computer effectively in business, what opportunities might be available and the advantage of pursuing them.

6. GAME EXPERIENCE

The full model was first used with 60 students over a twelve-week period. They received handouts on 'Information for the Players' and the 'Starting Position' in addition to outputs for all six teams for three periods based on sample decisions which were chosen for their diversity. The 'Information for the Players' contains a summary of the game, more specific information on each of the decision areas and detailed directions on the input procedures using the forms. The 'Starting Position' describes the current decision levels, assets and outstanding commitments. Certain items in the files are given as estimates. Eight primary and eight secondary decisions were made with a primary and secondary sequence taking about a week at the beginning of play and about four days near the end. As expected, some bugs appeared and a few more were generated by some changes in the program as a result of an early evaluation of the model by the players and the administrator.

However, the extensive pre-testing and supporting diagnostics avoided any holdup in the play itself. The one example of a spectacular error [9] was triggered by a change based on immediate operating experience. After a period or two of familiarization the runs proceeded smoothly with only an occasional minor exception. The game was run as an extra-curricular activity. It was recognized in advance by the author and by the participants during the exercise that such a mode of play falls far short of realizing the potential of this kind of activity. It is essential that a more formal setting be organized with a significant faculty effort from a number of disciplines. This first run might be considered more of a final checkout phase, although the interest level of most of the participants was very high and a post-play questionnaire indicated great hopes for its future usefulness. As mentioned earlier, two student projects were undertaken during the first game.

7. POSSIBLE FUTURE PROJECTS

The present form of the game provides a basic framework for further developments of interest to faculty and students. Useful extensions or modifications could be developed in such areas as -- data organization, technical support such as the use of terminals and displays, the development of decision programs to complement or replace human effort, the use of different languages for comparison of performance and ease of use, the use of different decision rules such as alternative ways of merging various sales influences, additional decision subroutines, etc. The common framework encourages such extensions in the form of small research projects that would benefit both the immediate participants and succeeding groups of students.

NOTES

1/The environment was designed while the author was at the Glasgow University Computing Laboratory and has been implemented on an IBM 360 Model 67 at the Naval Postgraduate School, Monterey, California.

2/One of the best known examples is widely used by IBM. In this model, three teams compete to sell a product in four districts. Each decision period a team allocates its available funds over eight alternatives: production, additional plant, marketing in each district, research and development, and consultancy information. Prices can also be changed in each district. The model processes these inputs from each team to generate production and marketing results, and the funds available for the next period. This cycle is repeated as often as desired (twenty times would simulate five years of play). Perhaps the most remarkable feature of this simple model is the considerable enthusiasm it can engender in team members for at least a full day of play.

3/See, for example, Thomas H. Naylor. 'Simulation and Gaming.' Bibliography 19, Computing Reviews, Vol. 10, No. 1, Jan. 1969. Martin Shubik. 'Gaming: Costs and Facilities.' Management Science, Vol. 14, No. 11, July 1968. K. J. Cohen and E. Rhenman. 'The Role of Management Games in Education and Research.' Manage-

ment Science, Vol. 7, No. 2, Jan. 1961. J. M. Kibbee, C. J. Craft.' and Burt Nanus. 'Management Games.'
Reinhold Publishing Co., New York, 1961. P. S. Greenlaw, L. W. Herron, and R. H. Rawdon. 'Business Simulation.' Prentice Hall, Englewood Cliffs, New Jersey, 1962.

4/e.g., If a salesman left a team (or was fired) at the end of quarter 1, he would be listed as available for the next hiring run (quarter 1.5). He could be hired before the start of quarter 2 and assigned by his new employer via a wuarter 2 input..

5/The computer was an IBM 360, Model 67. The operating system was OS/MVT and the IBM 2314 was the disc unit used.

6/ 2314 Disc Layout

Disc (each region contains 2 cylinders of space)	(Under this management) Year	Quarter	Input Region	Output Region
Region 8 (Initial Processing Only)	1	1	8	9
Region 9		1.5 (hiring)	9	10
Region 10		2	10	11
Region 11		2.5 (hiring)	11	12
Region 12		3	12	9
		3.5 (hiring)	9	10
		4	10	11
		4.5 (hiring)	11	12
	2	1	12	9
		.	.	. etc.
		.	.	.
		.	.	.

7/ For example, Model 10 could be used with initially no marketing decisions allowed. Variables such as prices would continue at their initial level until permission was granted for changes.

8/ For example,

	Game A Input	Game A Output	Game B Input	Game B Output	Game C Input	Game C Output	
Process	8	9					
	9	10	8	9			
	10	11	9	10	8	9	
	11	12	10	11	9	10	
	12	9	11	12	10	11	etc.

9/ The salary rate in salesmen files was changed to represent his return for a week instead of a month. The test to see if a salesman is satisfied with his salary includes comparison with his peers but it was not adjusted to allow for the change and to each salesman the industrial average was of the order of four times his salary. Every salesman quit, flooding arrays and causing general chaos.

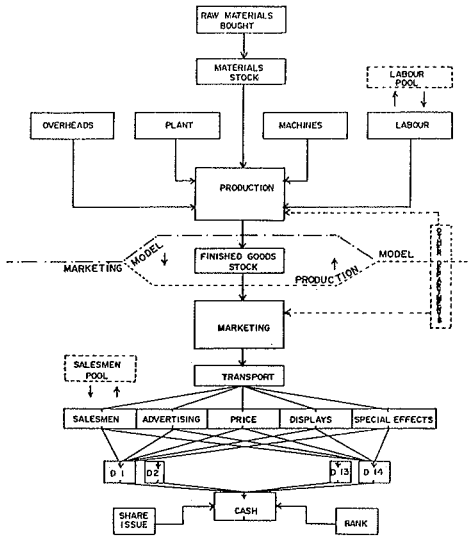


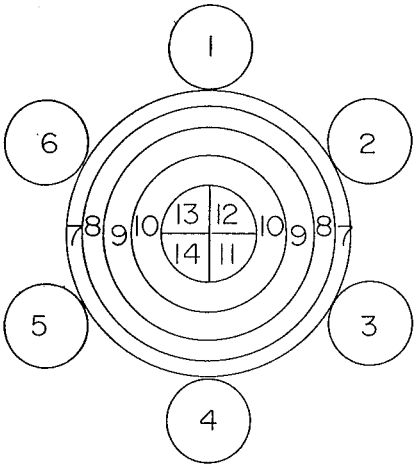
FIG. 1.

MODELS AVAILABLE

(An X indicates that this column is included in the model number of that row.)

Model Number	SPECIAL: Funds for production this period and next	SPECIAL: Orders this period estimate orders next two periods value this pd.	INPUTS: Raw mats	PRODUCTION: All other production variables	MARKETING: Price advert.	MARKETING: Salesmen	MARKETING: Other sales effects	MARKETING: Orders	MARKETING: Finished goods	MARKETING: Trucks	FINANCE
1	X		1	X							
2	X		3	X							
3	X	X	1	X					X		
4	X	X	3	X					X		
5			1	X	X			X	X		X
6			3	X	X			X	X		X
7			1	X	X	X		X	X		X
8			3	X	X	X		X	X		X
9			1	X	X	X	X	X	X	X	X
10			3	X	X	X	X	X	X	X	X
	Funds for marketing this period and next	Production and current manufacturing costs of teams									
11	X				X						
12		X			X			X	X		X
13	X				X	X					
14		X			X	X		X	X		X
15	X				X	X	X	X			
16		X			X	X	X	X	X	X	X

FIG. 2.



MARKET AREAS

Areas 1 - 6 are towns each being the factory base of one team.

Areas 7 - 10 are country districts with a smaller population/district.

Areas 11 - 14 are city districts with a larger population/district.

The country and city districts are considered to be equidistant from the towns.

FIG. 3

FIG. 4

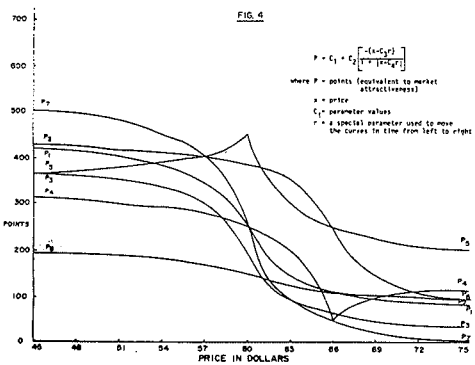
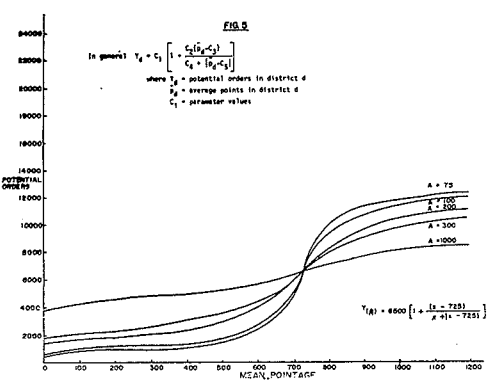
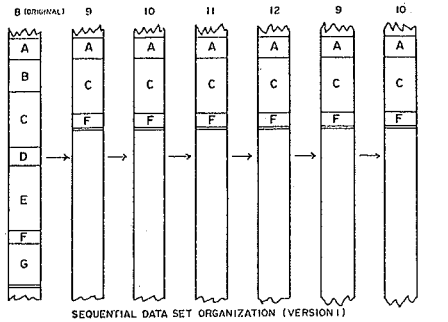


FIG. 5





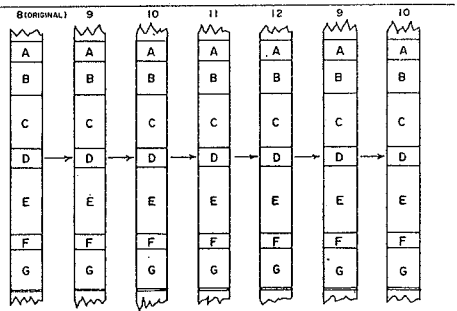


FIG. 6

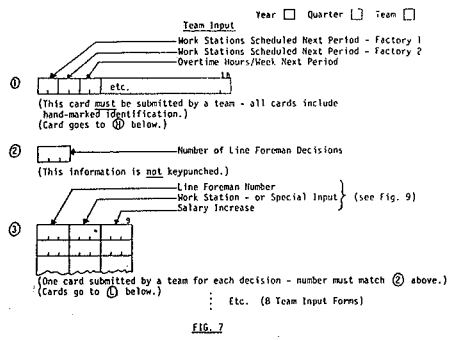


FIG. 7

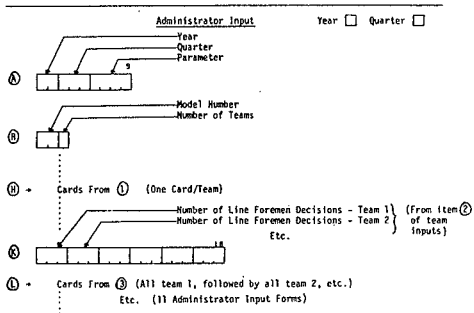


FIG. 8

INPUT ALTERNATIVES - LINE FOREMEN

A	B	C

A = Line Foreman Number
B = Work Station Number
Use Odd Numbers in Range:
1- 95 (present factory)
101-147 (new factory)

Ex. 1 | 0 2 | 0 7 | 1 0 0 | → foreman 21 is to supervise station numbers 71 and 72
B = EVEN - corrects to ODD - one less

Ex. 2 | 0 1 5 | 0 2 2 | 0 0 0 | → foreman 15 supervises station numbers 21 and 22
B = 97 - Foreman sent for one month of further training
B = 98 - Foreman sent for two months of further training
Eligibility will be checked in each case and he will return at end to present work station

Ex. 3 | 0 3 6 | 0 9 8 | 0 0 0 | → foreman 36 sent for further training for 2 months
B = 99 - Foreman fired - notice - 1 quarter

Ex. 4 | 0 0 1 | 0 9 9 | 0 0 0 | → foreman 1 fired
C = salary increase (individual foreman)

Ex. 5 | 0 0 6 | 0 0 0 | 0 1 5 | → foreman 6 has a raise of $15 in his quarterly salary
C = 900+N for an N% increase in all foreman salaries
A = 0

Ex. 6 | 0 0 0 | 0 0 0 | 9 0 6 | → all foremen in this team get a 6% raise in quarterly salary

Ex. 7 | 0 1 7 | 0 9 7 | 0 1 0 | → foreman 17 sent for further training for 1 month and given a $10 raise

NOTE: Two separate cards for the same foreman would cause the second to be skipped with a note to that effect.

FIG. 9

FIG. 10

FOR YEAR 1 PERIOD 1.5 SALESMEN HIRING

TEAM NO	NO OF SALESMEN APPROACHED
1	1
2	2
3	3
4	4
5	5
6	6

FOR TEAM 1

SALESMAN NO	EX-TEAM	SALARY OFFERED	BONUS OFFERED
21	3	650	100

FOR TEAM 2

SALESMAN NO	EX-TEAM	SALARY OFFERED	BONUS OFFERED
7	7	450	50
25	7	550	0

FOR TEAM 3

SALESMAN NO	EX-TEAM	SALARY OFFERED	BONUS OFFERED
21	2	520	200
16	3	450	50
21	3	600	300
SALESMAN NO	EX-TEAM	SALARY OFFERED	BONUS OFFERED

(Excerpt from the salesmen hiring decisions listing for the administrator)

FOR TEAM 1 YEAR 1 PERIOD 1.5 SALESMEN HIRING DECISION

NO OF SALESMEN APPROACHED = 1

SALESMAN NO	EX-TEAM	SALARY OFFERED	BONUS OFFERED
21	3	650	100

(Portion of the above for Team 1 - normally on a separate page)

FIG. 11

```
M= 1    NO=>    22
M= 1    NO=>    23
        NO=>    71
        NO=>    72
        NO=>    73
        NO=>    74
        NO=>    75
        NO=>    76
```

OUTPUT FOR YEAR 1 PERIOD 1	MLA	MLAF	MLB	MLBF
NOS MEN NEXT PERIOD	30.	3.	40.	4.
WAGES THIS PERIOD	26190.00	2819.80	40840.00	5145.99
TRAINING COSTS THIS PERIOD	4500.00	0.0	0.0	240.00
NOS OF THISTEAM WHO LEAVE TEAPE	1	0	3	0
NOS OF THIS YEAR TO POOL	0	0	4	0
NO OF THOSE WHO ARE POTENTIAL FOREMEN	0		1	

MEN NEXT PERIOD 31 2 2 1 1 3 0 82 4 10 EFFICIENCY=1.050

Administrator Output for Non-Line Personnel

ITEM 1 YEAR 1 PERIOD 1

TEAM NO 9

	MLA	MLAF	MLB	MLBF
NOS MEN NEXT PD	30.	3.	40.	4.
WAGES THIS PERIOD	26190.00	2818.80	40840.00	5145.99
TRAINING COSTS THIS PERIOD	4500.00	0.0	0.0	240.00
NOS OF THIS TEAM WHO LEAVE GAME	1	0	3	0
NOS OF THIS TEAM TO POOL	0	0	9	0
NOS OF THOSE WHO ARE POTENTIAL FOREMEN	0		1	

EFFICIENCY =1.000

Corresponding Team Output for Non-Line Personnel (normally on a separate page)

FIG. 12

L-MEN NO	CAT NO	STATION NO	Foreman Potential P=POTENTIAL	CURRENT RATING	Further Improvement Factor IOC	Rating Potential Without Further Training EXPERIENCE	P	Speed of Learning Factor IOC
41	1	71		1	1	0	1	1
42	1	72		1	1	0	1	1
43	1	73		1	1	0	1	1
44	1	74		1	1	0	1	1
45	1	75		1	1	0	1	1
46	1	76		1	1	0	1	1
47	1	77		1	1	0	1	1
48	1	78		1	1	0	1	1
49	1	79		1	1	0	1	1
50	1	80		1	1	0	1	1
51	1	81		1	1	0	1	1
52	1	82		1	1	0	1	1
53	1	83		1	1	0	1	1
54	1	84		1	1	0	1	1
55	1	85		1	1	0	1	1
56	1	86		1	1	0	1	1
57	1	87		1	1	0	1	1
58	1	88		1	1	0	1	1
59	1	89		1	1	0	1	1
60	1	90		1	1	0	1	1
61	1	91		1	1	0	1	1
62	1	92		1	1	0	1	1
63	1	93		1	1	0	1	1
64	1	94		1	1	0	1	1
65	1	95		1	1	0	1	1

Partial Listing of Line Workers - Team 5

FIG. 13

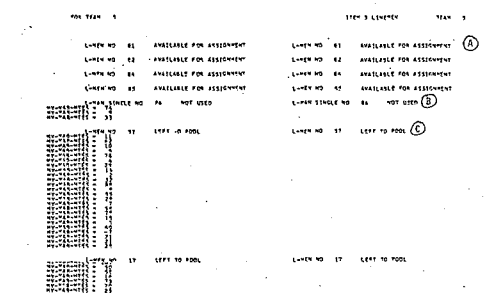


FIG. 14

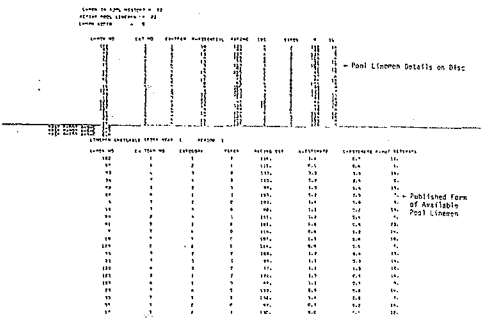


FIG. 15

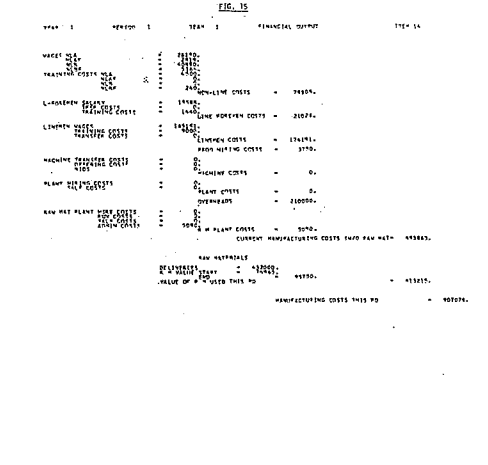


FIG. 16

PERIOD 1 YEAR 1 TEAM 1

ADVERTISING COSTS = 43000.

SALESMEN SALARY = 7145.
TRAINING COSTS = 0.
SALESMEN COSTS = 7145.
SALESMEN MEETING = 300.
DISPLAY COSTS = 27000.
R AND D COSTS = 0.

FIN GOODS ADMIN COSTS = 10000.
WAREHOUSE HIRING = 0.
RUN = 0.
SALE COST = 0.
FIN GOODS COSTS = 10000.

TRUCK RUN COSTS = 10500.
DRIVING HIRING = 0.
RUNS = 0.
TRUCK COSTS = 10500.
MARKETING COSTS = 144965.

FINISHED GOODS
MFG COSTS LAST PD = 460000.
F G VALUE START = 48000.
END = 121040.
COSTS OF GOODS SOLD = 878960.

	DEPREC FUND	SALE INCOME	WAGE&SALES
MACHINES	52819.	0.	0.
PLANT	60750.	0.	0.
R M PLANT	0.	0.	0.
F G PLANT	0.	0.	0.
TRUCKS	18510.	0.	0.
TOTAL	131879.	0.	0.
PRIOR TOTAL	114756.		

FIG. 17

PERIOD 1 YEAR 1 TEAM 1
PROFIT AND LOSS STATEMENT

OUT		IN	
COST OF GOODS SOLD	= 878960.	SALES REV GENERATED -	= 1314780.
MARKETING	= 144965.	+ AND EQ SALE	= 0.
SPECIAL	= 0.		
INTEREST	= 4500.		
FINAN ADMIN	= 0.		
TOTAL	= 1028425.		= 1314780.

PROFIT = 286355.
YEAR TO DATE IN = 1314780.
OUT = 1028425.
PROFIT = 286355.
TAX = 0.
ACCUM TAX ADJ = -50000.
DIVIDEND
BANK ARREARS = 147000.
UNTARMED = 0.
PAID THIS PD
BANK = 0.
DAD = 0.

MONEY SOURCES

TYPE OF FUND	AMOUNT	PER CENT	PERIODS TILL DUE
1	300000.	6.0	5
3	1050000.	7.0	0
4	2250000.	8.0	0

FIG. 18

PERIOD 1 YEAR 1 TEAM 1
CASH FLOW

OUT		IN	
MANUFACTURING	= 493863.	SALES RECEIPTS	= 1320000.
RAW MATERIALS	= 432000.	PL AND EQ SALE	= 0.
MARKETING	= 144965.	MONEY ADDED	= 0.
SPECIAL	= 0.	UNEARNED REV	= 0.
PL AND EQ PUR	= 0.		
DEPREC ALLOC	= 17123.		
MONEY REPAID	= 0.		
INTEREST BANK	= 4500.		
DEB	= 0.		
EMER	= 0.		
FINAN ADMIN	= 0.		
DIVIDENDS BANK	= 0.		
DAD	= 0.		
TAX	= 0.		
TOTALS	1092491.		1320000.

BEGINNING CASH = 100000.
CLOSING CASH = 327549.

FIG. 19

PERIOD 1 YEAR 1 TEAM 1

	LAST YEAR (total)	THIS YEAR (to date)
YEARLY INCOME	= 4000000.	= 1314780.
YEARLY EXPENSE	= 3900000.	= 1028425.
YEARLY PROFIT	= 200000.	= 286355.
END OF YR DEPREC FUND	= 109056.	= 131879.
END OF YR CASH	= 100050.	= 327549.
TAX PAID	= 0.	= 0.

TYPE OF FUND	AMOUNT	PER CENT	PERIODS TILL DUE
1	300000.	6.0	5
3	1050000.	7.0	0
4	2250000.	8.0	0

TEACHING SOCIOLOGICAL RESEARCH METHODS WITH THE USE OF A COMPUTER

Dean H. Harper and Hanan C. Selvin

1. THE COMPUTER SIMULATION OF OPINION FORMATION IN A POPULATION

Underlying the computer simulation is a complex but rather general model. It is complex in that it tries to capture all of the important features of mass phenomena, rather than provide a parsimonious theory for any instance. It is general that it can be used for any one of a number of different sociological problems, e.g., attitudes, behavior or past experiences.

The computer simulation is modular; it has several components related to each other in the sequential arrangement indicated in Diagram I. Some of the components can be bypassed. Further some of the components consist of a 'family' of alternative procedures--any one of which is used for a particular simulation. Several of these components will be discussed briefly--corresponding to the boxes of Diagram I.

1.1 *The Simulated Population*

The sociologist needs to describe the population (Box A). This could be a hypothetical population or it might be that of a particular community or society. The population is described in terms of its distribution in respect to a number of demographic variables, e.g., sex, race, education, and the like.

If the simulation deals with a dynamic population, then the sociologist must also supply parameters of change, viz., birth rates, death rates, marriage rates and the like. By Monte Carlo devices, the program then determines which people die, marry or have children (Box F).

1.2 *Opinion Formation Components*

If the simulation is static, then it begins with the

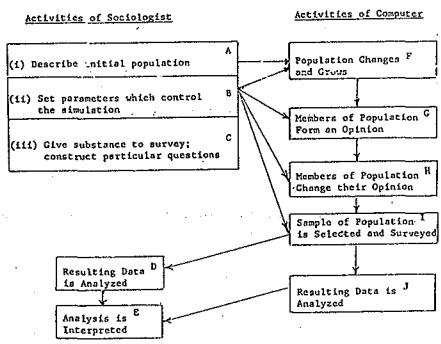


Diagram I: Human and Computer Components of the Simulated Survey

opinion formation component (Box G). In this component, members of the population undergo experiences and form opinions relative to the issue of interest.

The model for this segment is shown in Diagram II. Diagram II represents a rather general theory which expresses the causal connection between four types of variables: (i) demographic traits of respondents, (ii) their past experiences, (iii) their psychological traits, (iv) their attitudes and opinions. The demographic variables serve as independent variables and the attitudes and opinions serve as dependent variables; experiences and psychological traits are intermediate variables--partially dependent on the demographic characteristics and independent in respect to opinions and attitudes. These relations reflect the time order assumed among the different variables; that is, for example, psychological traits are formed in the individual prior to his opinions and attitudes about specific issues.

The exogeneous variables are those that have not been explicitly included in the survey. This will, of course, vary from survey to survey. This model is represented in linear equations like the following:

(1) $y_{ik} = \Sigma_j a_{ij} X_{jk} + e_i$,

where X_{jk} is the amount of independent variable j possessed by respondent k, a_{ij} is the 'effect' of independent variable j on dependent variable i, Y_{ik} is the amount of dependent variable i possessed by respondent k, and e_i is an error term incorporating the effects of all variables not explicitly incorporated in the survey.

For example, X_{jk} might be education or age of respondent k, Y_{ik} might be the degree of political participation; a_{ij} would then be a measure of the effect of education on political participation. It would be constant over all individuals in the population, and is a parameter to be set by the sociologist doing the simulation.

Equation (1), then, states that a variable labeled political participation, which is an intermediate variable, is 'caused' by a number of other variables which work their effects in an additive way-- the effect of one being added to the effect of another. The term e_i incorporates the effects of the exogeneous variables. At another point in the computer routine for this component, X_{jk} represents both independent and intermediate variables and Y_{ik} represents some dependent variable. (The model can be made more complex by including quadratic terms and interaction terms. For a description of a different model, see Harper (1968). For other examples of the simulation of public opinion formation, see McPhee (1961) and Pool (1964).

1.3 *The Interview Component*

For the work reported here, the component of opinion change, represented in Box H, was bypassed; therefore, it will not be discussed. Once respondents form their opinions, then the program moved to the survey component, represented by Box I.

The inputs to this segment are Y_{ik}, the values of the independent variables i for respondents k. The interview segment contains a function that maps the continuous variables, Y_{ik}, to a discrete variable, Z_{ik}, which has, say, integer values from 1 to M_i, where M_i represents the number of response alternatives to question i.

For example, suppose question i, part of a survey on attitudes concerning overpopulation, was the following:

> Should two men with the same income but different size families pay the same amount of income tax or should the man with the larger family pay less?
>
> ——— same tax
>
> ——— larger family pay less

Here $M_i = 2$. And other questions can have 3, 4, or many more alternatives.

The function $f(Y) = Z$, can be one that partitions Y into M_i equidistant segments, or it can be more complex. But once the value of Z is computed, this becomes the coded response of the respondent, e.g., if $Z_{ik} = 2$ then respondent k gives the second of the two answers to the question above.

However, this is the response he would give if systematic error did not operate. The opinion formation component incorporated random error; the interview component incorporates systematic error. That is, there are a number of observation errors that intervene between respondent and interviewer, which, in some cases, alter the respondent's answer to the question. These errors are (1) the respondent is not at home or he refuses to be interviewed, (ii) he agrees to be interviewed, but he refuses to answer a particular question, (iii) he begins the interview but terminates it before it is completed, (iv) he gives a false answer to a particular question, (v) his answer is recorded incorrectly by the interviewer. The occurrence of error (i) is a function of the demographic characteristics of the respondent; errors (ii), (iii), and (iv) are functions of both demographic characteristics and of question. Error (v) is a function of interviewer and of question. These functions are parameters to be specified by the sociologist (Box B).

The output of this component is like that of the typical survey: a set of punched cards which contained the coded answers of a sample of respondents. At this point the simulation can either punch out data, which can be analyzed by the sociologist, or it can go directly to a data analysis component which analyzes the data in one of several ways; those are the ways that sociologists typically use: tabular analysis, regression analysis, path analysis. (For a discussion of this see (Hirschi and Selvin (1967).) However, data analysis is still an art so that the computer (Box J) is a poor substitute for the sociologist (Box E).

The computer simulation has two objectives: research and pedagogy. Among its research uses are the following: (i) to replicate and thereby 'explain' a set of real data, (ii) to devise a complex, as opposed to a parsimonious, theory of the process of opinion-formation and the process of surveying it, which will enable the 'sensitivity testing' of different elements in the theory, and (iii) to explore the relative usefulness of different techniques of data analysis. The pedagogical objectives of the simulation are discussed next. (For another example of computer usage in sociological training see (Meyers (1969).)

2. TRAINING STUDENTS WITH THE SIMULATION

So far, the simulation has been used in two ways: to train students (i) in the procedures of data analysis, and, (ii) in the techniques of questionnaire construction. Each of these will be discussed briefly.

2.1 *Teaching the Techniques of Data Analysis*

Among the topics that have been studied, with the aid of the simulation, are the following: (1) the use of different techniques of data analysis, e.g., tabular analysis, regression analysis, and so on; (2) the occasions when one or another technique is appropriate; (3) the effects of random, systematic and sampling errors in data--either alone or in

combination and how to deal with them in each technique; (4) the relative usefulness of different questionnaire designs and sampling schemes as these are affected by techniques of data analysis.

Students carry out a series of assignments of varying difficulty--each assignment directed to one of the topics above. In a final assignment, the student is given a set of data and asked to analyze it as if it were real data.

The student must decide which technique to use, and then use it--either with the aid of a counter-sorter and desk calculator or by going to the computer and using his own programs to analyze the data. He attempts to infer the causal structure underlying the data. Since the artificial data were generated from a known causal structure, which is known because it was arbitrarily created, his inferences can be checked against the 'truth'. In this way he can learn about the utility of a technique of data analysis and of his ability to use it. Further, he can compare his ability to make inferences from different techniques of analysis and thereby learn about the **relative** utility of different techniques.

In addition some students have been excited about the possibilities of the simulation and have created their own exercises. For example, one student simplified the simulated phenomena to include 7 independent variables and 2 dependent variables, without any kind of error. He then examined the relationship between the two dependent variables and 3 of the 7 independent variables and thereby studied the random error created by the four variables now considered to be exogeneous.

It should be pointed out that the student does not yet interact with a computer. Rather he is given a set of generated data to study and examine away from the simulation. It should also be noted that some of the problems which are studied could be investigated by statistical analysis to arrive at expected values. However, that the student works them out in the equivalent of a 'sampling experiment' does not detract from his appreciation of statistical analysis. On the contrary students become excited in the possibilities of survey research and they see the usefulness of statistics for sociology in a way that they did not through the more traditional methods of teaching survey research.

2.2 *Teaching Questionnaire Construction*

This second use of the computer simulation for training students focuses on the problems of constructing questionnaires and analyzing the pretest data--steps (3) through (6) among those articulated earlier. The **opinion formation** segment of the simulation was modified for this purpose. This modification was an attempt to incorporate into the simulation some of the effects of question wording and order on the results of the survey. Specifically, four effects were studied: those from (i) vague or ambiguous questions, (ii) questions which, by virtue of wording, evoked more than one attitude dimension, (iii) questions which evoked a response because of an extreme emotional content, (iv) the placement of questions relative to each other, i.e. response to one question is influenced by the previous questions that were asked. Obviously, these are not all of the errors that can be made in questionnaire design; but they reflect the major errors which students have made. For some discussion of these problems, see Deming (1944) and Hagburg (1968).

How to incorporate these effects into a simulation is not easily answered. There are a number of approaches. For example, one might view the respondent as a list-processing device. Each respondent is a repository of a number of lists. The list is an ordered sequence of verbal statements which represent the respondent's feelings and attitudes about particular issues. Answering a question consists of consulting different lists. If a statement on a list is similar to one of the response alternatives of the question, that response alternative is selected. Where a question is vague or ambiguous, the respondent may not find a response on any of his lists and hence resort to a 'don't know' answer, or he may find possible answers on several lists and choose one on a highly arbitrary basis. (For examples of this, see Abelson (1963), and Gullahorn and Gullahorn (1963).

Although this is, we feel, a useful approach, it is not the approach we took. Rather, we conceived of each question being characterized by numerical parameters representing (i) degree of vagueness, (ii) presence of words or phrases of high emotional content, (iii) the different dimensions dealt with in the question. These incorporate the first three effects mentioned above. These numerical parameters alter the functional relationship between independent and dependent variables expressed in equation (1) above.

For example, where a question has a high degree of vagueness, then the exogeneous variables become more important in influencing responses--which means that the student finds less of a relationship between responses and demographic variables.

The effects of question wording and the like are greater for some respondents than for others. That is, there is an interaction between respondent and question. To capture this, we also need to characterize respondents by those attributes that determine their reaction to question wording and order.

Two characteristics of the respondent are **salience** of an issue for him and his **feelings** about that issue. For example, if one issue is 'water pollution', then this can be of high or low salience for the respondent; further, he can vary from feeling that water is not polluted and have strong feelings about those who claim it is, to being indifferent, to believing that water is polluted and having strong feelings about those who pollute it or fail to prevent it from being polluted.

If a respondent has a high salience on the dimension 'water pollution' but a low salience on some other dimension--say political activity--then a question measuring these two dimensions will be most likely to evoke a response from him based on the first dimension rather than the second. Each time an issue is raised in a question, it increases the salience of that issue for subsequent issues.

These charateristics of respondents--salience towards issues and the feelings about the issues--are represented as dependent variables in the form of equation (1) above. That is, they are, in part, functions of demographic variables, psychological traits and past experiences; they are also functions of exogeneous variables. Further, the variable 'salience' is changed during the course of the survey in response to the questions that are asked.

This, then, is the structure of the stimulation as a training device. How does the student use it? He first constructs a questionnaire which is then examined by his instructor who characterizes each question, by assigning numerical values to the parameters, in respect to vagueness, slanted wording, and dimensions being measured. These

become input to the computer. A pretest sample of, say, 50 is generated. Attitude and salience scores are computed; Finally responses to the questionnaire are punched out.

The student analyzes the pretest data and then revises the questionnaire--and the parameters describing the questions are suitably altered by the instructor. For the second pretest, the revised parameters and the output of the first pretest becomes input. Then, in the second pretest, the student gets answers to his questionnaire from the same sample he used in the first pretest. That is, the sample of 50 give responses to his original questionnaire and then respond to his revised questionnaire as if they had not answered the first questionnaire. Between the two questionnaires, nothing changes in the respondents. Whatever changes that occur are due to the revisions in the questionnaire. Thus, the student can study the results of different questions on the same population. This is shown in Diagram III.

How this approach works can be explained best by an example. Students were asked to design a survey that would get at people's opinions and attitudes about environmental change, pollution and destruction; and, of course, they were expected to provide ultimately some understanding of the underlying causal structure of those attitudes, and therefore had to incorporate some theory into their survey.

In this assignment one student produced the following question:

Some people have demonstrated against environmental pollution. How do you feel about this?

—— for it

—— against it

Obviously, there are a number of difficulties with such a question. The word 'demonstrate' may have become a 'loaded' word, which evokes attitudes toward riots, hippies or blacks. Thus the question would be characterized as having a high degree of emotional content. In the pretest 12% of 50 respondents said they were 'for it' and 88% said they were 'against it'. It is not clear what these answers mean. Are the 88% who answered 'against it', opposed to environmental pollution or opposed to demonstrations? Thus, the question has a high degree of ambiguity or vagueness. Further, a question which most people answer in the same way does not provide much useful information; it would be a better question if the proportion choosing one of the alternatives was closer to 50%.

The student revised the question to:

By writing public officials, holding meetings, and the like, some people are expressing their opposition to environmental pollution.

How do you feel about their actions?

—— favor these actions against pollution

—— do not favor these actions against pollution

The second wording may remove the bias present in the first wording, but the question still has other deficiencies. The term 'environmental pollution' will mean different things to different people and hence will evoke a number of responses difficult or impossible to interpret. Hence, this question would still be characterized as having a high degree of vagueness.

Suppose environmental pollution were made more specific – e.g., 'pollution of rivers by industrial waste and ineffective sewage disposal'. Now the question is less slanted and less vague but it has one other shortcoming. It would seem to evoke attitudes about one of two different phenomena: collective activity and water pollution. Thus, the question would be characterized as measuring these two dimensions.

The second time, the student got the results shown in Table I which relates education, an independent variable, to response to this item, the dependent variable. That is, 19 of the 50 respondents had at least some college education. Of these, 54% (or 10) said that they were favorable toward these public actions on pollution, whereas of the 31 with a high school degree or less, 52% (or 16) were favorable.

One would expect that education would make a difference in how people answer this question; with increasing education, people would more likely favor political action against pollution. However, this expectation is not born out by the pretest data.

The student revised the question a third time – but into two questions: one concerned respondents' attitudes about water pollution and the other concerned political action against pollution. This time he obtained a sharper relationship between demographic variables and responses to the question. His third attempt resulted in a much better questionnaire than his first attempt and he clearly understood why it was better.

Now, it should be obvious that the computer simulation is not necessary to teach students the principles of questionnaire design; a teacher could make the same assignment and respond didactically pointing out the defects in the student questionnaire. But with the computer simulation, the teacher does this also. The difference is that the results obtained from the computer simulation make the lesson both more meaningful and exciting.

3. CONCLUSION

From these two experiences with the use of the computer as an aid in teaching research methods to graduate students in sociology, we conclude that the computer and computer simulation of phenomena offers considerable and still unmeasured potential in the education of the young. For us, it has made the lessons for our students more realistic and relevant, and it has motivated them to teach themselves in a way that earlier approaches had failed to do.

REFERENCES

Abelson, R.P., 1963, Computer Simulation of 'Hot' Cognition. In Tomkins, S.S., and S. Messick, eds., Computer Simulation of Personality (Wiley, New York), pp. 74-96.

Abelson, R.P., 1968, Simulation of Social Behavior. In Lindzey, G. and E. Aronson, eds., Handbook of Social Psychology, Vol. 5, (Addison Wesley, Boston), pp. 274-356.

Coleman, J.S., 1964, Mathematical Models and Computer Simulation. In Faris, R.E., ed., Handbook of Modern Sociology, (Rand McNally, Chicago), pp. 1027-1062.

Deming, W.E., 1944, American Sociological Review, 9, 359-369.

Glock, C.Y., 1967, Survey Design and Analysis in Sociology. In Glock, C.Y., ed., Survey Research in the Social Sciences, (Russell Sage Foundation, New York), pp. 7-62.

Gullahorn, J. and J.E. Gullahorn, 1963, Behav. Sci., 8 354-362.

Hirschi, T. and H.C. Selvin, 1967, Delinquency Research, (The Free Press, New York).

Hagburg, E.C., 1968, Public Opinion Quarterly, 32, 453-456.

Harper, D., 1968, The Computer Simulation of a Sociological Survey, In Gardin, J.C. and B. Janlin, eds., Calcul et Formulisation dans les Sciences de l'Homme (Centre National de la Recherche Scientifique, Paris), pp. 239-251.

Meyers, E.D. Jr., 1969, Project IMPRESS: Time Sharing in the Social Sciences, AFIPS Conf. Proc., Vol. 34, pp. 673-680.

McPhee, W.N., 1963, Public Opinion Quarterly, 25, 184-193.

Orcutt, G.H., M. Greenberger, J. Korbel and A.H. Rivlin, 1961, Microanalysis of Socio-economic Systems: a Simulation Study, (Harper, New York).

Pool, I., R.P. Abelson and S. Popkin, 1964, Candidates, Issues, and Strategies: A Computer Simulation of the 1960 Presidential Election, (M.I.T. Press, Cambridge).

Potter, R.G. and J.M. Sakoda, 1966, Demography. 3, 450-461.

Rosenberg, M., 1968, The Logic of Survey Analysis (Basic Books, New York).

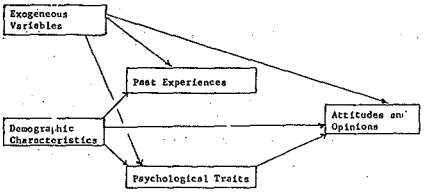


Diagram II. Model of Opinion Formation.

	High School Degree or Less	Some College or More	Total
Favor Political Actions	51.5%	53.5%	52%
Oppose Politic l Actions	48.5	46.5	48%
	(31)	(19)	(100)

Table I: Relationship between Education and Attitude about Political Actions on Pollution

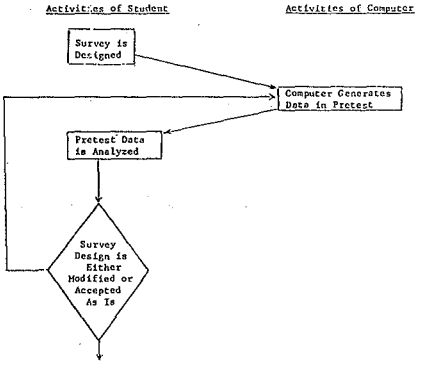


Diagram III. Use of the Simulated Survey as a Device for Teaching Questionnaire Construction

GENETIC EXPERIMENTS ON A COMPUTER

Robert Hončariv and Jan Vinař

1. INTRODUCTION

The teaching of genetics at university level presents some rather special problems. The distinct logical structure of the subject and the statistical nature of its laws would compel us to count genetics among the physico-mathematical sciences rather than with the traditional biological disciplines. There is, however, one obstacle: in physics, for instance, the student can observe the working of basic laws by means of experiments; In mathematics, he can master the basic techniques by applying them to a great variety of problems. No such means are readily available for the student of genetics.

An object for genetic experiments must fulfil very strict conditions: it must be fruitful and short-lived; preferably of a higher type organism to make observation easier. One such an organism is the fruit fly Drosophila melanogaster. Practically all modern genetics owes its birth to this fly.

We see that the suitable objects are few. But if we ask for an object which is easy to manipulate, easy to observe and can produce some tens of generations of some hundreds of descendants each in a week (as compared with physical and chemical experiments which take about an hour) we come to the disappointing conclusion that „there is no such animal."

The usual solution has been to demonstrate the results of several experiments; to have a little practice in various techniques without the possibility of conducting meaningful experiments and stick to theory afterwards. Even so, the description of a genetic experiment (quite apart from the fact that it consists of the proverbial thousands of words which are hardly equivalent to a demonstration) tends to contain mostly technical details concerning the feeding and care of the animals, etc., leaving little room for the basics of experiment-planning which the student never learns.

The authors have been, for some time, working on a computer-based model of genetic processes which would teach the student just this. If he can get any number of descendants from two chosen animals (or plants) in several minutes by pushing a few buttons, he can afford to give due attention to the „strategy" of the experiment. The teacher can use his responses to evaluate not only his knowledge of the subject but also his general ability to make correct decisions in a complicated situation. In addition to this, the student is brought into close contact with the computer and gets at least some idea of what kind of help he can expect of it in his work.

2. THE MODEL

As the basis of our system we have chosen the model proposed in Neyman (1964). Since our object was stochastic **modelling** rather than statistical analysis, we felt free to make certain modifications in the model so as to incorporate certain features which were not contained in the **original** model. We shall sketch briefly the process of **formation** of new cells from the point of view of this **model.**

A new cell (it is to be remembered that any organism starts its life as one cell) arises from the combination of parts of two parent cells.

A cell (Fig. 2.1.) contains four chromosomes A, B, C, D, grouped into two **chromosome pairs** AB and CD. The two chromosomes of any pair are **homologous** with respect to each other. On every chromosome there are five **loci** (circles in Fig. 2.1.) in which **genes** may be located. Two genes which are located opposite each other in homological chromosomes form an **allelic pair**.

Any gene may be in one of two states (we speak of alles) it may be **dominant** or **recessive**. Let us represent a dominant allele by 1 (one) and recessive allele by 0 (zero). Now any property of the organism is influenced by one or more allelic pairs in the original cell. Let us consider the example of the pea: its pods may be either yellow (Y) or green (G). This property is determined by one allelic pair:

Table 2.1

1 st gene	2 nd gene	property
0	0	G
1	0	Y
0	1	Y
1	1	Y

We can express these relations also like this:

Table 2.2

1 st gene	2 nd gene	property	
		G	Y
0	0	1	0
1	0	0	1
0	1	0	1
1	1	0	1

Another example: the flowers of snapdragons may be white, red or pink. This trait is again determined by one allelic pair, thus:

Table 2.3

1 st gene	2 nd gene	R	W	P
0	0	0	1	0
1	0	0	0	1
1	1	0	0	1
1	1	1	0	0

After these examples we are prepared to accept the following statement: **any trait of the organism may be expressed as a (Boolean) function of 2n (Boolean) variables representing different alleles**

3. IMPLEMENTATION AND PERFORMANCE

3.1 *Programming*

It does not seem necessary to go deeply into the problems of actually programming this model. Even at this stage a competent programmer would be able to program the model himself.

A few details, however, might be helpful. The machine we used is the Soviet computer MINSK 22. We shall briefly recapitulate the relevant details:

- two-address instructions
- fixed word size of 36 bits plus a sign bit
- regretfully, no practical time sharing possibilities
- the input is from two tape readers and a typewriter, the output on two tape-punches a line-printer and again a typewriter. This typewriter is used for man-machine communication in this program. For a machine representation of the cell, it was decided to divide the machine word into 6 bytes of 6 bits each and to use the last four to represent chromosomes. The first bit of any byte is not significant.
- a detail which, it was found, greatly facilitates crossing over handling.

  The same representation was used for the gametes: one cell held both gametes with the first and third bytes (of the four) for one gamete and the rest for the other.

The actual process of hybridization was simulated as follows:

3.1.1 (One) A random number, uniformly distributed on the interval (0); 1. was generated (in the sequel we shall speak simply of random numbers) and the splitting probabilities for the one chromosome were subtracted from it successively until the first non-positive result. This determined the location of the split in the first chromosome pair.

The same operation was performed for the other chromosome pair and the crossing-over rearrangements were simulated.

3.1.2 The division into gametes now entailed only determining whether or not to interchange bytes 1 and 2 and 3 and 4. The probability of each interchange was 0,5.

3.1.3 The same process was repeated for the other parent cell.

3.1.4 The combination of two chosen gametes into a new cell now required only an optional six-bit left shift for the paternal gamete word, an optional six-bit right shift for the maternal gamete word, two masking operations and a logical sum. It is clear from the description that for one hybridization, eight random decisions were made.

3.1.5 For the new cell, 20 random decisions were made concerning mutations. Namely, if there was a „O" in any given bit, the decision was made on the basis of the „R to D" probability, otherwise the „D to R" probability was used. This completed the process of hybridization.

3.1.6 Every new cell was now searched for any sublethal gene combinations listed in a special table. A fixed „survival" probability" was associated with each such combination. If a combination was found, a random decision was made as to whether to „liquidate" the new cell. Thus, a new cell had to pass several „traps" of varying danger.

3.1.7 If the cell passed this test, it was searched for gene combinations listed in another table, which signified the presence of certain properties. If a property was found, it was displayed and a special „group counter" assigned to that property was incremented by one. This ended the process.

3.1.8 If the cell did not pass the tests of step 6, it was „liquidated" and the process started again with step 1. Thus one „hybridization" always produced one „live" cell and perhaps several „abortions". In all, even in this simple model, at least 28 random decisions are made in each hybridization. This accounts for the fact that in spite of the inherent simplicity of the genetic mechanisms the individual results are practically unpredictable.

This program forms the nucleus of the whole model. Around this, various programs were built to insure the performance described in 3.2. The program was written in an autocode commonly used for the machine. However, to ensure efficiency in logical operations and index register use, steps 3.1.1-3.1.5 were written in machine code.

3.2 *Performance*

In order to come as close as possible to natural conditions, we must try to give the student as much information as the actual experimenter has, but make „looking inside the mechanism" difficult. This means thar ordinarily the student is able to watch the process of hybridization to know all its properties (or, in genetic terms, its phenotype), but not the actual gene combination (or genotype). Thus for the student the visible outcome of one hybridization cycle will be a list of properties of the new individual (this appears on the typewriter).

However, to make possible a deeper understanding of the process, a „genetic protocol" is printed in parallel on the line printer. In this protocol, a detailed graphic representation of every hybridization cycle is given, similar to that of fig. 3.2. Moreover, if the new cell is an „abortion", it is also represented, but the „properties" line is filled with crosses. On the other hand, these „abortive cycles" are not represented on the typewriter in any way, since they are not usually readily apparent to the experimenter.

In addition to the basic faculty of listing each new organism, the system has various additional features which are listed here.

3.2.1. The number of descendants which are to be produced is specified by the student.

3.2.2. There are several ways of choosing parents for the next generation.

a) Their genotypes can be read in from one of the tape readers. This is used in two ways: for specifying the original parents as a part of problem statement and for providing „control lines". i.e., parents of a specified genotype for a testing purpose.

b) Their genotype may be set from the typewriter, i.e., by the student. The use of this should be restricted.

c) The listing of properties of a new descendant on the typewriter is followed by a query „selected as parent no.": As a reply to this, the student writes „1" if he wants the individual in question used as a father in the next generation, „2" for mother and „0" for exclusion from breeding. The prospective parents are put into a „paternal stack" and a „maternal stack" each of which holds 20 entries. From these stacks,

either of both parents may be selected at random.

d) It is also possible to repeat the hybridization with the same parents as before.

3.2.3. After the desired number of descendants has been produced, the machine types out the number of occurrences of all properties listed in the model.

3.2.4. For statistical purposes it is often necessary to deal with large numbers of descendants. As the listing of each separate descendant on the typewriter, and even more so on the line-printer, slows the whole process, it can be shut out by changing the switch setting on the control-panel. This is a departure form our principle of exclusive type writer control and will probably be changed in later versions.

3.2.5. Statistical tests can be performed on the results of hybridization. At present, only the chi-square test, commonly used in biological research, is used. Other tests will be added as their need is felt.

For the record of a typical student-machine conversation, we refer the reader to Appendix 2. To facilitate understanding, we remark that in the autocode used, integers from 1 to 400 can be used as labels for statements. Aside from normal jump instructions, jumps can be specified in input data. Namely, if an integer is followed by a left bracket, an input instruction results in a jump to the corresponding label, while the value of the variable specified in the instruction remains unchanged.

We should also like to point out that both appendices are records of actual texts (in Appendix 1, of input data and in Appendix 2 of an actual conversation) with the exception of comments in brackets which have been added by the authors.

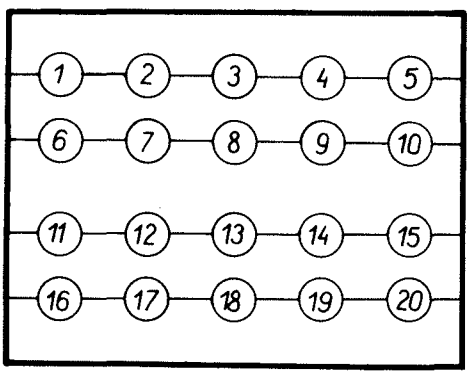
4. PRESENT AND FUTURE POSSIBILITIES

In spite of its small size (as compared to the Drosophila with its 4 chromosome pairs and some 10,000 genes, or man, whose 23 chromosome pairs are estimated to contain more than 50,000 genes) the behaviour of the model is at times quite complicated and exhibits most of the traits investigated in classical genetics. Of course it is not necessary to try to model only real organisms. The investigation of genetics of dragons, for instance, might be quite interesting for the student. The main task for the present seems to be the coding and testing of the largest possible number of useful models. In addition, some modifications will probably have to be made in the program itself: for instance, new routines for statistical testing will be added. One serious limitation of the system is that it operates on a machine with no time sharing and no remote terminals. This means the student has to come to the computing centre and occupy the whole machine for some time.

The authors would be very interested in helping anybody to put a similar system into operation on machine without these limitations.

REFERENCES

NEYMAN, J. 1964: First course in probability and statistics, Holt, Rinehart and Winston, Inc., NEW YORK, ch. 3, and 3.1



APPENDIX 1

/ A TYPICAL GENETIC MODEL /

/STATEMENT OF PROBLEM/

THE PODS OF THE VARIETY OF PEA WITH WHICH THIS EXAMPLE IS CONCERNED MAY BE GREEN(G) OR YELLOW(Y).THE SEEDS MAY HAVE TWO DIFFERENT SHAPES:ROUND (R) OR WRINKLED(D),WE SHALL BE ALSO INTERESTED IN THE COMBINATIONS OF THESE PROPERTIES.WE_HAVE_YR AND GG_THE PARENTS OF THE FIRST GENERATION ARE TWO PLANTS OF THE TYPES YR AND GG,YOU ARE TO DETERMINE THE MECHANISM OF HEREDITY.

/SPLITTING PROBABILITIES/

0,00 0,20 0,10 0,05 0,65 0,00 0,30 0,40 0,20 0,10 10$(

/PROBABILITIES FOR D→R MUTATIONS/

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10$(

/PROBABILITIES OF R→D MUTATIONS/

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 40(

40(/THIS IS END OF TABLE OF SUBLETHAL PROPERTIES THE TABLE IS EMPTY/

11 16 25(35(R,0 1
11 25(16 27(R,0 1
16 25(11 27(R,0 1
25(11 16 27(R,0 1
2 7 25(35(Y,0 3
2 25(7 27(Y,0 3
7 25(2 27(Y,0 3
25(2 7 27(G,0 4
16 7 25(11 2 27(YR,0 5
2 16 25(11 7 27(YR,0 5
2 16 7 25(11 27(YR,0 5
11 7 25(2 16 27(YR,0 5
11 2 25(16 7 27(YR,0 5
11 2 7 25(16 27(YR,0 5
11 16 7 25(2 27(YR,0 5
11 2 16 25(7 27(YR,0 5
11 2 16 7 25(35(YR,0 5
2 7 25(11 16 27(YG,0 6
7 25(11 2 16 27(YG,0 6
2 25(11 16 7 27(YG,0 6
16 25(11 2 7 27(YR,0 7
11 25(16 2 7 27(YR,0 7
11 16 25(2 7 27(YR,0 7
25(11 2 16 7 27(YR,0 8
28(

/INTERPRETATION TABLE: THE FIRST LINE IS INTERPRETED THUS:
IF THE ALLELES 11 AND 16 ARE DOMINANT, THEN-REGARDLESS OF WHICH ALLELES ARE RECESSIVE/ THE INDIVIDUAL HAS THE TRAIT R AND BELONGS TO GROUP 1/

1 R0
2 Y0
3 T0
4 G0
5 TR0
6 YG0
7 RR0
8 GG0
92

/TABLE OF GROUP NAMES/

A P P E N D I X 2 :

/A TYPICAL STUDENT—MACHINE CONVERSATION/

/THE STUDENT CONTRIBUTIONS ARE UNDERLINED/

THE PODS OF THE VARIETY OF PEA WITH WHICH THIS EXAMPLE IS CONCERNED MAY BE GREEN(G) OR YELLOW(Y).THE SEEDS MAY HAVE TWO DIFFERENT SHAPES:ROUND (R) OR SQUARE(S).WE SHALL BE ALSO INTERESTED IN THE COMBINATIONS OF THESE PROPERTIES:GR,GS,YR AND YS.THE PARENTS OF THE FIRST GENERATION ARE TWO PLANTS OF THE TYPES YR AND GS.YOU ARE TO DETERMINE THE MECHANISM OF HEREDITY.

1[/THIS MEANS THAT PARENTS ARE TO BE SELECTED FROM INPUT/
94[/START HYBRIDIZATION/
FATHER:
S,G,GS,
MOTHER:
R,Y,YR,

NUMBER OF DESCENDANTS:5

PROPERTIES:
R,Y,YR,

SELECTED AS PARENT NO.:1
PROPERTIES:
R,Y,YR,

SELECTED AS PARENT NO.:2
PROPERTIES:
R,Y,YR,

SELECTED AS PARENT NO.:1
PROPERTIES:
R,Y,YR,

SELECTED AS PARENT NO.:2
PROPERTIES:
R,Y,YR,

SELECTED AT PARENT NO.:0

136[- CHI-SQUARE TEST /THE STUDENT CALLS THE CHI-SQUARE PROGRAM
1 1 AND TESTS WHETHER THE ACTUAL FREQUENCIES
2 1 OF GROUPS 1 AND 2 CONFORM TO THE THEORETICAL
133[CHI-SQUARE-TEST 3 : 1 RATIO/
- - - - - - - - -

GROUP NO:	FREQUENCY	EXPECTED	ACTUALLY	DEVIATION
1	3	150,0	152	2,00
2	1	50,00	48	-2,00

CHI-SQUARE= ,1067
THE DIFFERENCE IS NOT SIGNIFICANT

136[- CHI-SQUARE TEST /THE SAME FOR GROUPS 3 AND 4/
3 1
4 1
133[
CHI-SQUARE-TEST
- - - - - - - - -

GROUP NO.	FREQUENCY	EXPECTED	ACTUALLY	DEVIATION
3	3	150,0	148	-2,00
4	1	50,00	52	2,00

CHI-SQUARE= ,1067
THE DIFFERENCE IS NOT SIGNIFICANT

136[- CHI-SQUARE TEST /THIS TEST IS TO SHOW WHETHER THE TWO PROPERTIES
5 3 ARE INDEPENDENT; THIS WOULD INDICATE THAT THE
6 3 CORRESPONDING ALLELIC PAIRS ARE ON DIFFERENT
7 3 CHROMOSOME PAIRS/
8 1
133[CHI-SQUARE-TEST
- - - - - - - - -

GROUP NO.	FREQUENCY	EXPECTED	ACTUALLY	DEVIATION
5	9	112,5	115	2,50
6	3	37,50	33	-4,50
7	3	37,50	37	-,500
8	1	12,50	15	2,50

CHI-SQUARE= 1,102
THE DIFFERENCE IS NOT SIGNIFICANT

131[

1 R 5 /THIS IS THE LIST OF FREQUENCIES OF VARIOUS
2 S 0 PROPERTIES IN THE OFFSPRING SINCE THE
3 Y 5 PROPERTIES S AND G ARE NOT MANIFESTED,
4 G 0 THE STUDENT CONCLUDES THEY ARE RECESSIVE/
5 YR 5
6 YS 0
7 GR 0
8 GS 0

125[- FATHER SELECTED AT RANDOM /FATHER/NO. 1/ AND MOTHER /NO.2/ ARE
127[- MOTHER SELECTED AT RANDOM NOW CHOSEN AT RANDOM FROM AMONG THOSE
135[-HYBRIDIZATION STARTED "MARKED" FOR FURTHER BREEDING/

FATHER: /THESE ARE PROPERTIES OF THE
R,Y,YR, CHOSEN PARENTS/
MOTHER:
R,Y,YR,

NUMBER OF DESCENDANTS:200
/THE SWITCHES HAVE BEEN RESET SO AS TO " SHUT OFF" THE LISTING
OF EACH DESCENDANT ON THE TYPEWRITER AS WELL AS THE GENETIC
PROTOCOL ON THE LINE PRINTER/

1 R 152 /THIS IS THE STRUCTURE OF THE SECOND FILIAL GENE-
2 S 48 RATION IF THE STUDENT'S ESTIMATE WAS CORRECT THEN
3 Y 148 THE DOMINANT PROPERTIES SHOULD OCCUR THREE TIMES
4 G 52 OFTEN. THAN THEIR RECESSIVE COUNTERPARTS/
5 YR 115
6 YS 33
7 GR 37
8 GS 15

AUTHORS INDEX

ABRAMOV, A.A. II/1
ADDERLEY, J. II/235
AIKEN, R.M. II/153, III/133
ALLISON, Chr.S. III/151
ANDERSEN, Chr. II/91
ANTIPOV, I.N. II/1
ARNOLD, R. II/3, 9
ARSAC, J. I/69
ASHENHURST, R.L. II/157
ASPINALL, D. II/161
AUSTING, R.H. II/167

BARNES, B. II/171
BARNETT, A. II/241
BARNETT, M.P. III/137
BELLIS, B.T. II/15
BELLISSANT, C. III/145
BENNETT, J.M. III/151
BENTIL, M.A. I/103
BERGER, A. II/21, III/155
BERRY, P.C. II/25
BESTOUGEFF, H. III/341
BLATNY, J. II/289
BLAXTER, C.W. II/95
BOHNERT, L.M. III/159
BOLLIET, L. II/175
BONHAM, G.C. II/33
BORDOGNA, J. III/1
BOWEN, C.R. II/99
BRODERICK, W.R. III/245, 43
BUCKINGHAM, R.A. I/19
BULL, G.M. II/177, III/47
BURDETT, D. III/51

CAPON, I.N. III/167
CARVER, R.J.T. II/395
CHARP, S. I/35
CHEVION, D. I/115
CHURCHILL, S. II/41
COADY, E.R. II/401
COLE, A.J. III/115
COLYER, A.G. III/71
CONWAY, D.E. II/47, III/171
COUGER, J.D. II/247
COWAN, D.D. II/33
COWIE, J.B. III/249

DARBY, C.A. II/51
DAVIS, G.B. II/253
DESSAU, E. II/293
DI GIORGIO, G.V. III/187
DOMBAL, F.T. de III/77
DONOVAN, J.J. II/183
DOUGLAS, D. II/257
DUTTON, J.B. II/185
DUTTON, P.E. II/57

EMANUELS, H.B. II/441

FALKOFF, A.D. II/25
FARRINGDON, M.G. II/57
FEREIRA, O.M. II/223
FENICHEL, R.R. III/175
FINDLER, N.V. II/191
FINERMAN, A. II/195
FOSTER, J. III/111

GILLIGAN, J. III/71
GOTTERER, M.H. II/171
GRAM, Chr. II/409
GRANT, A.H.M. II/395
GROCHLA, E. II/105
GROSHEIDE, J. I/23

HALL, A.W. II/415
HAMBLEN, J.W. II/205
HAMMER, P.C. I/65
HARPER, D.H. III/259
HART, J.F. III/7
HART, R. II/217
HEBENSTREIT, J. I/105, II/329
HIGHLAND, N.J. II/323
HIRANO, J. III/221
HONCARIV, R. III/265
HOWSON, Chr.J. III/79
HSIAO, D.K. III/1
HUTCHINSON, M.G.P. II/59

IVERSON, K.E. II/25

JACKSON, H.L.M. II/67
JOHNSON, B. II/423

KADAR, I. II/297
KAHN, A.B. III/13
KAHN, A.G. I/109
KLOOSTER, W. v.d. III/129

KOROTKIN, A.L. II/51
KOTAL, M. II/331
KUO, C. II/339
KUO, F.F. II/335
KURKI-SUONIO, R. II/343

LANGEFORS, B. I/29
LAVINGTON, S.H. III/85
LEE, M.H. II/71
LEWIN, D.W. II/347
LEWIS, J.W. II/429
LEWIS, R. II/77
LI, D.H. II/299

MAKINSON, G.J. III/179
MARGULIES, F. I/87, 107
MARLOW, H.G.B. II/263
MASCARENHAS, OLIVEIRA. S. II/223
MAURER, W.D. II/353
MAYDELL, U.M. von III/89
McAREAVY, J.F. II/269
McCLINTOCK, S. III/235
McKAY, G.C. II/433
McLEAN, L.D. III/183
McLEAN, R.F. II/339
McLEAN, R.S. III/183
MEYER, L.F. II/303
MILBURN, R.G. III/199
MILLIN, D. III/37
MILLER, I.G.T. II/79
MIOLA, A. III/187
MIURA, T. III/221
MORRIS, C. III/37
MOSKOV, A.T. III/97
MOSKOVA, T.T. III/97
MUNN, R.J. III/205
MURPHY, B.M. III/19

NIELEN, G.C. II/357

ODEH, R.E. II/359
OED, G.V. II/275
OPPENHEIMER, A. III/71

ORGANICK, E. II/433
OWENS, J.D. II/363

PAPERT, S. I/73
PARDO, M. II/305
PASK, G. I/41
PEGG, D. I/83
PEGG, P. II/57
PENNY, J.P. II/185
PENNY, R. II/9
PEREIRA DE LUCENA, C.J. II/369
PESTI, L. II/297
PINATTI, D. II/223
POEL, W.L. van der III/209
PRYWES, N.S. III/1

RABEN, J. II/225
RALSTON, A. II/101
RIDING, G. III/215
RIGAL, J.L. II/375,81
ROBERTS, F.D.K. II/359
ROHL, J.S. III/85
ROMANO, A. III/23
RON-ELL, A.H. II/313
ROWE, B.D. II/435
RUBINOFF, M. II/377, III/29

SACERDOTI, G. II/117
SAMPSON, H.L. III/103
SANDBOURGH, R.E. III/199
SCHEEPMAKER, B. I/21, 119, II/357, 441
SCHILDKRET, A. III/51
SCHINKEL, A. II/123
SCORER, J.W. III/111
SCRIVENS, B. II/3
SEFTON, C.H. II/383
SEIBT, D. II/105
SELVIN, H.C. III/259
SHNEIDERMAN, B. II/323
SIMENSEN, C.J. II/445
SLAMECKA, V. II/229
SLEEMAN, D.H. III/115

SLEMON, A. III/7
SMITH, R.W. III/31
SPROWLS, R.C. III/119
STAINES, B.R.J. II/451
SUTTIE, I.P. III/7
SVEISTRUP, P. II/281
SZYPERSKI, N. II/105

TAKAYANAGI, A. II/131
TERRIBILE, A. III/187
TIENARY, M. II/139
TINSLEY, J.D. II/85
TOMASSO, C. II/79
TSUDA, J. III/221
TULLY, C.J. II/285

URA, S. II/131

VALETTE, A. III/241
VEER, G. van der I/79
VEJSOVA, A. II/319
VERMIJS, A.J.A. III/227
VINAR, J. III/265
VINSONHALER, J.F. III/37
VOLLEBERGH, J.J.A. I/97
VOS, M. de III/241
VRIES, J.G. de III/129

WALKER, T.M. II/389
WELDON, E.J. II/335
WENTE, K.W. II/459
WEST, A.S. III/123
WHITE, T.C. II/141
WILKINS, B.R. II/347
WILLIAMS, N. II/41
WILSON, B.J. II/57, 463
WINDER, C.L. III/37
WOLFF, K. III/205
WRIGHT, A.H. II/149

ZEMANEK, H. I/25
ZEMOR, P. II/469
ZINN, K.L. I/21, 119 III/235
ZUNDE, P. II/229

LIST OF AUTHORS AND ADDRESSES

Abramov, A.A. Computing Centre,
Academy of Sciences of the U.S.S.R.,
Vavilovastra 40,
Moscow B 333 U.S.S.R.

Adderley, J. The National Computing Centre
Quay House, Quay Street,
Manchester, U.K.

Aiken, R.M. The University of Tennessee.
Computing Center.
Knoxville Tennessee 37916, U.S.A.

Allison, Chr.S. Basser Computer Department,
University of Sydney.
Sydney. New South Wales 2006,
Australia.

Andersen, Chr. The Danish EDP Council.
I, Vesterbrogade.
1620 Copenhagen V. Denmark

Antipov, I.N. Computing Center,
Academy of Sciences of the U.S.S.R.
Vavilovastra 40
Moscow B 333. U.S.S.R.

Arnold, R. INDICOM Project.
Waterford Township School District
1325 Crescent Lake Road.
Pontiac. Michigan 48054, U.S.A.

Arsac, J. Université de Paris.
Faculté des Sciences.
Institut de Programmation.
Tour 42. Quai St. Bernard.
Paris 5. France.

Ashenhurst, R.L. The University of Chicago.
Committee on Information Sciences.
Chicago. Illinois 60637, U.S.A.

Aspinall, D. Department of Computer Science.
The University.
Manchester 13. U.K.

Austing, R.H. Computer Science Center.
University of Maryland,
College Park, Maryland 20740, U.S.A.

Barnes, B. The Pennsylvania State University.
College of Science Department
426 McAllister Building.
University Park, Pa. 16802, U.S.A.

Barnett, A. Barnett Data Systems
Suite 507. 1010 Rockville Pike.
Rockville, Maryland 20852, U.S.A.

Barnett, M.P. The H.W. Wilson Company,
950 University Avenue,
Bronx, New York 10452, U.S.A.

Barrit, M.M. Edinburgh Regional Computer Center
Kings Building
Mayfield Road
Edinburgh EH 9-372 – U.K.

Bellis, B.T. Edinburgh Merchant Company
Schools.
Daniel Stewart's College.
Edinburgh EH4 3EZ U.K.

Bellisant, C. Université de Grenoble.
Mathématiques Appliquées.
Cedex 52, 38 Grenoble-Gare. France

Bennett, J.M. The Basser Computing Dept.
School of Physics.
University of Sydney.
Sydney, New South Wales 2006,
Australia.

Bentil, M.A. Ghana Institute of Management and
Public Administration Greenhill,
P.O.B. 50
Achimota, Ghana

Berger, A. Abteilung für Bürotechnik.
Feldmühlgasse II,
A-1130 Vienna, Austria

Berry, P.C. IBM Thomas J.Watson Research Center.
P.O.Box 218.
Yorktown Heights,
New York 10598, U.S.A.

Bestougeff, H. Faculté des Sciences de Paris.
Ordinateur pour Etudiants. Tour 13,
9, Quai St. Bernard.
75 Paris V, France

Blatny, J. Department of Computers,
Technical University,
Bezetechova 2,
Brno, C.S.S.R.

Blaxter, C.W. The National Computing Centre,
Quay House, Quay Street,
Manchester. U.K.

Bohnert, L.M. State University of New York at
Albany.
School of Library Science.
1400 Washington Avenue,
Albany, New York 12203, U.S.A.

Bolliet, L. Mathémathiques Appliquées Informatique.
Université de Grenoble.
Cedex 53.
38 Grenoble-Gare. France.

Bonham, G.C. Ontario Dept. of Education.
44 Eglinton Ave.
West Toronto 310,
Ontario, Canada

Bordogna, J. The Moore School of Electrical Engineering,
University of Pennsylvania.
Philadelphia, Penn. 19104, U.S.A.

Bowen, C.R. IBM.
Old Orchar Rd.
Armonk, New York 10504, U.S.A.

Broderick, W.R. The Royal Liberty School.
Computer Dept.
Hare Hall.
Upper Brentwood Rd.
Romford, Essex, U.K.

Buckingham, R.C. Institute of Computer Sciences
44, Gordon Square
London, W.C. I, U.K.

Bull, G.M.I The Hatfield Polytechnic Computer Science,
Hatfield, Herts. U.K.

Burdett, D.	Hostos Community of the City University of New York. 260 East, 161 Street. Bronx, New York 10451. U.S.A.
Canfora, A.	IBM Italia Spa. Via Pirelli 18. 20124 Milano. Italy
Capon, I.N.	Department of Computing Science. The University of Adelaide. Adelaide. South Australia 5001.
Carver, R.J.T.	Master Register Ltd. Third Floor. Royal Buildings. 2, Mosley St. Manchester. U.K.
Charp, S.	Divison of Instructional Systems. The School District of Philadelphia. 5th and Luzerne Streets. Philadelphia, Penna 19140, U.S.A.
Chevion, D.	Office Mechanisation Centre Postbox 13016 Jerusalem. Israël
Churchill, S.	Dept. of Computer Applications. The Ontario istitute for Studies in Education. 102 Bloor Street W. Toronto 5, Ontario, Canada.
Coady, E.R.	Department of Health, Education and Welfare. Social Security Administration Data Processing and Accounts. Room 2AII. 6401 Security Blvd. Baltimore, Maryland 21235, U.S.A.
Colle, A.J.	Dept. of Education. Research Council's Computer Based Learning Project. The University of Leeds, U.K.
Colyer, A.G.	ICL Education Research. Landwater House. High Wycombe, Bucks, U.K.
Conway, D.E.	North Staffordshire Polytechnic. Computing Dept. Stafford. U.K.
Couger, J.D.	University of Colorado. School of Business. Colorado Springs Center. Cragmor Rd. Colorado Springs. Colorado 80907. U.S.A.
Cowan, J.D.	University of Waterloo. Dept. of Applied Analysis and Computer Science. Waterloo, Ontario, Canada
Cowie, J.B.	Naval Postgraduate School Dept. of Business Administration and Economics. Monterey, California 93940, U.S.A.
Darby, C.A.	American Institutes for Research. 8555 Sixteenth Street. Silver Spring, Maryland 20910. U.S.A.
Davis, G.B.	Management Information Systems Research Center. University of Minnesota. Minneapolis, Minnesota 55108. U.S.A.
Dessau, E.	Computing Research Centre. United Nations D.P. Bratislava 2. Czechoslovakia
Di Giorgio, G.V.	Centro di Calcolo Interfacolta, Universita di Roma. Roma. Italy
Dombal, F.T. de	University Dept. of Surgery. The General Infirmary. Leeds I. U.K.
Donovan, J.J.	Massachusetts Institute of Technology. Project MAC 545. Technology Square, Cambridge, Mass. 02139. U.S.A.
Douglas, D.	Dept. of Adult Education. Sydney University Extension Board. Sydney 2006. Australia.
Dutton, J.B.	University of Alberta. Dept. of Computing Science. Edmonton, Alberta. Canada
Dutton, P.E.	Danum Grammar School for Boys. Armthorpe Rd. Doncaster. U.K.
Emanuels, H.B.	Centre for Development of Automation in Municipalities. Oude Molstraat 34, The Hague. Netherlands
Falkoff, A.D.	IBM Thomas J.Watson Research Center. P.O.Box 218. Yorktown Heights. New York 10598. U.S.A.
Farringdon, M.G.	Dept. of Computer Science Computer Centre, University College of Swansea. Singleton Park, Swansea, Glam. U.K.
Fereira, O.M.	Faculty of Education and Educational Technology. Federal University. San Carlos. Sao Paulo. Brazil
Fenichel, R.R.	Massachusetts Institute of Technology. Project MAC 545 Technology Square. Cambridge, Mass. 02139. U.S.A.
Findler, N.V.	Dept. of Computer Science, State University of New York at Buffalo. 4250 Ridge Lea Rd. Amherst. New York U.S.A.
Finerman, A	Department of Computer Science. Faculty of Engineering and Applied Sciences. State University of New York at Buffalo. 4250 Ridge Lea Rd. Buffalo. New York, 14226. U.S.A.
Foster, J.	Mod Navy. HMS, Colingwood. Fareham, Hants. U.K.
Galigan, J.	ICL Education Research Landwater House High Wycombe, Bucks. U.K.

Gotterer, M.H. The Pennsylvania State University.
College of Science .
426 McAllister Building.
University Park, Pa. 16802. U.S.A.

Gram, Chr. The Danish E.D.P. Council.
1, Vesterbrogade.
1620 Copenhagen-V. Denmark.

Grant, A.H.M. ICL Central Management Services.
P.O. Box 3. Folshow Hall.
Wimslow,
Cheshire. U.K.

Grochla, E Betriebswirtschaftliches Institut für Organisation und Automation an der Universität zu Köln
5, Köln-Lindenthal
45 Universitätsstrasse.
Germany.

Hall, A.W. Capital Computer Applications Ltd.
Audley House.
10 Margaret Street.
London W.1 U.K.

Hamblen, J.W. Southern Regional Education Board.
130 Sixth Street NW,
Atlanta, Georgia 30313. U.S.A.

Hammer, P.C. College of Science.
Computer Science Dept.
The Pennsylvania State University
426 McCallister Building
University Park. Penns, 16802. U.S.A.

Harper, D.H. The University of Rochester.
River Campus. Dept. of Sociology.
Rochester, New York 14627. U.S.A.

Hart, J.F. The University of Western Ontario.
Faculty of Science.
Dept. of Computer Science.
Staging Building
London, Canada.

Hart, R. New College of Hofstra.
Hampstead.
New York 11550. U.S.A.

Hebenstreit, J. Ecole Supérieure d'Electricité.
Université de Paris.
Computer Science Dept.
10 Avenue Pierre Larousse.
92 Malakoff. France.

Highland, N.J. United States Educational Foundation in Finland,
Salomonkatu 17-A.
Helsinki, Finland.

Hirano, J. Hitachi Central Research Laboratory.
6th Dept. Higashi-Koigabuko.
Kukubunji,
Tokyo, Japan.

Honcariv, R. Safarik University.
Computing Center.
Namestie Fabruareve vitazstva 9.
Kosice. C.S.S.R.

Howson, Chr.J. SPL International.
Real Time Division.
33/34 Alfred Place.
Tottenham Court Rd.
London WC1. U.K.

Hsiao, D.K. The University of Pennsylvania.
The Moore School of Electrical Engineering.
Philadelphia. Pa. 19104. U.S.A.

Hutchinson, M.G.P. HREOD Burma School.
Akers way Mordedon.
Swindon Wirlshire. U.K.

Iverson, K.E. IBM Thomas J.Watson Research Center.
P.O. Box 218.
Yorktown Heights, New York 10598, U.S.A.

Jackson, H.L.M. North Staffordshire Polytechnic.
Faculty of Mathematics, Statistics and Computing.
Stafford U.K.

Johnson, B. SPL International.
Medical Services Division.
75 Grosvenor Street.
London W1 U.K.

Kadar, I. John V. Neumann Society for Computing Sciences.
Budapest V.Szabadsag ter 17.
Hungary.

Kahn, A.B. The University of Wisconsin.
Computer Sciences Dept.
1210 West Dayton Street.
Madison. Wisconsin 53706. U.S.A.

Khan, A.G. O.E.C.D.,
2. Rue André Pascal
Paris, France.

Klooster, W. v.d. Philips N.V.
Educational Research OA-4.
Kastanjelaan 1.
Eindhoven. Netherlands.

Korotkin, A.L. American Institutes for Research.
8555 Sixteenth Street.
Silver Spring. Maryland 20910. U.S.A.

Kotal, M. Ministry of Education.
Computer Education.
Praha 1. Mala Strana.
Karmeliska 8. C.S.S.R.

Kuo, C. The University of Strathclyde.
Dept. of Naval Architecture Livingstone Tower.
Richmond Street,
Glasgow C.1. U.K.

Kuo, F.F. Dept. of Electrical Engineering
The University of Hawaii 25-65
The Mall. Honolulu.
Hawaii 96822. U.S.A.

Kurki-Suonio, R. University of Tampere.
Kalevantie 4
Tampere. Finland,

Langefors, B. Royal Institute of Technology.
Department of Administrative Information Processing.
Lindstedtsvägen 15.
100 44 Stockholm. Sweden.

Lavington, S.H. Department of Computer Science.
The University
Manchester 13. U.K.

Lee, M.H. City of Leicester Polytechnic
School of Mathematics.
Computing and Statistics.
P.O. Box 143.
Leicester, U.K.

Lewin, D.W. Department of Electronics.
The University.
Southampton, N.H., U.K.

Lewis, J.W.
The Hatfield Polytechnic.
Computer Science Department.
Hartfield. Hertfordshire. U.K.

Lewis, R.
Chelsea College of Science and Technology.
University of London.
Centre for Science Education.
Bridges Place,
London S.W., 6. U.K.

Li, D.H.
School of Business Administration.
Department of Accounting.
University of Washington.
Seattle, Washington 98105. U.S.A.

Makinson, G.J.
The University of Liverpool.
Department of Computational
and Statistical Science.
Victoria Building. Brownlow Hill.
Liverpool U.K.

Margulies, F.
Graf Starhemberggasse 13/ 9/ 14.
A-1040. Wien. Austria.

Marlow, H.G.B.
Associated Industrial Consult Ltd.
Old Court House. Old Court Place.
Kensington, Londen W.8, U.K.

Mascarenhas,
Oliveira. S.
Universidade de Sao Paolo.
Escola de Engenharia de Sao Carlos.
Ave. Dr. Carlos
Botelho 1465.
Sao Carlos. Sao Paolo. Brazil.

Maurer, W.D.
University of California.
211 Cory Hall,
Berkeley, California 94770. U.S.A.

Maydell, U.M. von
University of Alberta,
Dep. of Computing Science.
Edmonton, Alberta, Canada.

McAreavy, J.F.
Department of the Army
U.S. Army Management Engineering
Training Agency,
Rock Island, Illinois 61201, U.S.A.

McKay, G.C.
University of Houston.
Department of Computer Science.
Cullen Blvd.
Houston, Texas 77004, U.S.A.

McLean, L.D.
The Ontario Institute for Studies in Education.
Dep. of Computer Applications.
102 Bloor Street West.
Toronto 5, Ontario, Canada.

McLean, R.F.
University of Strathclyde
Mechanical Engineering Group.
Montrose Street.
Glasgow C, 1, U.K.

McLean, R.S.
The Ontario Institute for Studies in Education,
Department of Computer Applications.
102 Bloor Street West.
Toronto 5, Ontario, Canada.

Meyer, L.F.
Universidad Nacional de Asunción .
Instituto de Ciencas.
Centro Nacional de Computatión.
Avenida Espana 1098.
Asunción. Paraguay.

Milburn, R.G.
Univac Division. Sperry Rand Corporation.
Burnsville, Minneapolis 55378, U.S.A.

Millin, D.
309 Computer Center.
Michigan State University.
East Lansing Michigan, U.S.A.

Miller, I.G.T.
University of Strathclyde
Dundasvale Computer Centre
George Street
Glasgow, U.K.

Miola, A.
Instituto Nazionale per le Applicazioni del Calcolo.
Consiglio Nazionale delle Ricerche.
Piazzale delle Scienze 7.
00185 Roma, Italy.

Miura, T.
Central Research Laboratory
Hitachi Limitad
Kokubunji
Tokyo. Japan.

Morris, C.
Michigan State University.
College of Communications.
East Lansing. Michigan. 48823, U.S.A.

Moskov, A.T.
High Institute of Electrical and Mechanical Engineering.
1, 19 February street,
Sofia, Bulgaria.

Moskova, T.T.
C.I. Project.
6 A Kamenodelska Street.
Sofia, Bulgaria.

Munn, R.J.
University of Maryland,
Institute for Molecular Physics.
College of Arts and Sciences.
College Park, Maryland 20742, U.S.A.

Murphy, B.M.
I C L house,
Putney,
London S.W. 15 U.K.

Nielen, G.C.
Philips' Gloeilampenfabrieken,
I S A-A.
Eindhoven, Neth.

Odeh, R.E.
Department of Mathematics.
University of Victoria. Victoria.
British Columbia, Canada.

Oed, G.V.
Victoria University of Wellington.
Department of Accountancy.
P.O. Box 196.
Wellington, New Zealand.

Oppenheimer, A.
I C L Education Research,
Landwater House,
High Wycombe, Bucks, U.K.

Organick, E.
University of Houston.
Department of Computer Science.
Cullen Boulevard.
Houston, Texas 77004, U.S.A.

Owens, J.D.
Staten Island Community College.
The City University of New York
715 Ocean Terrace.
Staten Island, New York 10301, U.S.A.

Papert, S.
Massachusetts Institute of Technology.
Department of Mathematics and Artificial Intelligence Laboratory.
Canbridge 39, Massachusetts, U.S.A.

Pardo, M.
Empressa de Servicio de Computacion.
Training Division.
Huerfanos 1374, 8 y 9 pisos.
15 Santiago, Chile.

Pask, G. Brunel University. Institute of Cybernetics.
Kingston Lane.
Uxbridge, Middlesex, U.K.

Peggy, D. Bedford School,
Bedford U.K.

Penny, J.P. Department of Computing Science.
University of Alberta.
Edmonton, Alberta, Canada.

Penny, R. INDICOM Project.
Waterford Township School District.
1325 Crescent Lake Road,
Pontiac, Michigan 48054, U.S.A.

Pereira de Lucena, C.J. Computer Science Department.
Pontificia Universidade Catolica do Rio de Janeiro.
Rue Marques de Sao Vicente 209/263.
Rio de Janeiro, Brazil.

Pesti, L. John V. Neumann Society for Computing Sciences.
Budapest V.
Szabadsag Ter 17. Hungary.

Pinatti, D. Universidade de Sao Paolo.
Centro de Processamento de Dados.
Avenida Dr. Carlos Botelho 1465.
Sao Paolo, Brazil.

Poel, W.L. van der University for Technological Sciences.
Delft, The Neth.

Prywes, N.S. The Moore School of Electrical Engineering.
University of Pennsylvania.
Philadelphia 19104, U.S.A.

Raben, J. Queens College of the City University of New York.
Flushing, New York 11367, U.S.A.

Ralston, A. State University of New York at Buffalo.
Department of Computer Science.
4250 Ridge Lea Road.
Buffalo, N.Y. 14226, U.S.

Riding, G. Department of Computer Science.
The University.
Manchester 13, U.K.

Rigal, J.L. U E R Informatique de Gestion.
Université de Paris-Dauphine.
Place du Maréchal de Lattre de Tassigny.
75 Paris 16, France.

Roberts, F.D.K. Department of Mathematics.
University of Victoria.
Victoria, British Columbia, Canada.

Rohl, J.S. Department of Computer Science.
The University.
Manchester 13, U.K.

Romano, A. Instituto di Fisica.
Universitá degli Studi di Bari.
Via Amendola 173.
Bari, Italy.

Ron-Ell, A.H. The Israel Institute of Productivity.
Tel Aviv.
4 Henrietta Szold Street. Israel.

Rowe, B.D. The National Computing Centre Ltd.,
Quay House.
Quay Street,
Manchester 3, U.K.

Rubinoff, M. University of Pennsylvania.
The Moore School of Engineering
Philadelphia, Pa. 19104, U.S.A.

Sacerdotti, G. Olivetti Training Department,
Via Jervis 77.
10015 Ivrea. Italy.

Sampson, H.L. Madison Public Schools.
Management Information Services.
P.O. Box 2189. 545 West Street,
Madison, Wisconsin 53703. U.S.A.

Sandbourgh, R.E. Univac Division. Sperry Rand Corp.
Burnsville, Minneapolis 55378, U.S.A.

Scheepmaker, B. Department of Organisation and Automation.
Joint Institute for Applied Psychology.
Berg en Dalseweg 127.
Nijmegen. Netherlands.

Schildkret, A. Hostos Community College of the City University of New York.
260 East, 161 Street.
Bronx, New York 10451, U.S.A.

Schinkel, A. S.O.A.G.
Paleisstraat 5,
The Hague, Netherlands.

Scorer, J.W. Mod (Navy) H M S Collingwood.
Fareham, Hants, U.K.

Scrivens, B. Project INDICOM.
Waterford Township School District.
1325 Crescent Lake Road.
Pontiac, Michigan 48054, U.S.A.

Sefton, C.H. Department of Mathematics.
Woolwich Polytechnic.
Wellington Street.
London S.E. 18, U.K.

Seibt, D. Betreibswirtschaftliches Institut für Organisation und Automation andder Universität zu Köln.
5 Köln-Lindenthal,
Universitätstrasse 45, Germany.

Selvin, H.C. State University of New York.
Department of Sociology.
Stony Brook, New York, U.S.A.

Shneiderman, B. State University.
Agricultural and Technical College at Farmingdale.
Data Processing Department.
New York 11735, U.S.A.

Simensen, C.J. Univac Data Processing Division
Sperry Rand Corporation.
P.O. Box 8100.
Philadelphia, Penna 19191, U.S.A.

Slamecka, V. Georgia Institute of Technology.
School of Information and Computer Science.
Atlanta, Georgie 30332, U.S.A.

Sleeman, D.H. Department of Education.
Research Council's Computer Based Learning Project.
The University of Leeds,
Leeds, U.K.

Slemon, A. Althouse College of Education
University of Western Ontario,
London, Canada.

Smith, R.W. University of California.
Santa Barbara, California 93106, U.S.A.

Sprowls, R.C. University of California,
Los Angeles Graduate School of Business Administration,
Los Angeles, California 90024, U.S.A.

Staines, B.R.J. Enfield College of Technology,
Computing Centre.
Queensway.
Enfield, Middlesex, U.K.

Suttie, I.P. Faculty of Science.
Department of Computer Science.
Staging Building.
University of Western Ontario.
London, Canada.

Sveistrup, P. Regnecentralen.
Scandinavion Information Processing Systems.
2000 Copenhagen.
Finsensvej 9-1. Denmark.

Szyperski, N. Betriebswirtschaftliches Institut für Organisation und Automation an der Universität zu Köln.
5 Köln-Lindenthal.
Universitätsstrasse 45, Germany.

Takayanagi, A. Japan Information Processing
Dev. Center.
1-5 Shibakoen 21
Minato-Ku, Tokyo, Japan.

Terribile, A. Instituto Matematico.
Universitá di Roma.
Roma, Italy.

Tienary, M. Computer Science Department,
University of Helsinki.
Töolönkatu 11,
Helsinki 10, Finland.

Tinsley, J.D. The National Computing Centre Ltd.
Quay Street.
Manchester 3, U.K.

Tomasso, C. University of Strathclyde.
Dundas Vale Computer Centre.
George Street.
Glasgow, U.K.

Tsuda, J. Central Research Laboratory.
Hitacho Limited.
Kokubunji, Tokyo, Japan.

Tully, C.J. London School of Economics and Software Sciences Ltd.
17 Curzon Street.
London W.1. U.K.

Ura, S. Faculty of Engineering. Japan Information Processing Development Center.
1-5 Shibakon 21.
Minato-Ku.
Tokyo, Japan.

Valette, A. Faculté de Sciences de Paris.
Ordinateur Pour Etudiants.
Tour 13.
9, Quay Saint Bernhard.
75, Paris 5e. France.

Veer, G. van der Vrije Universiteit.
Psychologisch Research Laboratorium.
De Boelelaan 1087 A.
Amsterdam, Neth.

Vejsova, A. Institute for Economy and Management of Scientific and Technical Development.
Konviktska 5.
Praha 1. C.S.S.R.

Vermijs, A.J.A. Philips' Telecommunicatie Ind.
Hilversum, The Netherlands.

Vinar, J. Safarik University.
Computing Centre.
Namestie Februareve vitazsta 9,
Kosice, U.S.S.R.

Vinsonhaler, J.F. 309 Computer Center.
Michigan State University.
East Lansing, Michigan 48823, U.S.A.

Vollebergh, J.J.A. Institute for Applied Psychology
Berg en Dalseweg 127
Nijmegen, The Netherlands.

Vos, M. de Equipe de Linguistique Anglaise de l'Institut Charles V.
Paris 4e France.

Vries, J.G. de Philips' N.V. Educational Research OA 4.
Kastanjelaan 1,
Eindhoven, The Netherlands.

Walker, T.M. Computer Science Department
University of Houston.
Cullen Blvd.
Houston, Texas 77004, U.S.A.

Weldon, E.J. Department of Electrical Engineering.
University of Hawai'
2565 The Mall,
Honolulu, Hawaii 9622, U.S.A.

Wente, K.W. AIV Institut.
61 Darmstadt.
Schöfferstrasse 2, Germany.

West, A.S. University of Denver.
Denver Research Institute.
University Park.
Denver, Colorado 80210, U.S.A.

White, T.C. American Federation of Information Processing Societies.
210 Summit Avenue.
Montvale, New Jersey 07645. U.S.A.

Wilkins, B.R. Department of Electronics.
The University.
Southampton, N.H., U.K.

Williams, N. Computer Services.
North York Board of Education.
Toronto, Ontario, Canada.

Wilson, B.J. Winchester College.
Culver Road.
Winchester, Hants, U.K.

Winder, C.L. Michigan State University.
College of Social Science.
East Lansing. Michigan 48823, U.S.A.

Wolff, K. Science Teaching, Center,
University of Maryland.
College Park 20742, U.S.A.

Wright, A.H. I C L Recruitment and Academic Relations.
Computer House.
Euston Centre.
London N.W. 1, U.K.

Zemanek, A. I B M Research Laboratory,
Parkring 10,
Wien, Austria.

Zemor, P. SERTI.
17 Rue de Monsigny.
Paris 2e. France.

Zinn, K.L. University of Michigan.
The Center for Research on Learning and Teaching.
1315 Hill Street,
Ann Arbor, Michigan 48104, U.S.A.

Zunde, P. School of Information and Comp. Science.
Georgia Institute of Technology.
Atlanta, Georgia 30332, U.S.A.