A student directed, computer assisted, programming environment

Martha Ann Marks
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd
Part of the Computer Sciences Commons

Recommended Citation
Marks, Martha Ann, "A student directed, computer assisted, programming environment " (1970). Retrospective Theses and Dissertations. 4249.
https://lib.dr.iastate.edu/rtd/4249

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
MARKS, Martha Ann, 1941-
A STUDENT DIRECTED, COMPUTER ASSISTED,
PROGRAMMING ENVIRONMENT.

Iowa State University, Ph.D., 1970
Computer Science

University Microfilms, A XEROX Company, Ann Arbor, Michigan
A STUDENT DIRECTED, COMPUTER ASSISTED, PROGRAMMING ENVIRONMENT

by

Martha Ann Marks

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of The Requirements for the Degree of DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

Head of Major Department

Signature was redacted for privacy.

Dean of Graduate College

Iowa State University
Ames, Iowa

1970
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIFICATION OF SYSTEM</td>
<td>1</td>
</tr>
<tr>
<td>DEVELOPMENT OF COMPUTER ASSISTED INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>Foundations</td>
<td>9</td>
</tr>
<tr>
<td>Drill, Tutorial, and Dialogue</td>
<td>11</td>
</tr>
<tr>
<td>Simulation and Gaming</td>
<td>12</td>
</tr>
<tr>
<td>Scholarly Aids</td>
<td>13</td>
</tr>
<tr>
<td>CAIMARK SYSTEM</td>
<td>15</td>
</tr>
<tr>
<td>Overview</td>
<td>15</td>
</tr>
<tr>
<td>Command Language</td>
<td>17</td>
</tr>
<tr>
<td>Philosophy of Use</td>
<td>21</td>
</tr>
<tr>
<td>Sample CAIMARK Session</td>
<td>22</td>
</tr>
<tr>
<td>INTERNAL STRUCTURE OF SYSTEM</td>
<td>30</td>
</tr>
<tr>
<td>CAIMARK Program Organization</td>
<td>30</td>
</tr>
<tr>
<td>Programming Environments</td>
<td>37</td>
</tr>
<tr>
<td>MTMT and CAIMARK</td>
<td>39</td>
</tr>
<tr>
<td>Creation and Editing of Files</td>
<td>40</td>
</tr>
<tr>
<td>ANALYSIS OF SYSTEM</td>
<td>41</td>
</tr>
<tr>
<td>Limitations</td>
<td>41</td>
</tr>
<tr>
<td>Extensions</td>
<td>42</td>
</tr>
<tr>
<td>Evaluation</td>
<td>44</td>
</tr>
<tr>
<td>Summary</td>
<td>47</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>48</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>53</td>
</tr>
<tr>
<td>APPENDIX A: PROGRAM LISTING</td>
<td>54</td>
</tr>
<tr>
<td>APPENDIX B: INFORMATION FILES</td>
<td>60</td>
</tr>
<tr>
<td>TOYLANG</td>
<td>60</td>
</tr>
<tr>
<td>HOWWORKS</td>
<td>71</td>
</tr>
</tbody>
</table>
SPECIFICATION OF SYSTEM

It has long been assumed that if the uniqueness of each student is provided for, his probability of learning is increased. Since learning is ultimately the task of the individual and each individual is indeed different, this assumption seems reasonable. With respect to his goals, his areas of interest, his rate of comprehension, whether he does best when greatly challenged, or whether he discourages easily, whether he works in bursts of activity or at a steady pace, one student differs from another. Such differences must be considered to determine the optimal educational program for each individual. With a private tutor directing the program, instruction can be individualized; the student's educational needs can be satisfied in the way best suited to his goals and learning characteristics. This greatly desired goal of providing an excellent tutor to direct the education of each student has been completely unattainable in the context of universal education.

The application of the computer to the field of education has been viewed by many as the means for at last satisfying this goal of providing individualized education for the masses. However, a major obstacle remains. To the author's knowledge, no general theory on how to individualize instruction exists. Although a great deal of effort has been expended trying to determine how people learn and to develop
a theory of education, Miller (30) states, "Research on human learning has not produced a substantial body of scientifically established knowledge that is useful for actual teaching."

As a result of this lack of a theoretic basis, the attempts to mechanize teaching, which is still very much an art rather than a science, have met with minimal success. This does not mean that use of the computer in education should be postponed, but rather that the methods of applying the computer should be re-examined.

Fundamental to much of the work done in the area of computer based instruction has been the computer control of the learning situation. Through the utilization of the results of past performance, the computer has attempted to direct the student, but as indicated above, no general theory on how to do this in even a quasi-optimal manner exists.

Perhaps a reversal of the direction of control in computer based education should be examined. Student direction is particularly appropriate when applied to two computer application areas, problem solving and information retrieval. Both of these areas are readily adapted to learning situations and have not yet been thoroughly explored or generally exploited. As Zinn (52) states, "It is desirable to make primary sources of knowledge more accessible to students through organized files of information, procedures and
associated learning tools. The increasing use of computers and electronic technology and an emphasis on skills of information acquisition and decision making recommend giving students more control over the learning environment."

If the student directs the computer based instructional situation, he individualizes his educational program by investigating his own areas of concern and by proceeding at his own rate in the sequence of steps that he finds most natural. He has the opportunity to probe, analyze, and experiment, learning by discovery as he does so. For the mature student this is most appropriate, for as he progresses in his education, his goals should increasingly become ones of self-education.

Bloom (4) states, "Individuals are expected as they mature, to solve problems on their own and to make decisions wisely on the basis of their own thinking...it is recognized that unless the individual can do his own problem solving he cannot maintain his integrity as an independent personality." Particularly in the latter stages of his education, it is important that a student be addressing the task of how to inform himself for he should be assuming the control of his own education.

In his taxonomy for educational objectives in the cognitive domain, Bloom (4) divides the domain into a hierarchical structure of knowledge, comprehension, application,
analysis, synthesis, and evaluation. Knowledge is the simplest objective to obtain. Comprehension is built upon knowledge for it involves the understanding of knowledge up to the point of interpretation and extrapolation. In many educational environments knowledge and comprehension are greatly emphasized because they are the easiest both to teach and to test. Most current Computer Assisted Instruction (CAI) applications follow this trend. These objectives are fundamental for young children, but for the mature student the educational objectives should involve far more than knowledge and comprehension. The student must learn to apply his knowledge, to create, and to evaluate. These latter educational objectives are particularly difficult to teach; however, if the student is to learn to educate himself and to progress to the point of being able to contribute to the body of knowledge, he must become adept at analysis, synthesis, and evaluation.

Problem solving in a programming environment offers an excellent opportunity for the student to exercise the skills of analysis, synthesis, and evaluation. Often it is possible for a student to solve a problem even though he has no real comprehension of how he arrived at the solution or, indeed, why it is a solution. Programming a solution helps avoid this dilemma of the competent but non-comprehending student.

In order to construct a program to solve his problem,
the student must first have thoroughly analyzed the problem. The construction of an algorithm or method of attack for solving his problem and the representation of it in some programming language constitute an exercise in synthesis. The student gains greater insight into the problem area by being forced to analyze the problem and to explicitly state a method of solution. Guided by the appreciation that comes only from doing, the student learns not only to recognize the awkwardness that accompanies an inept solution but to appreciate the elegance of simplicity and efficiency. Experience in evaluation is gained.

Ideally, a problem solving environment and the capability to ask questions in English should be provided to satisfy the goals of the student directed learning session at the computer. If the student encounters either a computer usage problem or a subject area problem which he cannot handle while working in the problem solving mode, he switches into the dialogue mode and converses with the computer; the student asks the questions and the computer answers them. Such a dialogue would place the control in the hands of the student. Unfortunately, this ideal situation is currently unattainable. Computer processing of the English language is not yet advanced enough to support such a sophisticated system. However, even if a natural language could be used for communication with the computer, techniques for handling the
answer processing are still in the research stage for any but the smallest information store and simplest of questions. Much work is currently being done in the areas of natural language processing and question-answer systems but the technology is not yet ready to support the student directed computer dialogue.

If, however, the student is given complete control, not to ask questions but to search the information store to find the answers for himself, then the system can easily be built within the constraints of today's technology. Giving the student control of such a system which combines a problem solving environment with a collection of information files seems to have great potential. The student can progress at his own rate. He is free to examine the material which interests and concerns him. The sequence in which he proceeds through the material is of his own choosing and, consequently, should be the one most natural for him. The student is encouraged to experiment and probe in order to more fully understand the material being studied. He can do so in private without fear of having his mistakes ridiculed. The student should learn that taking the wrong path can be a worthwhile experience if it helps him to gain insight into the topic being studied. Thus, the system provides an excellent opportunity for the student to learn through discovery.

A computer system for student directed use in problem
solving and information file searching must be simple and natural. Since it is to be a tool, the student should find it so natural that the mechanical use of the system can be done almost without conscious effort. The command language must be small in order that it can be quickly learned and easily remembered. The commands must be highly mnemonic, have simple syntax, and be short enough to keep the mechanics of interactions (such as typing) from becoming troublesome. The system must contain instructional materials to teach its own use; both a complete explanation of the system and a short summary should be provided.

Other design goals for such a system involve a different audience, the course author. If the system is to be used, it must be possible to create information files with a reasonable investment of time and effort. Organizing information and writing about it is nearly always considered difficult, but file construction should not be much more difficult than organizing a course and creating lecture notes for it. The constraints of the system must not be unduly binding on the course author.

Thus, the system must be easy for both the student and the course author to use. Simplicity is the keyword: simplicity of computer technique, simplicity of command language, and simplicity of file organization.

The programming environments should exist as systems
apart from the information retrieval system. Their design and maintenance should be independent and separate from the information file system, and each programming environment should be independent of all others.

In his discussion of the advantages of such a system, Zinn (53) states, "I believe that in this mode students could accomplish more scholarly work of greater quality during a given period of study and acquire more skill in searching for and organizing information than through discontinuous encounters with structured and strictly controlled tutorial instruction interspersed with periods of independent study."

Based on personal experience and supported by Wodtke's (49) findings, a system as highly verbal as the information system portion of the proposed student directed, computer assisted, programming environment should be implemented using a Cathode Ray Tube (CRT) terminal for communication with the computer. Wodtke found that there was an increase of 25% in instruction time for on-terminal experience with highly verbal, college level material using a teletype terminal, but there was no commensurate increase in learning. This increase in learning time was attributed to the slow character-by-character output rate (10 characters per second) on a teletype terminal. With a CRT information can be output orders of magnitude more rapidly.
DEVELOPMENT OF COMPUTER ASSISTED INSTRUCTION

Foundations

The use of teaching machines has long been investigated with the hope of finding a way to bring the mass production techniques of today's industry to the field of education. High quality teaching, in great quantity and at low cost, has been the goal. In the United States, Sidney L. Pressey (32), an Ohio State University psychologist, developed an automatic testing machine in 1920. This machine presented multiple-choice questions to the student user. If the student answered correctly, the next question was presented; otherwise, the error was recorded and the student was required to retry the question until he succeeded in answering correctly. Pressey noted that his machine could be considered a teaching, as well as a testing, machine. Education, however, was not yet ready for mechanization and in 1932 Pressey reluctantly dropped his work on machines.

In the fifties the idea of teaching machines again was advanced. B. F. Skinner (40), working with pigeons, found it was possible to teach them by reinforcing their correct responses immediately. He conjectured that his "contingency of reinforcement" theory would also be applicable to human learning, and in 1954 he promoted the idea of a teaching machine that would present material in steps so small that
the student was practically guaranteed to give the correct response. Immediate feedback of results was emphasized.

Norman A. Crowder (11) introduced a different approach to machine teaching which he labeled "intrinsic programming". His approach entailed much larger steps than those in Skinner's method. In this technique as the student assimilated the material, he was moved on through the lesson; otherwise, he was "branched" back for remedial work.

Crowder's branching techniques were difficult to handle in textbook form; however, the details that made "intrinsic programming" difficult in book form could be easily handled on the computer. The computer could accept and evaluate responses, had nearly unlimited branching capabilities, and could branch on a variety of criteria. In 1958, Nancy Anderson and Gustave Rath (33), at the IBM Research Center, used an IBM 650 to teach binary arithmetic in one of the first experiments in the instructional use of computer.

With their experiment, the age of computer aided education was begun. In the ensuing decade a great deal of effort was directed toward applying the computer to the field of education. So much in fact, that it is impossible to give a comprehensive list of even the most significant projects in the limited space of this paper. Instead, the author proposes to categorize current work and to indicate representative projects in each area. Zinn's (51) categorization
of the three major modes (or uses) of the computer in instruction will be used. They are:

1. drill, author controlled tutorial, and dialogue tutorial,
2. simulation and gaming,
3. scholarly aids.

Drill, Tutorial, and Dialogue

The first category comprises what is frequently thought of as CAI. It is based primarily upon the theories advanced by Skinner and Crowder. Patrick Suppes (41) of Stanford University is perhaps the most widely known researcher in the field; his work has been primarily in the area of drill and practice, initially applied to elementary school mathematics. Computer aided drill and practice also has been applied to the field of foreign language study with projects developed for college level Russian at Stanford and for German at State University of New York, at Stoneybrook. Recently, Uhr (46) at the University of Wisconsin has been investigating methods for programming the computer to generate the exercises needed for such CAI drill applications from rules or paradigms input by the course author.

Author controlled tutorial programs are an attempt to produce individualized, tutorial adaptations of lecture or textbook material. Penn State, Florida State and the University of Illinois are sites for work of this type. The
PLATO project (3) at Illinois is particularly well known. To lessen the CAI author's burden in developing course material for tutorial systems, a proliferation of CAI languages has sprung forth. Perhaps the best known of these languages is IBM's COURSEWRITER (30). One of the most advanced is PLANIT (14), developed at System Development Corporation (SDC).

Dialogue tutorials have attempted to give the impression of conversation with the computer. This gives the student the illusion that he is in control of the session. Best known examples of this approach are the physics problem solving session written by Taylor (44) using the ELIZA system and the exercise in medical diagnosis by Feurzeig (18) of Bolt, Beranek, and Newman, Inc. (BBN).

Simulation and Gaming

The computer program underlying a game or simulation is a model designed to provide an appropriate reply, regardless of the student's input. This mode of operation enables the examination of decision making skills and the testing of hypotheses about the area being studied. Models range from very realistic representations with maximum fidelity, particularly for military and space applications in which simulations are often used to give instruction in areas where actual instruction is difficult or impossible to achieve, to very abstract representations which help to amplify the basic
principles through simplification of extraneous detail. An example of such an application is the Sumerian game (48), which has the student assume the role of king of a mythical monarchy. By making decisions concerning the utilization of the resources of the kingdom, the student learns some basic economic principles.

Scholarly Aids

The areas of information handling and problem solving are classified under scholarly aids. The work in the field of information handling has been concerned with the development of the computer technology necessary to support the systems desired. General application to education has not yet occurred but the potential has been recognized. Problem solving applications tend to be overlooked, perhaps because they are so obvious. Although the emphasis of computer based instruction has been in the area of drill and practice, some work has been done in problem solving. The Massachusetts' Board of Education project (34), for example, has used the computer as a mathematics laboratory. Junior high and high school students in arithmetic, algebra, analytic geometry and trigonometry classes have used programming as a tool in problem solving to augment their understanding via the use of algorithms. The CRICISAM experiment in the teaching of sophomore calculus at Iowa State University (and other universities) is another currently active project. In this
experiment the computer is used to enable the student to achieve a more intuitive understanding of the material and to emphasize an algorithmic approach.

An area of growing interest for application of the computer to the field of education is that of instruction management. This application, often called individually prescribed instruction, uses the computer to generate a suggested instructional program tailored to the individual student on the basis of his educational profile, the course objectives, and the instructional materials available. The University of Pittsburgh (15) is one of the sites where research in this area is being pursued.
CAIMARK SYSTEM

Overview

CAIMARK is a pilot program designed to illustrate the feasibility of a student directed, computer assisted, programming environment approach to computer assisted education. The CAIMARK system consists of two major parts: an information system and a group of programming environments. The programming environments offer a variety of problem solving facilities. The information system provides files and facilities for using them; the student searches the files in order to answer his own questions. The CAIMARK system is completely controlled by the student user via a small set of commands. These commands allow the student to discover what files exist in the system, to examine any of the files, and to switch from the information system to a programming environment and back again. Thus, a student can work in a problem solving environment, yet have various information files readily available. This enables him to quickly find answers to the questions that arise while he works in the problem solving mode. The information files can also be used to suggest exercises or to give guidance to the student. Even divorced from the programming environment, the information file portion of the system is a usable tool for education. For areas in which the materials change rapidly, creating the need for constant updating, the use of such a
The information system which has been designed to use the IBM 2260 CRT terminal, consists of a collection of files. Each file is itself a collection of records; each record is a single screen image (12 lines of 80 characters each). Line 1 is reserved for heading information identifying the terminal, the file currently being accessed and the record being viewed. Line 2 serves two purposes. The user places his commands to the system there and the system places all error messages for the user in that line. The remaining ten lines of the screen hold the information contained in the record. Thus each record effectively has an 800 character capacity.

Each file is organized in outline form. Each record in the file is given a Record-In-File (RIF) number consisting of three portions, two digits each, separated by periods. These numbers give a hierarchical order to the file since each portion of the RIF number can be thought of as specifying a level. The leftmost portion is considered to represent level one, or the top level. To have a three level number is a somewhat arbitrary decision and may easily be changed. The size of the RIF number is increased by three additional characters for every level added. Three levels seems to give sufficient organizational structure and yet
requires a reasonably sized RIF number.

To access information from the system, the user requests a particular file. The requested file is made active and its introductory record, the one with RIF number 00.00.00, is displayed on the 2260 screen. The student is then free to examine any record contained within that file. At any time the student may request a different file, switch into a programming environment or, if confused, simply ask the system for help.

Under the current pilot system a file is restricted to 80 records and the information store to 20 files. Since the data set containing all of the files currently has only 100 records, it places an even greater constraint on the size of the system. All of these parameters could easily be changed for a larger system.

Command Language

Commands are issued by typing a command keyword (or its abbreviation) in line 2. If the command requires an argument, the argument should be typed following the command keyword, but separated from it by blanks. For the computer to be aware of the information typed on the screen by the user, the student must depress the SHIFT key and hit the ENTER key when the entry is complete. (In the remainder of this paper, hit ENTER will mean depress the SHIFT key and hit ENTER.)
The computer will respond by executing the command requested or by placing an error message in line 2 of the display.

The following commands constitute the complete command language for the CAIMARK system: SIGNON, OFF, RETURN, HELP, FILELIST, FILEREQ, VIEW, UP, DOWN, FORWARD, BACK, and PROGRAM. The three most important commands for the student to remember are: OFF, HELP, and RETURN. The letter underlined in each command word given below is the official abbreviation for that command. The computer will acknowledge either a command keyword or a single letter abbreviation but not a misspelled command word.

OFF commands the termination of the student's session at the terminal. It will be the last command he issues for that session.

HELP places the beginning frame of the HOWWORKS file on the screen. That file contains the rules for using the CAIMARK system, a description of the system, an explanation of the command language and an example of a terminal session. After reading the HOWWORKS file the student should be able to continue his session.

RETURN causes an introductory display containing a list of the system commands to be presented on the screen. This display should help to reorient
the student to the system.

**PROGRAM ARG** switches the student into the programming environment specified by ARG.

**FILELIST** displays on the 2260 screen a list of the names of all files currently in the information system.

**FILEREQ ARG** causes the introductory record, with RIF number 00.00.00, of the file specified by ARG to be displayed on the screen. It is assumed that this record contains a table of contents or introductory material that indicates the organization of the file.

**VIEW ARG** displays the record of the current file whose RIF number is specified by ARG. The last file requested is the current file. When a student first signs on the system, the INTRDUCT file is current. At any time there is only one current file (and one current record).

**FORWARD** causes the current RIF number (the RIF number of the record currently displayed) to be modified and the record specified by this new number to be displayed. The current RIF number is modified as follows: the rightmost non-zero portion of the RIF number is incremented by one. This corresponds to remaining on the same level in the outline construction, but moving forward one point. If the
current RIF number is 00.00.00, the FORWARD command causes record 01.00.00 to be displayed. 

**BACK** corresponds to **FORWARD**. The rightmost non-zero portion of the current RIF number is decremented by one and the record specified by this new number is displayed. If the current RIF number is 01.00.00, the **BACK** command causes record 00.00.00 to be displayed. Each 2 digit portion of the RIF number is handled as a circular counter with range 1 to 99 during the processing of the **FORWARD** and **BACK** commands.

**DOWN** causes the leftmost zero portion of the current RIF number to be replaced by 01 and the record specified by this new number to be displayed. **DOWN** corresponds to descending to the first point of the next level in an outline.

**UP** corresponds to **DOWN**. The rightmost non-zero portion of the current RIF number is replaced by 00 and the record specified by this new number is displayed.

**SIGNON** is not currently used.

In an actual system, as opposed to the pilot system, **SIGNON** and **OFF** would have slightly different functions. **SIGNON** would record the student's name and accounting information. **OFF** would return control to an idling display
within CAIMARK. The system would then wait in readiness for the next student to sign on.

Philosophy of Use

The CAIMARK system is primarily for use by the mature student interested in learning to educate himself. However, under the label "student directed, computer assisted, programming environment" there is room for significant divergence of approach. Files can be written to lead the student through the material, record by record, on a predetermined path. However, the student is always free to disregard the suggested path and travel on his own. The author envisions this self-direction as the strong point of the system. Consequently, file organization is viewed with an eye toward making it easy for the student to discover what information is available and where it can be found. With this in mind, the author advocates a file organization scheme using a hierarchy of tables of contents (or directories).

The author views the CAIMARK system primarily as a workshop or lab to be used in conjunction with an alternate presentation of material, probably a lecture. The student works at CAIMARK in the problem solving mode and switches into the information system to find immediate answers to his questions as they arise. Therefore, it is essential that the files be organized so that a student can easily determine where to look for an answer. If the subject
area is one in which terminology is difficult, a primer file should exist which would permit the student to find material on the concept in question, even if he does not know the official terminology. If the system is to be used as the sole method of presenting material, it is essential that the student be presented with a complete set of course objectives. It is impossible for the student to evaluate himself if he does not know what the accepted standards are.

Sample CAIMARK Session

A sample session with the CAIMARK system might be as follows. When the student signs onto the system, the introductory record of the INTRDUCT file (see Figure 1) is displayed. Since he is interested in programming in the TOY language, he types the command PROGRAM TOY on line 2 of the display (see Figure 2) and hits ENTER. The system switches him into the TOY programming environment. The introductory display of the TOY environment is displayed (see Figure 3) and the system is ready to receive his program. After working a while, the student generates a syntax error in the transfer statement he is trying to submit (see Figure 4) but cannot discover why the statement is in error. In order to check the syntax of the TOY transfer statement, he issues the command RETURN and is switched back into the information system portion of CAIMARK. The introductory record of INTRDUCT (see Figure 1) is the information
displayed upon his return. He then commands FILELIST to refresh his memory on the name of the TOY language file. After checking the list of files available (see Figure 5), he issues the command FILEREQ TOYLANG (see Figure 6). The introductory record of this file (see Figure 7) informs him that the RIF number for the formal specification of the TOY syntax is 02.00.00, so he issues the command VUE 02.00.00. He is informed that his input was not a recognizable command (see Figure 8). Noting the misspelling, the student commands VIEW 02.00.00 (see Figure 9). The information displayed requests that he command DOWN to view the syntax. When he does, the record presented (see Figure 10) does not contain enough information, so he issues the command FORWARD. The next record (see Figure 11) gives him the information he seeks. The student returns to the programming environment by commanding PROGRAM TOY.
THE COMMANDS AVAILABLE ARE LISTED BELOW:
RETURN OFF HELP
FILELIST FILEREQ
BACK FORWARD
DOWN UP
VIEW PROGRAM
IF YOU ARE UNFAMILIAR WITH THE SYSTEM, TYPE HELP, SHIFT AND HIT ENTER, AND YOU WILL BE PRESENTED WITH EXPLANATORY MATERIAL. OTHERWISE, SELECT YOUR COMMAND AND PROCEED.
NOTE: THE ONLY PROGRAMMING ENVIRONMENT CURRENTLY AVAILABLE IS TOY.

FIGURE 1. THE FIRST RECORD OF THE INTRODUCT FILE.

THE COMMANDS AVAILABLE ARE LISTED BELOW:
RETURN OFF HELP
FILELIST FILEREQ
BACK FORWARD
DOWN UP
VIEW PROGRAM
IF YOU ARE UNFAMILIAR WITH THE SYSTEM, TYPE HELP, SHIFT AND HIT ENTER, AND YOU WILL BE PRESENTED WITH EXPLANATORY MATERIAL. OTHERWISE, SELECT YOUR COMMAND AND PROCEED.
NOTE: THE ONLY PROGRAMMING ENVIRONMENT CURRENTLY AVAILABLE IS TOY.

FIGURE 2. THE COMMAND PROGRAM TOY ENTERED ON LINE 2 OF THE DISPLAY SCREEN SHOWN IN FIGURE 1.
TOY COMPUTING FACILITY

ERROR MESSAGES WILL APPEAR ON LINE2. PROGRAM LISTING WILL ACCUMULATE BEGINNING IN LINE3. PROGRAM STMTS ARE INPUT ONE AT A TIME. AFTER TYPING IN A STMT, HOLD SHIFT, AND HIT ENTER. MESSAGES WILL APPEAR IN LINE2. TO EXECUTE LISTED PROGRAM TYPE EXECUTE. TO RETURN TO FILE INFORMATION PROGRAM TYPE RETURN. PROGRAM SUBMITTAL MAY BEGIN.

FIGURE 3. THE INTRODUCTORY DISPLAY FOR THE TOY ENVIRONMENT.

SYNTAX ERROR. EITHER CORRECT STMT OR TYPE RETURN, SHIFT, AND ENTER.
KK=105;KKK=13*(KK+15);

GO TO AROUND;

FIGURE 4. DISPLAY SCREEN MIDWAY THROUGH SUBMITTAL OF A TOY PROGRAM WITH SYNTAX ERROR.
FILELIST

ALL FILES AVAIL ON SYS LISTED BELOW. PLACE COMMAND FILEREQ NAMEOFFILE ON LINE2.
TOYLANG           HOWWORKS           INTRODUCT           METACAI

FIGURE 5. THE IMAGE DISPLAYED WHEN THE FILELIST COMMAND IS GIVEN.

FIGURE 6. THE FILELIST DISPLAY WITH THE COMMAND FILEREQ TOYLANG ENTERED IN LINE 2.
TABLE OF CONTENTS FOR THE TOYLANG FILE

Currently the Toy Language file is comprised of the materials listed below:

01.00.00 Toy Reference Manual
02.00.00 Formal Syntax Specification for Toy
03.00.00 Toy Exercises

To reference any of these units issue the command VIEW followed by the appropriate unit number. To submit a program in Toy issue the command PROGRAM TOY and you will be switched to the Toy Programming Environment. To return to the file information system from the Programming Environment issue the command RETURN; have fun playing with Toy.

FIGURE 7. The introductory display of the TOYLANG file.

FIGURE 8. In line 2 above, the error message generated by the erroneous command VUF 02.00.00.
TABLE OF CONTENTS FOR THE TOYLANG FILE
VIEW 02.00.00 COGNIZABLE COMMAND. IF UNABLE TO FIND ERROR, TYPE HELP.
CURRENTLY THE TOY LANGUAGE FILE IS COMPRISED OF THE MATERIALS LISTED BELOW.
   01.00.00 TOY REFERENCE MANUAL
   02.00.00 FORMAL SYNTAX SPECIFICATION FOR TOY
   03.00.00 TOY EXERCISES
TO REFERENCE ANY OF THESE UNITS ISSUE THE COMMAND VIEW FOLLOWED BY THE APPROPRIATE UNIT NUMBER. TO SUBMIT A PROGRAM IN TOY ISSUE THE COMMAND PROGRAM TOY AND YOU WILL BE SWITCHED TO THE TOY PROGRAMMING ENVIRONMENT. TO RETURN TO THE FILE INFORMATION SYSTEM FROM THE PROGRAMMING ENVIRONMENT ISSUE THE COMMAND RETURN; HAVE FUN PLAYING WITH TOY.

FIGURE 9. THE CORRECT COMMAND VIEW 02.00.00 TYPED OVER THE ERROR MESSAGE IN LINE 2 OF THE DISPLAY SCREEN.

FORMAL SYNTAX FOR TOY. TYPE FORWARD TO VIEW THE REST

<DIGIT>::=0112...9
<LETTER>::=A|B|C|...Z
<CON>::=<DIGIT>|<CON><DIGIT>
<VAR>::=<LETTER>|<VAR><LETTER>
<AOP>::=+-|
<MOP>::=*
<ROP>:::==|
<EL>::=<VAR>|<CON>|(<EXP>)
<TERM>::=<EL>|<TERM><MOP><EL>
<PRIM>::=<TERM>|<PRIM><AOP><TERM>

FIGURE 10. THE FIRST DISPLAY OF THE ACTUAL SYNTAX SPECIFICATION FOR TOY.
<EXP>::=<PRIM>
<REL>::=<EXP><ROP><EXP>
<ASSIGN>::=<VAR><FXP>
<TRANSF>::=GOTO<VAR>
<IF>::=IF<REL>THEN<STMT>
<GET>::=GET(<VARLIST>)
<PUT>::=PUT(<VARLIST>)
<VARLIST>::=<VAR>|<VARLIST>,<VAR>
<STMT>::=<ASSIGN>|<TRANSF>|<GET>|<PUT>
<ST>::=<STMT>|<IF>

FIGURE 11. THE FORMAL SYNTAX RECORD GIVING THE DEFINITION OF THE TOY TRANSFER STATEMENT.
INTERNAL STRUCTURE OF SYSTEM

CAIMARK Program Organization

All files for CAIMARK are stored on a single, direct access data set named MTMT2260.BOOKMARK. MTMT2260.BOOKMARK consists of 100 records, each of which is 960 characters in length and has a regional (1) organization. Records within a regional (1) data set are ordered and numbered, starting at zero; the records are then accessible via the record numbers. The first record of MTMT2260.BOOKMARK (see Figure 12) contains directory information for the data set. It indicates the number of files on the data set, the record number of the next free record and the name, first record number and number of records in the file for each file. Each number is right justified in a field of width 4; each file name is left justified in a field of width 8. The first record of each file holds directory information for that file. Each record within a file, except the first, is identified by a CAIMARK RIF number and is accessed within CAIMARK by that number. The file directory lists for each record in the file its RIF number and immediately following, right justified in a field of width 4, a position number indicating where the record is located in the data set (see Figure 13). When a user requests a file via the FILEREQ command, its file directory information (found in its first record) is obtained and stored in the active file directory. The
FIGURE 12. THE DIRECTORY INFORMATION RECORD FOR THE DATA SET MTMT2260.BOOKMARK.

00.00.00 2301.00.00 2402.00.00 2502.01.00 2602.02.00 2702.03.00 2802.04.00
2902.05.00 3003.00.00 3103.01.00 32

FIGURE 13. THE FIRST RECORD OF THE HCWORKS FILE CONTAINING THE FILE DIRECTORY INFORMATION FOR THAT FILE.
second record of the file, assumed to have RIF number 00.00.00, is displayed on the 2260 screen. RIF number 00.00.00 is taken as the current RIF number. When a record is to be viewed, the active directory is searched for its RIF number, its data set position number is retrieved, and the record with that position number is brought in from the data set and displayed on the 2260 screen.

When a file directory is to be searched for a RIF number, the subroutine SEARCH (which uses a binary chopping search technique) is called. The first argument of SEARCH is the RIF number. Upon return from SEARCH, the second argument holds the corresponding data set position number and the third argument indicates a message to be displayed.

The CAIMARK system is embedded in the Multiple Terminal Monitor Task (MTMT) system. Communication between the CAIMARK program and the 2260 terminal is handled by the MTMT subroutine DSP2260. This routine requires two arguments: a 960 character string containing the screen image to be displayed and a fixed binary integer (between 1 and 12 inclusive) indicating the line on which the cursor should be placed. The cursor designates the position on the screen where the next character entered at the terminal will be placed. When control returns to CAIMARK from the subroutine, the information that was on the display screen when the user hit ENTER replaces the 960 character string
given as the first argument of DSP2260.

When the CAIMARK system is activated, a message indicating this is displayed on the 2260 screen and the user is requested to hit ENTER. When control returns to the system, the identification number of the terminal in use is retrieved from positions 3 and 4 of the first screen image by the CAIMARK program. This information is used to associate MTMT2260.BOOKMARK (the CAIMARK data set) with the data definition names available through MTMT at the participating terminal. MTMT2260.BOOKMARK is opened and the data set directory information is retrieved. The INTRDUCT file is made active; its file directory information is retrieved and stored as the active file directory. The INTRDUCT record with RIF number 00.00.00 is displayed on the terminal screen.

From this point on, the session is directed by the student user. For the session to proceed, he must type a command in line 2 of the 2260 display screen and return control to the CAIMARK system by hitting ENTER. The CAIMARK program then searches line 2 of the returned screen image from left to right for the first non-blank sequence of characters. If these characters compose a command keyword or an official abbreviation for a command keyword, control is switched to the portion of the CAIMARK program that processes that feature. Otherwise, an error message is displayed on the screen indicating that the input was not a recognizable command.
If the command cannot be completely processed, an appropriate error message is displayed on line 2 of the 2260 screen. Otherwise, a new record is displayed. In any event, for the session to proceed the student must once more issue a command to the CAIMARK system.

Internally, the various command features are processed as explained below. For all commands that require an argument, the subroutine FINDARG is called. It returns the next non-blank sequence of characters found during a left to right scan of the remainder of line 2. If none is found, it messages the error.

OFF branches control to the end of the CAIMARK program. Control is subsequently returned to the first display of the MTMT system.

HELP is processed as a FILEREQ command to the HOWWORKS file.

RETURN is processed as a FILEREQ command to the INTRDUCT file.

FILEREQ, as indicated above, retrieves the directory information (from the data set MTMT2260.BOOKMARK) for the file specified by the argument of the command and places this information in the active file directory. The second record of the file, the one with RIF number 00.00.00, is displayed on the 2260 screen. If the file requested does not
exist, an error is messaged.

FILELIST causes a display of the files currently available in the MTMT2260.BOOKMARK data set. This display is generated from the data set directory information when the CAIMARK system is activated.

PROGRAM issues a subroutine call to the program supporting the programming environment requested. If an unknown programming environment is requested, an error message is displayed. A RETURN command is processed when control returns to CAIMARK from the programming environment.

VIEW searches the active file directory for the RIF number requested, retrieves the associated data set position number, retrieves the record with that number from the data set, and displays it on the 2260 screen. If the RIF number requested is not in the file directory, the error is messaged.

UP, DOWN, BACK, and FORWARD all modify the current RIF number to generate a new one. The commands are then processed as if a VIEW command had been issued with the new RIF number as an argument.

UP zeroes out the rightmost non-zero portion of the current RIF number. If both the second and third portions of the current RIF number are zero when the command UP is issued, an error is messaged.
DOWN replaces the leftmost zero portion of the current RIF number with 01. If both the second and third portions of the current RIF number are non-zero when the command DOWN is issued, an error is messaged.

FORWARD increments the rightmost non-zero portion of the current RIF number. If the current RIF number is 00.00.00, the FORWARD command causes record 01.00.00 to be displayed.

BACK decrements the rightmost non-zero portion of the current RIF number. If the current RIF number is 01.00.00, the BACK command causes record 00.00.00 to be displayed. Each 2 digit portion of the RIF number is handled as a circular counter with range 1 to 99 during the processing of the FORWARD and BACK commands.

SIGNON is currently a non-functional command and is processed as an OFF.

If, by chance, a CAIMARK program error occurs during a terminal session, the error handling routine gains control. A message is displayed on the screen indicating the oncode, a numeric code that identifies the type of error, the onsource, which holds the field that was being processed when a conversion error occurred, and the onkey, the value of the key for the record that caused the I/O condition to be
raised. The student responds by issuing the RETURN command to continue his session. Thus, CAIMARK can recover from its own errors and permit the student to proceed.

Programming Environments

Each programming environment that is available on the system is a self-contained unit. Each follows its own rules and conventions. It is assumed that at least one information file of explanatory material exists for each programming environment in the system.

The system theoretically can support both interpreters and compilers as programming environments. Interpreters are attached by calling them directly from the CAIMARK program. The interpreter must handle its own interaction with the terminal. CAIMARK interacts with an interpreter only to the extent that it gives control to the interpreter and receives control back from it; no information is transmitted between them. For CAIMARK to attach a compiler, it would call an assembler subprogram to act as an interface. The interface routine would handle I/O to the terminal, the attachment of system modules, and the transfer of information between the terminal and the system programs. Thus, if the PL/1 language were the programming environment desired, CAIMARK would call INTERFACE. INTERFACE would handle the input of the PL/1 program from the terminal and attach
the PL/1 compiler, giving it information as to where the input program could be found. The object code output from the compile step would be placed on disk and INTERFACE would call the loader. The resulting load module would be scheduled for execution. After the execution was completed, the output would be returned to the terminal via INTERFACE. Thus, INTERFACE would act like a small operating system handling the tasks involved in calling a compiler from the terminal. The benefit to be gained by the use of a compiler must be weighed against the difficulty in using it. The main advantage is that the compiler already exists. Thus, it is unnecessary to expend effort in developing a new programming environment, only effort to create the interface. Any interface developed should be general enough to work for all compilers in the system. Currently, the worth of developing the interface is a moot point. The compilers are too large to fit into the current MTMT region, even if the interface existed. Hence, effort has not been invested to develop the interface.

A single interpreter, TOYLANG, is available as a programming environment in CAIMARK. TOYLANG is a small, arithmetic language designed solely for display purposes. It contains assignment, conditional, transfer, and I/O statements. The TOYLANG file gives the complete definition of the language (see Appendix B).
MTMT and CAIMARK

The following information explains how to establish and use CAIMARK as a foreground job in the MTMT system. Communication between the program and the user at the terminal is handled via the MTMT subroutine DSP2260. This subroutine transmits material from the program to the 2260 display screen and returns the screen image to the program with any changes made by the user. The CAIMARK program must be link-edited into the system library so that it can be called from the terminal. To call CAIMARK from the MTMT system, it is first necessary to change the associations between the data set names and the data definition names for the terminal so that the data set used by CAIMARK is accessible. The CAIMARK data set name, MTMT2260.BOOKMARK, must be associated with the data definition name SYSINxx (where xx is the terminal identification number). When this has been done, CAIMARK can be called as a foreground job by requesting option CAIMARK with parameter J through the opening display of the MTMT system. The CAIMARK command OFF returns the user to the MTMT system where he chooses option 0 to sign off the terminal completely.

It is important to note that MTMT is much too powerful a system to permit general student access to the facilities to be feasible. In an actual CAIMARK system (as opposed to the pilot system) the system would be activated once a day
by computer center personnel rather than by each user through MTMT. The command OFF would be changed to return control to a CAIMARK idling display and place the system in a wait state until another user appeared.

Creation and Editing of Files

Creation and updating of the files for the information system are performed outside of the CAIMARK system. This serves several purposes. First, it ensures the files' complete protection against misuse during a CAIMARK session. Since they cannot be written within CAIMARK, it is impossible for a student to inadvertently or maliciously destroy files. Secondly, since neither writing nor editing is to be done, there is no need to provide commands to handle these operations. The command language can remain small and simple. This means the language processing facilities in the system can remain miniscule and the system can be quickly mastered and easily used. Finally, there exist adequate file creation and editing facilities available within the MTMT system. The program EDIT960 permits easy editing of the regional(1) data set used in the CAIMARK program. To provide file editing facilities within CAIMARK would be a wasteful duplication of effort.
ANALYSIS OF SYSTEM

Limitations

Student control of the problem solving environment and information store is not the answer to all instructional situations. The system described in this paper was predicated on a mature, motivated user. The student directing his own studies must be mature enough to accept the responsibility that self-direction entails. He must have a substantial grasp of the material being studied in order to know where to look for answers. A well-organized primer file can help alleviate this particular problem, but it cannot solve it entirely. The student must be capable of honestly and realistically judging his own work. Poor students, in particular, seem to have difficulty in this area; they tend to be overly optimistic about their level of understanding. Motivation is necessary in order for the student to take advantage of the opportunity the system offers for experimentation and learning via discovery.

The system is poorly adapted to the "lost" student and to drill and practice applications since in both cases the student is unable to evaluate his performance effectively. Although the system is envisioned by the author as a workshop or lab to accompany a course lecture, it can be used as the sole method of presenting course material. If the system were used in this manner, it would be imperative
that the student be given explicit specification of the course objectives so that he has a basis for his self-evaluation.

Extensions

While working with the CAIMARK pilot program, several areas for possible extension have arisen. The most obvious and most immediate is to transform CAIMARK from a pilot project into an actual system, leaving its present organization unchanged. The modifications needed to do this are minor. They deal entirely with the size of the data set and the parameters that specify its organization. Currently, the data set contains a maximum of 100 records, the number of files is limited to 20, and the number of records within a file to 80. All of these limits are extendible.

To make the system more appealing as a research tool, means for collecting statistics could be added. This would greatly increase the significance of the project. The computer's potential as an educational research tool is, perhaps, as important as its potential as a teaching machine. By computer monitoring of student sessions at the terminal, the learning process can be observed as it never before has been. These computer monitored observations have the additional benefit that the student need never be aware that he is being observed. He can perform in private, unconcerned
and unembarrassed by the fact that he is being observed.

If the researcher is indiscriminate in his use of the data collection facilities available with the computer, he may find himself inundated with statistics. The major question becomes what should be monitored. In the CAIMARK system information on student file usage seems to be the most significant and appropriate data to collect. Information on the order of access of files, and of records within files, during a student session could be helpful not only in determining optimal ways to sequence materials but in discovering how students learn. This information on order of access could be compiled by having a trace of each session kept. The trace could list the files requested and the records within each file in the order they were viewed. Usage counts for each record of every file could be accumulated by the system as the trace was being compiled. This material could be quite useful in determining which records were superfluous and in evaluating the information files in general.

Another useful extension to the system would be to provide a workspace or save area for each user. Into this area the student would place any information he wished to keep from session to session. Thus, the student would use this area to develop his own personal information store. Major extensions to the CAIMARK system would be required to pro-
vide this facility. The command language would have to be enlarged to include a command to save information in the work area. A separate data set would have to be used for the work areas in order to maintain protection for the information files. The handling of two data sets would necessitate either more involved processing for the current file and record viewing commands or a new set of commands. In any event, adding student workspace to the system would involve significant modifications and extensions.

The current CAIMARK system is so simple that it is a great temptation to start modifying and extending it. However, much of the system's beauty is its simplicity. The temptation to add undue complexity should be contained.

Evaluation

Educational research is plagued by the following dilemma. It seems that experiments can be either realistic or well-controlled, but not both. It is possible to design well-controlled experiments on restricted portions of the learning process. However, there is no guarantee that the results of such experiments can be generalized to actual learning situations. It is also difficult to isolate meaningful test variables to be studied in these well-controlled laboratory-type experiments. If key variables are not isolated, the experimental results are completely irrelevant.
Field tests of material or techniques avoid the questions of generalization and of the isolation of important variables, but their validity is somewhat questionable since it is impossible to attain a completely controlled test.

When doing comparison field tests between a new educational approach and the conventional one, an additional factor known as the Hawthorne effect often clouds the results. That is, the role that simple curiosity plays in slanting results in favor of the new and different approach cannot be disregarded nor, unfortunately, can it be measured. Thus, there seems to be no really satisfactory way to evaluate educational techniques and theories.

Keeping this research dilemma in mind, the author advocates field testing of the student directed, computer assisted, programming environment approach as a means of evaluation. The results of the tests would be dependent upon the type of students participating, the quality of the file materials, the quality of the conventional classroom presentation, and to a very great extent, the attitudes of the field test administrators.

At least four different presentations should be compared:

1. The student directed computer system as the sole method of presenting material,
2. A lecture presentation with the computer system
used as an accompanying workshop or lab,

3. A conventional classroom presentation of the ma­
terial,

4. A lecture presentation with only the programming
environment portion of the student directed com­
puter system available to be used as an accompany­
ing lab.

The student population as a whole should cover the full
range of ability, maturity, and motivation. The population
for each of the presentations should be balanced with respect
to these attributes.

At best, the results of such field tests would merely
indicate trends, and at worst would have no significance at
all. That is not to say, however, that such field tests
would be worthless. They would play a very significant role
in helping to establish cost estimates for the CAIMARK ap­
proach. This is most important since the economic aspects,
as well as the academic ones, must be considered when evalu­
ating the system.

Computer hardware, software and course development ex­
penses must be determined. There is a tendency to concen­
trate upon the computer costs involved in such a project.
The cost for computer processing time, file storage, terminal
equipment, data communications networks, and computer soft­
ware development is only a part of the total cost for the
system. With past experience in CAI projects as a guide, it seems likely that the development of course (or file) materials may be the most significant expense. Information files, since they are structured in outline form, should be much simpler, hence less expensive, to prepare than course material for the traditional CAI "branching" approach or for a dialogue presentation. A reasonable estimate for the cost of course development should be possible to determine with the experience gained from field tests of the system.

Summary

A student directed, computer assisted, programming environment approach to the use of the computer for education is presented in this paper. This approach combines a problem solving environment with an information file system to produce a powerful learning tool. With this approach the student is given complete control of his learning session. He is free to use the system in any way that seems appropriate to him. The approach seems to have merit; however, a final evaluation of the approach must await extended field testing.

2. Becker, J. W. Run computer, run, a critique. ERIC ED 031 081. 1968.


42. Suppes, P. and Morningstar, M. Evaluation of three computer-assisted instruction programs. ERIC ED 031 408. 1966.


50. Yeager, J. L. and Slager, R. The learning research and development center at the University of Pittsburgh. ERIC ED 028 493. 1968.


ACKNOWLEDGMENTS

The author wishes to express her gratitude to Dr. Clair G. Maple for his help and guidance in the preparation of this paper. Without his concern it is doubtful that the research work and paper would have been completed. Special thanks are given to Dr. Robert M. Stewart for providing a teaching assignment that permitted the author time to cope with the trials of thesis preparation and for his encouragement, concern, and guidance. Special thanks are given to Dr. Robert J. Lambert for his continued support and assistance through the years since the author was an undergraduate advisee of his.

The author wishes to express her deep appreciation to her parents for their support, concern, and, perhaps most of all, endurance.

Thanks are given to Rex Thomas and Dennis Branstad for their aid given in so many ways.

The author finally wants to thank Mrs. Sharon Looschen, Mrs. Donald Phipps, and the Iowa State University Thesis staff for their respective assistance in thesis editing, typing, and final acceptance.
APPENDIX A: PROGRAM LISTING
FILES: PROC OPTIONS(Main);
DCL KEYS (80) CHAR (8), /*HOLDS CHAR KEYS FOR CURRENTLY active
FILE. */
SYSPRINT OUTPUT PRINT STREAM,
OWNST ENTRY, /* NAME OF TOYLANG PROGRAM */
ERNO FIXED BIN, /* INDICATES NO. OF MESSAGE FOR DISPLAY */
ARG CHAR (8),
SCREEN CHAR (960) INIT('FILELIST'),
SLAST CHAR (800) DEF SCREEN POS (161),
FINDARG ENTRY (CHAR (8)),
SEARCH ENTRY (CHAR (8), FIXED BIN, FIXED BIN),
DSP2260 ENTRY (CHAR (960), FIXED BIN (31, 0)),
BOOK FILE DIRECT KEYED INPUT ENV (F (960) REGIONAL (1)),
/* BOOK IS THE PL/1 FILE NAME FOR THE CAIMARK DATA SET */
NOS (80) FIXED BIN, /* HOLDS REGIONS NOS. ASSC. WITH KEYS FOR CUR. active FILE. */
BUF CHAR (960), /* USED AS BUFFER TO HOLD DIRECTORY INTO STORED ON active FILE. */
RGNO FIXED BIN, /* HOLDS REGION NO. OF 1ST RECORD IN CUR. active FILE. */
EXT FIXED BIN, /* NO. OF RECORDS IN CUR. active FILE. */
1 FILEDIRECT, /*MAIN LEVEL OF FOLLOWING STRUCTURE. */
  2 FNO FIXED BIN, /* NO. OF FILES IN THE SYSTEM */
  2 RNO FIXED BIN, /* NO. OF NEXT FREE REGION */
  2 INFO (20), /* HOLDS BASIC INFO ABOUT EACH FILE IN SYS */
    3 NAME CHAR (8), /* NAME OF FILE */
    3 REGNO FIXED BIN, /* REGNO OF 1ST REC OF FILE */
    3 EXTENT FIXED BIN, /* NO. OF RECS IN FILE */
LAB (0:12) LABEL INIT(
  ERR, ON, OFF, MET, RET, FLIST, FREQ, VIEW, FOR, BACK, DOWN, UP, HELP ), /* USED AS SWITCH TO BRANCH TO CODE TO PROCESS CURRENT COMMAND */
STAND CHAR (8),
OPS CHAR (96) INIT ('SIGNON OFF PROGRAM RETURN FILELISTFILEREQ VIEW FORWARD
  DOWN UP HELP '), /* COMMAND KEYWORDS (ORDER MATCHES LAB SWITCH) */
STUDENT DIRECTED, COMPUTER ASSISTED, PROBLEM SOLVING ENVIRONMENT. TO CONTINUE HOLD SHIFT AND HIT ENTER.*

PLACE YOUR COMMAND IN THIS LINE THEN HOLD SHIFT AND HIT ENTER*, 'YOUR COMMAND REQUIRES AN ARGUMENT. RETRY COMMAND SUPPLYING ONE.'

YOU ARE AT TOP LEVEL. SUPPLY A COMMAND OTHER THAN UP.*

YOU ARE AT BOTTOM LEVEL. SUPPLY A COMMAND OTHER THAN DOWN.*

FILE REQUESTED DOES NOT EXIST.*

INPUT NOT A RECOGNIZABLE COMMAND. IF UNABLE TO FIND ERROR, TYPE HELP*,

PROGRAMMING ENVIRONMENT REQUESTED DOES NOT EXIST ON THIS SYSTEM*.

/* MSG ARRAY HOLDS CAIMARK MESSAGES FOR DISPLAY */
KEY PICTURE '99.99.99',
(KKEY CHAR (8), (KEY1, KEY2 POS (4), KEY3 POS (7)) PIC '99')
DEF KEY,
KOY1 PICTURE '99.99.99', KKEY1 CHAR (8) DEF KOY1,
/* COMMAND ABBREVIATIONS (ORDER MATCHES LAB SWITCH) */
ON ERROR BEGIN;
ON ERROR SYSTEM;
FRAME ="ERROR IN PROGRAM. THE FOLLOWING ITEMS ARE ONCODE, ONSOURCE, AND ONKEY. TYPE RETURN, HOLD SHIFT & ENTER:||ONCODE||"||ONSOURCE||:||ONKEY;
GO TO FLASH3; END;
CALL DSP2260(FRAME, 10);
OPEN FILE(SYSPRINT) TITLE('SYSOUT'||TERMXX),
FILE(BOOK) TITLE('SYSIN' ||TERMXX);
READ FILE(BOOK) INTO(BUF) KEY(0);
GET STRING(BUF)EDIT(FNO,RNO)(2 F(4));
GET STRING(BUF)EDIT((INFO(J) DO J=1 TO FNO))
/* RETRIEVES DATA SET DIRECTORY INFO */
PUT STRING(SLAST)EDIT('ALL FILES AVAIL ON SYS LISTED BELOW. PLACE COMMAND
O FILEREQ NAMEOFFILE ON LINE2'',(NAME(J) DO J=1 TO FN0))
(A(80),36 A(20));
/* CREATES FILELIST DISPLAY */
ARG='INTRODUCT'; GO TO FREQ1;
/* A FILEREQ INTRODUCT COMMAND IS PROCESSED TO BEGIN SESSION */
FLASH: CALL SEARCH (KKEY,NUM,ERNO);
FLASH1: READ FILE (BOOK) KEY (NUM) INTO (FRAME);
FLASH2: IF ERNO=1 THEN KKEY=KKEY1; ELSE KKEY=KKEY1;
    SUBSTR(LINE1,73,8)=KKEY1; LINE2=MSG(ERNO);
FLASH3: CALL DSP2260(FRAME,2);
    DO I=1 TO 80 WHILE (L2(I)=' '); END;
    IF I>80 THEN DO; ERN0=3; GO TO FLASH2; END;
    DO J=I+1 TO 80 WHILE (L2(J)=' '); END;
    IPT=J;
    IF J=I+1 THEN II=INDEX(ABRV,L2(I));
    ELSE DO; STAND=SUBSTR(LINE2,I,J-I); II=(INDEX(OPS,STAND)+7)/8;
            END;
    /* COMMAND KEYWORD OR ABBREVIATION IS ISOLATED */
    GO TO LAB(II);
    /* BRANCH TO PROCESS CURRENT COMMAND */
    /* FILELIST */
FLIST: FRAME=SCREEN; GO TO FLASH3;
/* FILEREQ */
FREQ: CALL FINDARG(ARG);
/*SLOW LINEAR SEARCH FOR FILE - PILOT PROG ONLY. */
FREQ1: DO I=1 TO FN0;
    IF NAME(I)=ARG THEN DO;
        RGNO=REGNO(I ); EXT=EXTENT (I );
        READ FILE (BOOK) INTO (BUF) KEY (RGNO);
        GET STRING (BUF) EDIT ((KEYS(J), NOS(J) DO J=1 TO EXT-1))
           ((EXT-1)(A(8),F(4)));
        /* ESTABLISH ACTIVE FILE DIRECTORY */
        KEY=0; NUM=RGNO+1;
ERNO=1;
LINE1=(34)"**FILE**";
GOTO FLASH1; END; END;

ERNO=7; GO TO FLASH2;
/* VIEW */
VIEW: CALL FINDARG(ARG); KKEY=ARG; GO TO FLASH;
/* HELP */
HELP: ARG="HOW WORKS"; GO TO FREQ1;
/* RETURN */
RET: ARG="INTRODUCTION"; GO TO FREQ1;
/* UP */
UP: IF KEY3="00" THEN KEY3="00"; ELSE
IF KEY2="00" THEN KEY2="00"; ELSE DO; ERN0=5; GO TO FLASH2; END; /* CAN'T GO UP, ALREADY AT TO- */
GO TO FLASH;
/* DOWN */
DOWN: IF KEY2="00" THEN KEY2="01"; ELSE
IF KEY3="00" THEN KEY3="01"; ELSE DO; ERN0=6; GO TO FLASH2; END; /* CAN'T GO DOWN, ALREADY DOWN */
GO TO FLASH;
/* FORWARD */
FOR: IF KEY3="00" THEN KEY3=MOD(KEY3,99)+1;
ELSE IF KEY2="00" THEN KEY2=MOD(KEY2,99)+1; ELSE
KEY1=MOD(KEY1,99)+1;
GO TO FLASH;
/* BACK */
BACK: IF KEY3="00" THEN DO; KEY3=KEY3-1;
IF KEY3="00" THEN KEY3="99"; END;
ELSE IF KEY2="00" THEN DO; KEY2=KEY2-1;
IF KEY2="00" THEN KEY2="99"; END;
ELSE IF KEY1="00" THEN KEY1="99"; ELSE KEY1=KEY1-1;
GO TO FLASH;
/* OFF & SIGNON */
ON:OFF: GO TO STMT;
ERR: LINE2=MSG(8); GO TO FLASH3;
/* PROGRAM */
MET: CALL FINDARG(ARG); IF ARG="TOY" THEN CALL OWNST;
ELSE DO; ERNO=9; GO TO FLASH2; END;
GO TO RET;
FINDARG: PROC(ARG);
DCL ARC CHAR(8);
DO I=1 PT TO 80 WHILE (L2(I)='' ); END;
DO J=I TO 80 WHILE (L2(J)='' ) ; END;
IF J =I THEN DO; ERNO=4; GO TO FLASH2; END; /*COMMAND USED
REQUIRES ARGUMENT. SUPPLY ONE AND TRY AGAIN. */
ARG=SUBSTR(LINE2,I,J-I);
END FINDARG;
SEARCH: PROC(KEY,NUM,ERNO);
DCL KEY CHAR(8),(NUM,ERNO,LO,HI,MID) FIXED BIN;
LO=C; HI=EXT;
LOOP: DO I=1 TO LOG2(FLOAT(EXT-1))+1;
MID=(LO+HI)/2;
IF KEY<KEYS(MID) THEN HI=MID; ELSE
IF KEY>KEYS(MID) THEN LO=MID; ELSE
DO; NUM=NOS(MID); ERNO=1; RETURN; END;
END LOOP; ERNO=2;
END SEARCH;
STMT: END FILES;
APPENDIX B: INFORMATION FILES

TOYLANG
TABLE OF CONTENTS FOR THE TOYLANG FILE

Currently the TOY language file is comprised of the materials listed below:

00.00.00 TOY Reference Manual
01.00.00Toy Reference Manual
02.00.00 Formal Syntax Specification for TOY
03.00.00 Toy Exercises

To reference any of these units issue the command VIEW followed by the appropriate unit number. To submit a program in TOY issue the command PROGRAM TOY and you will be switched to the TOY Programming Environment. To return to the file information system from the Programming Environment issue the command RETURN; have fun playing with TOY.

TOY REFERENCE MANUAL

01.00.00 IntroducTion to TOY
01.02.00 Computation Statements
  01.02.01 Assignment
  01.02.02 Decision
  01.02.03 Transfer
  01.02.04 Input/Output
01.03.00 Control Statements

Type the command VIEW followed by the unit number of the section you wish to examine.
INTRODUCTION TO TOY

01.01.00

TOY IS A SMALL ARITHMETIC LANGUAGE DESIGNED FOR DEMONSTRATION PURPOSES. IT CONTAINS TWO TYPES OF INSTRUCTIONS, COMPUTATION AND CONTROL. ALL STATEMENTS (DELIMITED BY A SEMI-COLON) ARE SUBMITTED TO THE MACHINE ONE AT A TIME. AFTER PASSING THROUGH THE SYNTAX CHECKER, COMPUTATION STATEMENTS ARE APPENDED TO THE PROGRAM BEING ASSEMBLED. CONTROL STATEMENTS ARE IMMEDIATELY PERFORMED. PROGRAM EXECUTION IS REQUESTED BY THE CONTROL COMMAND EXECUTE. PROGRAMS REMAIN IN THE MACHINE MEMORY FOLLOWING EXECUTION. TO CLEAR THE MEMORY THE CONTROL COMMAND NEWPROG MUST BE GIVEN. A PROGRAMMING SESSION IS ENDED BY THE COMMAND RETURN; PROGRAMS ARE LIMITED TO 25 STATEMENTS.

TOY COMPUTATION STATEMENTS

01.02.00

THE TOY LANGUAGE PERMITS ONLY ONE DATA TYPE, INTEGER, AND NO DATA ARRAYS. VARIABLE NAMES MUST START WITH A LETTER AND CONSIST OF 1-8 LETTERS OR DIGITS. STATEMENTS ARE SUBMITTED ONE AT A TIME AND MUST BE DELIMITED BY A SEMI-COLON. STATEMENT LABELS ARE VARIABLE NAMES AND ARE SET OFF FROM THE STATEMENT BY A COLON. PROGRAMS ARE LIMITED TO 25 STATEMENTS. IF THE LIMIT IS EXCEEDED, YOU WILL BE AUTOMATICALLY RETURNED TO THE FILE SYSTEM.
ASSIGNMENT STATEMENTS

THE TOY ASSIGNMENT STATEMENT HAS THE GENERAL FORM

\[ \text{VARIABLE} = \text{EXPRESSION} ; \]

AN EXPRESSION CONSISTS OF A REASONABLE COMBINATION OF VARIABLES, CONSTANTS, THE OPERATORS +, -, *, / AND THE GROUPING SYMBOLS ( AND ) . THE USUAL HIERARCHY OF * AND / ABOVE + AND - HOLDS IN TOY. PARENTHESES MAY BE USED TO OVERRIDE THE HIERARCHY. VARIABLES MUST BE DEFINED ON THE LEFT SIDE OF AN ASSIGNMENT STATEMENT BEFORE THEY ARE USED ON THE RIGHT SIDE OF ANOTHER.

EXAMPLE:

\[ \text{ALPH} = 1253 + 15 \times (170/31 - A \times (BET - 1)) ; \]

DECISION STATEMENTS

THE TOY DECISION STATEMENT HAS THE GENERAL FORM

\[ \text{IF RELATION1 THEN STATEMENT1;} \]

RELATION1 CONSISTS OF TWO EXPRESSIONS SEPARATED BY ONE OF THE RELATION OPERATORS <, >, = . STATEMENT1 MAY BE AN ASSIGNMENT, A TRANSFER, OR AND INPUT/OUTPUT STATEMENT. IT MAY NOT BE ANOTHER DECISION STATEMENT. THE DECISION STATEMENT WORKS AS FOLLOWS, IF RELATION1 IS TRUE, STATEMENT1 IS EXECUTED; IF RELATION1 IS FALSE, STATEMENT1 IS NOT EXECUTED.

EXAMPLE:

\[ \text{IF } A > B \text{ THEN ALPH} = B \times (T\!+\!15) ; \]
TRANSFER STATEMENT

THE TOY TRANSFER STATEMENT HAS THE GENERAL FORM

GOTO LABEL1;

LABEL1 MUST BE A LABEL ON A STATEMENT IN THE PROGRAM. WHEN EXECUTED, THE TRANSFER STATEMENT CAUSES CONTROL TO PASS TO THE DESIGNATED LABELED STATEMENT. REMEMBER THAT STATEMENTS ARE LABELED BY PLACING A LABEL FOLLOWED BY A COLON PRECEDING THE STATEMENT.

EXAMPLES:

LOOP: A=A+1;
GOTO LOOP;

INPUT/OUTPUT STATEMENTS

THE TOY INPUT STATEMENT HAS THE GENERAL FORM

GET(VARIABLELIST);

WHEN A GET STATEMENT IS EXECUTED, THE VARIABLE NAMES IN THE VARIABLE LIST ARE DISPLAYED ON THE SCREEN AND THE USER IS REQUESTED TO GIVE VALUES FOR THEM, ONE BY ONE. WHEN ALL HAVE BEEN SUPPLIED, THE NEXT PROGRAM STATEMENT WILL BE EXECUTED

GET(ALPH,BETA,C);

THE OUTPUT STATEMENT USES THE KEYWORD PUT. WHEN A PUT IS EXECUTED, THE VARIABLES IN THE LIST AND THEIR CURRENT VALUES WILL BE DISPLAYED ON THE SCREEN, ONE AT A TIME. AFTER VIEWING THE INFORMATION, THE USER MUST PRESS ENTER TO PROCEED WITH PROGRAM EXECUTION.

PUT(ALPH,C,B);
THREE CONTROL STATEMENTS, EXECUTE, NEWPROG, AND RETURN, ARE AVAILABLE IN TOY. EACH MUST BE SUBMITTED AS A SEPARATE STATEMENT FOLLOWED BY A SEMI-COLON. ALL ARE COMPLIED WITH IMMEDIATELY WHEN ISSUED.

EXECUTE

THIS COMMAND CAUSES THE COMPUTATION PROGRAM CURRENTLY ASSEMBLED TO BE EXECUTED. THE PROGRAM IS NOT DESTROYED BY EXECUTION.

NEWPROG

THIS COMMAND CAUSES THE MEMORY TO BE CLEARED SO THAT A COMPLETELY NEW PROGRAM MAY BE SUBMITTED AND ASSEMBLED FOR EXECUTION.

RETURN

THIS COMMAND CAUSES CONTROL TO LEAVE THE TOY PROGRAMMING ENVIRONMENT AND RETURN TO THE FILE INFORMATION SYSTEM.

THE FORMAL SYNTAX FOR TOY IS SPECIFIED USING BACKUS-NAUR FORM NOTATION. SEMANTIC AND USAGE INFORMATION CAN BE OBTAINED FROM THE TOY REFERENCE MANUAL. TO VIEW THE SYNTAX TYPE DOWN, HOLD SHIFT, AND HIT ENTER.
<Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>::<S7Ip>:
FORMAL SYNTAX FOR TOY

<LABST>::=<VAR>;<ST>
<CONTROL>::=EXECUTE|RETURN|NEWPROG
<STATE>::=<LABST>|<CONTROL>
<SESSION>::=<STATE>|<SESSION>;<STATE>

TOY EXERCISES

A SET OF PROBLEMS TO HELP THE STUDENT BECOME ACQUAINTED WITH THE TOY LANGUAGE IS PRESENTED HERE.

03.01.00 PROB1  AN EXERCISE IN THE USE OF THE ASSIGNMENT AND INPUT/OUTPUT
03.01.01 SOLUTION TO PROB1

03.02.00 PROB2  AN EXERCISE IN THE USE OF THE DECISION AND TRANSFER STATEMENTS
03.02.01 SOLUTION TO PROB2

03.03.00 PROB3  AN EXERCISE IN THE PROGRAMMING TECHNIQUE OF LOOPING.
03.03.01 SOLUTION TO PROB3

SELECT THE PROBLEM YOU WISH TO WORK, TYPE VIEW AND THE UNIT NUMBER TO THE PROB.
WRITE A PROGRAM THAT WILL CALCULATE THE POLYNOMIAL \((3x^2-15x)x+7\) FOR ANY INTEGER VALUE OF \(x\). THE VALUE OF \(x\) SHOULD BE INPUT WITH A GET STATEMENT. THE ANSWER SHOULD BE OUTPUT WITH A PUT.

TO SWITCH TO THE TOY PROGRAMMING ENVIRONMENT TO TRY YOUR SOLUTION TO THIS PROBLEM TYPE PROGRAM TOY. TO SEE A SOLUTION TYPE DOWN.

```
GET(X);
ANSWER=(3*X*X-15*X)*X+7;
PUT(ANSWER);
EXECUTE;
```
WRITE A PROGRAM THAT WILL INPUT THREE INTEGERS AND OUTPUT THE LARGEST OF THE THREE.

TO SWITCH TO THE TOY PROGRAMMING ENVIRONMENT TYPE PROGRAM TOY .
TO SEE A SOLUTION TO THIS PROBLEM TYPE DOWN .

SO THAT THE ENTIRE PROGRAM MAY BE ON THE SCREEN AT ONCE, THERE MAY BE MORE THAN ONE STATEMENT ON A LINE. REMEMBER, HOWEVER, THAT THE PROGRAM MUST BE SUBMITTED ONE STATEMENT AT A TIME.

GET(NUM1, NUM2, NUM3);
IF NUM1>NUM2 THEN GOTO ONE; IF NUM2>NUM3 THEN GOTO OUT2;
OUT3: PUT(NUM3); GOTO OUT;
OUT2: PUT(NUM2); GOTO OUT;
ONE: IF NUM3>NUM1 THEN GOTO OUT3;
OUT1: PUT(NUM1);
OUT: END=0; EXECUTE;
WRITE A PROGRAM THAT WILL INPUT A NUMBER (GREATER THAN ONE) AND OUTPUT 1 IF THE NUMBER IS A PRIME AND 0 IF IT IS NOT.

TO SWITCH TO THE TOY PROGRAMMING ENVIRONMENT TYPE PROGRAM TOY.
TO SEE THE SOLUTION TO THE PROBLEM TYPE DOWN.

SO THAT THE ENTIRE PROGRAM MAY BE VIEWED AT ONCE MORE THAN ONE STATEMENT APPEARS ON A LINE. WHEN SUBMITTING, ONLY ONE STATEMENT CAN BE ON A LINE.
YES=1; NO=0; GET(NUM); TEST=2;
LOOP: IF TEST=NUM THEN GOTO OUTY;
   IF NUM/TEST*TEST=NUM THEN GOTO OUTN;
   TEST=TEST+1; GOTO LOOP;
OUTY: PUT(YES); GOTO OUT;
OUTN: PUT(NO);
OUT: END=0; EXECUTE;
HOW WORKS
T#XX HOWWORKS FILE 00.00.00

A GUIDE TO THE STUDENT CONTROLLED, COMPUTER ASSISTED, PROGRAMMING ENVIRONMENT.

THIS FILE IS COMPRISED OF THE FOLLOWING THREE UNITS.

01.00.00 INTRODUCTION TO THE SYSTEM
02.00.00 EXPLANATION OF THE COMMAND SET
03.00.00 EXAMPLES OF USE OF THE SYSTEM

PLEASE TYPE VIEW FOLLOWED BY THE UNIT NUMBER OF THE SELECTED TOPIC, HOLD THE
SHIFT KEY, AND HIT ENTER.

T#XX INTRODUCTION 01.00.00

THIS SYSTEM CONSISTS OF A COLLECTION OF INFORMATION FILES AND PROGRAMMING
ENVIRONMENTS. THE SYSTEM IS COMPLETELY CONTROLLED BY THE USER VIA A SMALL SET
OF COMMANDS. THESE COMMANDS PERMIT THE USER TO LEARN WHAT FILES EXIST IN THE
SYSTEM AND TO SEARCH THROUGH ANY OF THEM. THE USER ALSO CAN, VIA THE COMMAND
LANGUAGE, SWITCH INTO ANY OF THE AVAILABLE PROGRAMMING ENVIRONMENTS. UPON
ISSUING THE COMMAND RETURN HE CAN SWITCH BACK TO THE FILE HANDLING PART OF THE
SYSTEM. THIS ENABLES HIM TO WORK IN THE PROGRAMMING ENVIRONMENT YET HAVE READY
ACCESS TO THE INFORMATION FILES. SHOULD PROBLEMS OCCUR DURING PROGRAMMING,
ANSWERS SHOULD BE READILY AVAILABLE IN THE INFORMATION FILES.

TO PROCEED TO AN EXPLANATION OF THE COMMAND SET, TYPE FORWARD, SHIFT & ENTER.
A command is issued to the file system by typing a command word (delimited by at least one blank) in line 2, holding the shift key and depressing the enter key. The system will respond by executing the command or by presenting an error message in line 2. If a command requires an argument, the argument, preceded & followed by at least one blank, should be typed following the command. The system consists of a collection of files. Each file is organized as an outline would be, thus each unit in the file is given a unit number consisting of three portions, each of two digits, separated by periods. These numbers give a hierarchical order to the file. To use the system you must access a particular file and then a record within the file. Please type down, hold shift & enter.

These commands are three that should be remembered at all times.
OFF
   This command terminates your session at the terminal. It will be the last command you execute.
HELP
   This command will cause the beginning frame of the howworks file to be presented. This file (it is the one you are currently using) gives the rules for using the information file system.
RETURN
   This command will cause an introductory display containing a list of the system commands to be presented. The display should help to reorient you with respect to the system.
To see the rest of the command set type forward, hold shift, and hit enter.
FILELIST AND FILEREQ 02.02.00

FILELIST  THIS COMMAND PRESENTS A DISPLAY CONTAINING A LIST OF ALL THE FILES THAT EXIST WITHIN THE SYSTEM. ANY OF THESE FILES MAY BE ACCESSED BY ISSUING THE COMMAND FILEREQ.

FILEREQ ARG  THIS COMMAND ALLOWS YOU TO EXAMINE THE FILE WHOSE NAME IS GIVEN AS THE ARG. THE FILE NAMED MUST EXIST WITHIN THE SYSTEM.

THESE TWO COMMANDS ENABLE YOU TO DISCOVER WHAT FILES EXIST WITHIN THE SYSTEM AND TO ACCESS ANY ONE OF THEM.

TO SEE THE REST OF THE COMMAND SET TYPE FORWARD, HOLD SHIFT, AND HIT ENTER.

FORWARD AND BACK 02.03.00

EACH RECORD WITHIN A FILE IS IDENTIFIED BY A UNIT NUMBER CONSISTING OF THREE PORTIONS, EACH OF TWO DIGITS, SEPARATED BY PERIODS (EXAMPLE 02.01.00). THE COMMAND FORWARD CAUSES THE RIGHTMOST NON-ZERO PORTION OF THE UNIT NUMBER OF THE CURRENT DISPLAY TO BE INCREMENTED BY 1 AND THE RECORD WITH THIS NEW NUMBER TO BE DISPLAYED. IF RECORD 02.01.00 WERE CURRENTLY DISPLAYED, FORWARD WOULD DISPLAY 02.02.00. BACK HAS THE SAME EFFECT EXCEPT THAT THE RIGHTMOST NON-ZERO PORTION IS DECREMENTED, NOT INCREMENTED. HENCE IF RECORD 03.02.05 WERE CURRENTLY DISPLAYED AND THE COMMAND BACK WERE ISSUED, UNIT 03.02.04 WOULD BE DISPLAYED.

TO SEE THE REST OF THE COMMAND SET TYPE FORWARD, SHIFT & ENTER.
T#XX UP AND DOWN 02.04.00

THE COMMAND UP CREATES A NEW UNIT NUMBER AS FOLLOWS: THE RIGHTMOST NON-ZERO PORTION OF THE CURRENT UNIT NUMBER IS SET TO ZERO. THE RECORD WITH THIS NEW NUMBER WILL THEN BE DISPLAYED. THUS IF RECORD 03.04.00 IS THE CURRENT DISPLAY, UP WILL CAUSE RECORD 03.00.00 TO BE DISPLAYED. THE COMMAND DOWN WORKS AS FOLLOWS: A NEW UNIT NUMBER IS CREATED BY REPLACING THE LEFTMOST ZERO PORTION OF THE CURRENT NUMBER WITH 01. THE RECORD DESIGNATED BY THE NEW NUMBER IS THEN DISPLAYED. HENCE IF 05.02.00 WERE THE CURRENT RECORD, ISSUING THE COMMAND DOWN WOULD CAUSE RECORD 05.02.01 TO BE DISPLAYED.

TO SEE THE REMAINING COMMANDS TYPE FORWARD, HOLD SHIFT, AND ENTER.

T#XX VIEW AND PROGRAM 02.05.00

THE COMMANDS FORWARD, BACK, UP, AND DOWN ENABLE YOU TO DISPLAY FILE RECORDS RELATED WITHIN THE ORGANIZATIONAL SCHEME TO THE RECORD BEING DISPLAYED. THUS THEY permit YOU TO EASILY EXAMINE RECORDS THAT ARE CLOSELY RELATED TO THE RECORD BEING EXAMINED. THE COMMAND VIEW ARG LETS YOU ACCESS ANY RECORD WITHIN THE FILE CURRENTLY BEING EXAMINED. THE UNIT NUMBER OF THE DESIRED RECORD MUST BE SPECIFIED AS THE ARG OF THE VIEW COMMAND. EXAMPLE: VIEW 05.03.01

THE COMMAND PROGRAM ARG ENABLES YOU TO SWITCH INTO THE PROGRAMMING ENVIRONMENT SPECIFIED VIA THE ARG. FOR EXAMPLE, THE COMMAND PROGRAM TOY WOULD SWITCH YOU INTO THE TOY PROGRAMMING ENVIRON. TO RETURN FROM A PROG ENVIRON TYPE RETURN TO VIEW EXAMPLES OF USE OF THE COMMAND LANGUAGE TYPE VIEW 03.00.00 & ENTER.
AS AN EXAMPLE IN USE OF THE SYSTEM AND THE COMMAND LANGUAGE, LET US ASSUME THAT YOU WANT TO DO SOME PROGRAMMING IN TOY. YOUR FIRST COMMAND MIGHT BE PROGRAM TOY WHICH WOULD SWITCH YOU INTO THE TOY PROG ENVIRON. AFTER WORKING AWHILE YOU NEED TO CHECK THE REFERENCE MANUAL, SO YOU COMMAND RETURN. BACK IN THE FILE SYSTEM YOU ISSUE A FILELIST TO FIND THE NAME OF THE FILE YOU WANT. THEN YOU WOULD ISSUE A FILEREO TOYLANS. READING THE INTRODUCTORY DISPLAY OF THIS FILE YOU WOULD FIND THE UNIT NUMBER OF THE REF MANUAL AND ISSUE A VIEW 01.00.00. USING THE TABLE OF CONTENTS DISPLAY OF THE REF MANUAL, YOU COULD GET THE INFO YOU SEEK ABOUT I/O STATEMENTS BY COMMANDING VIEW 01.02.04. YOU RETURN TO TOY VIA THE COMMAND PROGRAM TOY. PLEASE TYPE DOWN, HOLD SHIFT, AND HIT ENTER.

WHEN YOU WERE THROUGH PROGRAMMING IN TOY, YOU WOULD TYPE RETURN. THIS WOULD RETURN YOU TO THE FILE SYSTEM. AT THAT TIME YOU COULD ACCESS ANY FILE IN THE SYSTEM, SWITCH TO ANOTHER PROG ENVIRON, OR END YOUR SESSION. TO END YOU WOULD TYPE OFF, HOLD THE SHIFT KEY, AND HIT ENTER. NOTE THAT YOU MUST FIRST RETURN TO THE FILE SYSTEM TO TERMINATE THE SESSION.

YOU SHOULD NOW BE READY TO TRY THE SYSTEM OUT FOR YOURSELF. TO REVIEW THE INTRODUCTION TO THE SYSTEM TYPE HELP. TO SEE A LIST OF THE COMMANDS TYPE RETURN. TO FIND WHAT FILES ARE AVAILABLE TYPE FILELIST. TO TERMINATE THE SESSION TYPE OFF. GOOD LUCK.