FINAL REPORT

NASA Grant Number NGR 05-020-~~457~~ 547

Principle Investigator Richard C. Atkinson
Professor of Psychology

Report Date June 1, 1974

Institute for Mathematical Studies in the Social Sciences
Stanford University
Stanford, California

N74-26704

FINAL REPORT

NASA Grant Number NGR 05-020-~~457~~ 547

Summary

This report describes work conducted under NASA Research Grant NGR 05-020-~~457~~547.

The first part of the report presents an overview of the areas explored under the grant; several topics relevant to the practical applications of computer-assisted instruction are discussed. The remaining sections present more specific details on the work conducted under the grant.

The goal of the project was research and development on strategies for optimizing the instructional process, and dissemination of information about the applications of such research to the instructional medium of computer-assisted instruction. Accomplishments included construction of the author language INSTRUCT, construction of a practical CAI course in the area of computer science and a number of investigations into the individualization of instruction, using the course as a vehicle.

The instructional system and the curriculum were used extensively by students from NASA installations, universities, and junior colleges. The course and the method of instruction have been received very enthusiastically.

There has also been a substantial amount of spin-off from the project. INSTRUCT has been used to develop four different curricula in widely varied fields of study. The first application was the AID course described above, and the second was a closely related course teaching the fundamentals of

programming in the BASIC language to inner-city high school students with low reading ability and no background in algebra. The INSTRUCT system was modified slightly to allow it to be used in a program of grammar instruction designed for junior high school deaf students. And finally, the U. S. Public Health Service used the INSTRUCT system in developing a set of review-test lessons in dental health.

Both the instructional system and the various curriculums have been the subject of a number of studies and revisions, aimed at providing more effective instruction. In particular, data were gathered on individual student performance in order to evaluate some basic questions related to learning theory, among them the issues of student vs. program control over the selection and pacing of curriculum, relative merits of multiple-choice vs. constructed response formats, and differences in student performance on items requiring different strategies for solution, e.g., algebraic formulation of problems as compared to translation of algebraic expressions into a programming language.

An abstract of each section of the report follows.

I. Friend and Atkinson. Computer-assisted instruction in programming:AID.

Research in learning theory and instructional strategies has received a new impetus in recent years from technological developments in the field of computer design. Computer-assisted instruction, entirely unknown ten years ago, is evidence of the rapid growth of computer applications in education and is already producing profound effects in the individualization of instruction. Since January, 1963, the Institute for Mathematical Studies in the Social Sciences has been conducting extensive programs of research and development in computer-assisted instruction.

In 1968, the Institute received funding from NASA to design and produce a course in programming using computer-assisted instruction as the instructional medium. The course was to be tutorial in nature and sufficiently self-contained so that students could use it without being supervised by an experienced teacher of programming, and it was to be suitable for use by NASA personnel. Developmental work started in the summer of 1968. A preliminary version was completed by February, 1969, and consisted of a coding language, a set of 20 one-hour lessons written in the coding language, and a set of programs to interpret coded lessons and to interact with students using standard teletypes as student stations.

A description of the course, some preliminary results from its pilot period of development, and a discussion of computer programs and coding languages are presented.

II. Friend, Fletcher, and Atkinson. Student performance in computer-assisted instruction in programming.

An instructional system for teaching the Algebraic Interpretive Dialogue (AID) programming language to college-age students, two control programs (one for presenting instructional material and one for interpreting students' AID productions), and data collected by the two control programs are described. The first 21 lessons of the course and classification of the lesson exercises are also described. Data based on student daily reports are presented and discussed. Item analyses of data gathered by the instructional program, including stepwise linear regression models of item difficulty and analyses of selected data collected by the interpreting program are presented and discussed.

The following were among the results of this investigation: Although first response errors were often those of AID syntax, these errors were easily corrected in subsequent responses, and, in general, the syntax of AID commands were easily mastered; although students easily learned the mechanics of the instructional system, they rarely used features that allowed student control of instructional content and sequence; algebraic formulation of problems appeared to be more difficult than transforming algebraic expressions into AID commands; students had the greatest difficulty understanding hierarchy of arithmetic operations, use of functions, and the execution sequence of AID commands.

III. Fletcher and Schulz. Providing software support for computer-assisted instruction.

The burgeoning use of computer-assisted instruction has left many system managers with the problem of appropriately modifying their software for CAI. In maintaining system support for CAI at Stanford University and in consulting with managers of commercial time-sharing systems, a set of common issues arises whenever the question of CAI comes up. These issues are not necessarily peculiar to CAI, but they receive greater emphasis in CAI time sharing. The intent of this paper is to document these issues and to make specific suggestions for adopting extant time-sharing systems for CAI.

IV. Friend. Computer-assisted instruction in programming: A curriculum description.

The course provides an introduction to computer programming for community college students who have taken high school algebra, and it is equivalent to a three quarter-unit course in computer science. All instruction is presented by computer, and a supplementary student manual is provided for reference. The course includes the topics of stored programs, use of variable, input and output control, syntax of algebraic and logical expressions, use of functions and subroutines, conditional clauses, and branching techniques. The instructional system implements under computer control teaching strategies that might be used by a human tutor such as individualizing the content, pace, and sequence of instruction, allowing for sufficient student control, tailoring wrong answer messages, and providing both remedial and extra-credit work. Students are required to interact "on-line" with a commercially prepared editor-interpreter similar to those found in many timesharing environments. Performance data from both the instructional program and the editor-interpreter are automatically stored for later retrieval and analysis. The organization of the course, types of exercises used, and content of each lesson are documented and an appendix lists the concepts associated with each exercise in the course.

V. Friend. 100 programming problems.

These 100 problems were prepared for students taking the computer-assisted course "Introduction to AID Programming," an introductory course in algebraic programming. Also included is a brief reference manual for the programming language AID.

The problems include applications of elementary programming techniques to a variety of fields such as elementary arithmetic and algebra, geometry, linear algebra, probability and statistics, consumer and business problems, and calculus. Most of the problems require no mathematical background beyond high school algebra, but some of the problems in linear algebra, statistics, and calculus will not be readily solved by students who do not have some acquaintance with fundamental concepts such as the sigma notation, solution of linear equations by determinants, limits of sequences and series, statistical correlation, and standard deviation. In all cases the necessary formulas are given.

These problems are not specific to the programming language AID, but can also be used in introductory courses in BASIC, FORTRAN, ALGOL, APL, etc.

VI. Beard, Lorton, Searle, and Atkinson. Comparison of student performance and attitude under three lesson-selection strategies in computer-assisted instruction.

Three problem selection strategies (student selection, program selection weighted by past performance, and forced selection independent of student history) were compared in a CAI course in computer programming. Various measures of aptitude, performance, and attitude were examined. No consistent difference was observed among the three groups. The results are discussed in terms of the specific experiment and the general problem of curriculum design for comparing path selection strategies. Continuing experimentation is described.

COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING: AID

by

Jamesine Friend and R. C. Atkinson

January 25, 1971

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

7

TABLE OF CONTENTS

N74-26785

COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING: AID*

Jamesine Friend and R. C. Atkinson

Stanford University
Stanford, California 94305

I. Computer-assisted Instruction in Programming

Research in learning theory and instructional strategies has received a new impetus in recent years from technological developments in the field of computer design. Computer-assisted instruction, entirely unknown ten years ago, is evidence of the rapid growth of computer applications in education and is already producing profound effects in the individualization of instruction. Since January, 1963, the Institute for Mathematical Studies in the Social Sciences has been conducting extensive programs of research and development in computer-assisted instruction.

In 1968, the Institute received funding from NASA to design and produce a course in programming using computer-assisted instruction as the instructional medium. The course was to be tutorial in nature and sufficiently self-contained so that students could use it without being supervised by an experienced teacher of programming. Supplementary material, such as manuals and a syllabus of readings in computer sciences, was to be supplied as part of the package.

The course was to be suitable for use by NASA personnel, and the feasibility of using the course as part of their training program was to be investigated. It was assumed that students would be at about the junior college level with no experience in mathematics beyond high school algebra and with no previous introduction to computer programming.

Work on the development of the course started in the summer of 1968. A preliminary version of half of the course was completed by February, 1969, and consisted of a coding language, a set of 20 one-hour lessons written in

*This research was supported by NASA Research Grant NGR-05-020-244.

the coding language, and a set of programs to interpret coded lessons and to interact with students using standard teletypes as student stations.

In the spring of 1969, about 15 students took the course. Performance data were collected (by hand) and summarized, and students were closely observed and interviewed after each session. The curriculum materials and necessary computer programs were revised and extended on the basis of data and observations of student reactions. The revised course is now complete and in use by NASA personnel. Data are being collected and analyzed.

The first decision made in the development of an introductory course in programming was what programming language to teach. Programming languages designed expressly for teaching purposes were not considered, since we felt that users of the course would benefit more from learning a language with immediate practical application, even if the language was initially more difficult to learn; for this same reason we felt that the language should be one that is widely available rather than one that is implemented on only a few computers, or only on computers produced by one manufacturer. Also, we anticipated that most students would eventually be working in an engineering or scientific environment and would have more need for an algebraic language such as FORTRAN than for a list-processing language such as LISP or a business-oriented language like COBOL.

The programming languages considered included FORTRAN, ALGOL, BASIC, and AID. For a first course, BASIC and AID are both excellent choices, because they are considerably simpler than either FORTRAN and ALGOL; nevertheless, they contain all of the structure needed to illustrate the basic principles of programming. AID[1] (Algebraic Interpretive Dialogue) is a high-level algebraic programming language with extensive interactive (or "conversational") abilities. This language is an adaptation for the PDP-10

[1] See PDP-10 AID Programmer's Reference Manual, Digital Equipment Corporation, Maynard, Massachusetts, 1968.

computer of JOSS,[2] a language developed by RAND Corporation for use by scientists, engineers, etc., who needed a powerful, easy-to-learn tool capable of performing complex algebraic tasks. A number of other minor variants of JOSS, such as CAL and FOCAL, are now implemented on a variety of computers. A complete description of AID will be found in the appendices. BASIC,[3] which was developed at Dartmouth as an elementary algebraic language for beginning students, is now widely implemented and is probably better known than AID (partly because all implementations use the same name). BASIC is somewhat more powerful than AID in its matrix manipulation commands, but AID has more power in recursively defined arithmetic functions. The greatest advantage of AID over BASIC, FORTRAN, or COBOL is that AID is not a compiler, but an interpreter with a large number of direct commands, which the student can begin to use the first day rather than having to delay hands-on experience until after he has learned the concept of a stored program and the necessary formats. These interactive capabilities are a great asset to a student just learning a programming language since they provide a kind of immediate reinforcement that cannot be supplied by a compiler. All in all, it was felt that AID had a slight edge as a beginner's language, but the final deciding factor was that AID had already been implemented for the PDP-10 computer we would be using, whereas BASIC would not be available to us for several months. Since that time, we have obtained a BASIC compiler and have completed a high school course in BASIC using the same structure and programs developed for the AID course.

[2] See Mark, S. L. and Armerding, G. W., The JOSS Primer. The RAND Corporation, Santa Monica, California, August, 1967; Shaw, J. S., JOSS: Experience with an Experimental Computing Service for Users at Remote Typewriter Consoles. The RAND Corporation, Santa Monica, California, May, 1965.

[3] See Kemeny, John G. and Kurtz, Thomas E., BASIC, Dartmouth College Computation Center, 1968.

II. Description of the Course "Introduction to Programming:AID"

The course consists of a set of 50 lessons, about one hour in length, plus summaries, reviews, tests, and extra-credit problems. A student manual, which includes instructions for operating the instructional program and a glossary of terms used in the course, has been prepared and is included in the appendices of this report. The course is equivalent to a three-unit junior college course.

The computer-assisted instruction and supplementary manual constitute a completely self-contained course. The lessons are tutorial in nature, that is, no previous knowledge of computers or programming is necessary. The only prerequisite for the course is a good background in algebra, as supplied by three semesters of high school algebra.

Computer-assisted instruction is given to the students by means of standard Model-33 teletypewriters, located in remote training centers, which will communicate with the PDP-10 computer located at Stanford by means of ordinary telephone lines. The problems are typed on the student's teletype by the computer and the student responds by typing his answers on the same teletype. After the computer analyzes the student's response, the student is informed as to whether his response was correct or incorrect, then he is given additional instruction and asked to respond again, or he is given a different problem.

The course does not require the supervision of a trained teacher of programming, but a one-day teachers' workshop should be given to acquaint teachers with operating procedures and to provide them with an overview of the content of the course.

Although the course is ordinarily used on a regularly scheduled basis in a college environment or training center, it is also well suited for individual use as an on-the-job training course for people working in association with a computer facility. Use by individuals can be on a nonscheduled basis or on a flexibly scheduled basis, since there are few time restrictions on the use of the computer; some students might prefer to spend several hours a day on the course, with the possibility that they could complete the course within a few weeks rather than distributing their lessons over several months.

The 50 lessons cover the following fundamental concepts of programming and the use of computers.

(1) An interactive time-sharing executive system.

(2) An interpreter.

(3) Concept of a stored program.

(4) Debugging techniques.

(5) Labels and variables.

(6) Loops.

(7) Input and output.

(8) Computer storage, including both core and disk.

(9) Subroutines.

(10) Recursive functions.

(11) List sorting and table look-up routines.

The student is required to write and debug at least 50 programs, several of which are major programs for solving difficult algebraic problems. An outline of the course is found in the appendices.

Each lesson covers one basic concept, varying in length from 50 to 200 problems and requiring about one hour for an average student to complete. A lesson contains three sections: a core lesson, a summary, and a review. Selected lessons contain an additional extra-credit section. The core lesson contains about 20 to 30 problems that present the concept and supplies some practice problems. At the end of the core lesson there is an optional summary of the lesson; the summary is typed in an 8 1/2" x 11" format, which the student can save as a permanent reference. Following the summary, there is an optional review section, which is divided into several parts, one for each idea presented in the core lesson, so that the student may review only that part of the lesson that he did not completely understand. The review problems, like the problems in the core lesson, are tutorial, not merely additional practice and present the ideas afresh from a different point of view. After the review section, there may be a short section of optional extra-credit problems; these are usually programming problems, which are much more difficult than the programming problems given in either the review or the core lesson. Most of the extra-credit problems require considerable thought and time, and the student is not expected to complete them during a current session, but may, instead, submit them at any time before the end of the course. Extra-credit problems are not supplied with each lesson, but there are at least 50 such problems in the entire course, and the teacher may wish to require some of these problems as homework assignments, or he may use them as tests.

After each group of five lessons, there is an optional self-test designed to help the student evaluate his understanding of the concepts presented to date. Since this test is designed for student's use and not for grading purposes, no report on student performance will be available to the teachers. Following the self-test, there is a general overview lesson that reminds the student of what has been taught and informs him which of the topics already covered are essential to the subsequent material. During the overview lesson, the student is given the opportunity to review entire lessons, or any individual topics from preceding lessons.

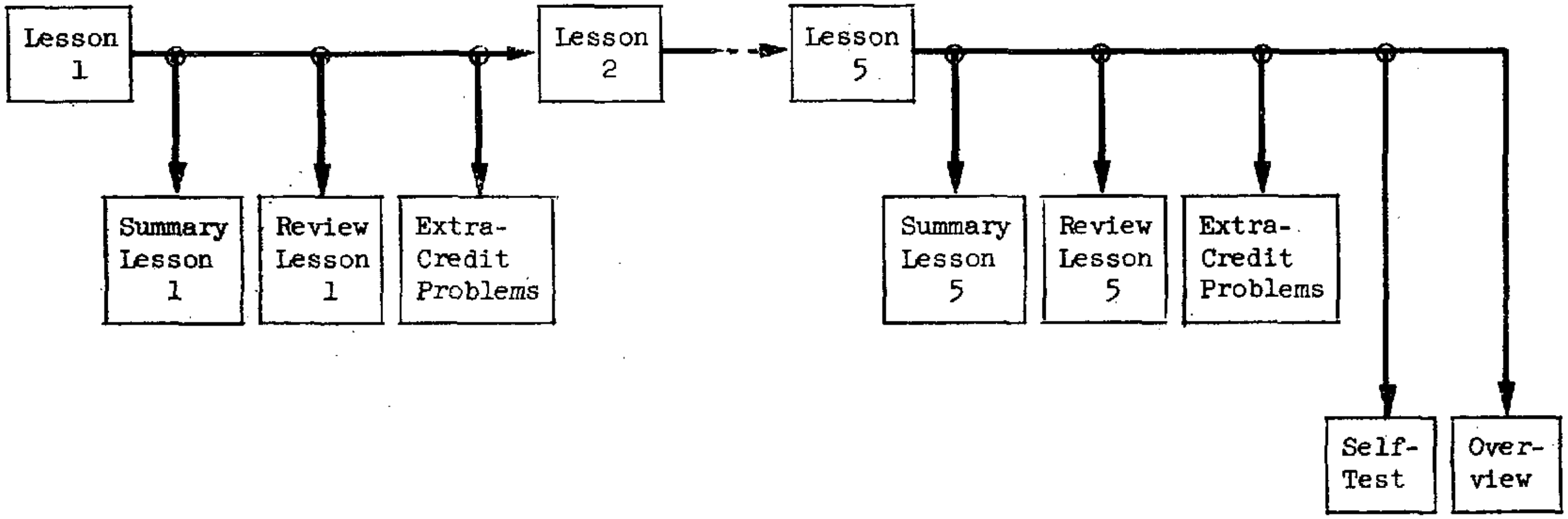
The structure of the course is illustrated in Figure 2 by a block diagram of a set of five lessons (with summaries, reviews and extra-credit problems), followed by a self-test and an overview lesson.

Before discussing details of the instructional strategy, we give a few examples of student interaction with the instructional program, starting with the first problems in Lesson 1. On a student's first day, he is given a student manual and seated at a teletype connected by telephone lines to the PDP-10 computer at Stanford. Following the instructions in the student manual (see the manual in the appendices), he signs on and starts the instructional program, which automatically starts at Lesson 1, Problem 1, for a new student. The program prints each problem in turn, then prints an asterisk to indicate to the student that he can respond, and awaits his response before proceeding. In the following typical sequence, the student's responses are marked by an asterisk at the left.

```
L1-1:                  INTRODUCTION TO PROGRAMMING
                          BY JAMESINE E. FRIEND

IN THE FIRST LESSON YOU WILL LEARN HOW TO USE THIS PROGRAM.

AFTER YOU TYPE YOUR ANSWERS YOU MUST PRESS THE RETURN KEY.
CAN YOU FIND THE RETURN KEY?
```



One lesson block is shown, including 5 lessons (with Summaries, Reviews, and Extra-Credit Problems), Self-Test, and Overview Lesson.

Student decision points are marked ⊕ .

Optional lessons are shown below the main line.

Figure 1. Structure of Course

```
*YES
GOOD.  DON'T FORGET TO PRESS THE RETURN KEY AFTER YOU TYPE YOUR ANSWERS.


L1-2:  WRONG ANSWERS ARE NOT COUNTED.  YOU ALWAYS GET ANOTHER CHANCE
IF YOUR ANSWER IS WRONG.

WHAT DOES THE COMPUTER PRINT WHEN IT IS READY FOR YOUR ANSWER?
   A.  AN EXCLAMATION POINT  !
   B.  A QUESTION MARK  ?
   C.  AN ASTERISK  *

TYPE "A", "B", OR "C".  (DON'T FORGET THE RETURN KEY.)

*C
CORRECT


L1-3:  IF MULTIPLE-CHOICE PROBLEMS HAVE MORE THAN ONE CORRECT ANSWER,
YOU MAY LIST THE CORRECT CHOICES IN ANY ORDER.

SUPPOSE B, C, AND D ARE THE CORRECT CHOICES FOR A PROBLEM.  WHICH OF
THESE WOULD BE CORRECT WAYS TO ANSWER?
   A.  D, B, C, A
   B.  B, D, C
   C.  B, C, D
   D.  D, B, C

*B
YOU HAVEN'T FOUND ALL OF THE ANSWERS YET.  START AGAIN.

*BCD
CORRECT
```

Lesson 1, a short introduction to the teaching program, continues by explaining to the student the various problem formats he will encounter in the course and gives him an opportunity to practice each of these formats. The second lesson begins by teaching the student how to start and stop the AID interpreter and how to use a few simple, direct commands to solve arithmetic problems.

```
L2-1:                           LESSON 2
                        USING AID FOR ARITHMETIC

IN THIS COURSE YOU WILL USE TWO DIFFERENT PROGRAMS:
   1.  THE TEACHING PROGRAM.  YOU ARE USING THE TEACHING PROGRAM NOW.
       THE TEACHING PROGRAM WILL TEACH YOU TO WRITE PROGRAMS USING THE
       AID LANGUAGE.
   2. THE AID INTERPRETER.  AFTER YOU LEARN TO WRITE AID PROGRAMS, YOU
      WILL USE THE AID INTERPRETER TO TRY OUT YOUR PROGRAMS.

YOU WILL ONLY BE ABLE TO USE ONE OF THE PROGRAMS AT A TIME SO YOU HAVE
TO KNOW HOW TO STOP A PROGRAM AND START ANOTHER.

WHICH PROGRAM ARE YOU USING NOW?

*THE TEACHING PROGRAM
CORRECT


L2-2:

HOW TO START THE AID INTERPRETER:
   FIRST, STOP THE TEACHING PROGRAM (TYPE CTRL-C).
   SECOND, TYPE "L AID" AND THE RETURN KEY.

HOW TO STOP THE AID INTERPRETER:
   TYPE CTRL-C.

AFTER THE TEACHING PROGRAM IS STOPPED, WHAT SHOULD YOU TYPE TO START
THE AID INTERPRETER?

*CTRL-C
WRONG.  TYPE A QUESTION MARK FOR A HINT OR CTRL-T FOR THE ANSWER.

*?
READ THE SECOND STEP UNDER "HOW TO START THE AID INTERPRETER."

*L AID
CORRECT


L2-3:  WHICH COMMAND WILL STOP THE AID INTERPRETER?

A.  CTRL-H
B.  CTRL-T
C.  CTRL-C
D.  CTRL-G

*C
CORRECT
```

After some practice in starting and stopping the AID interpreter, the TYPE command is introduced and the student practices using commands like

```
TYPE  15 + 249
TYPE  76 - 3 + 42
```

Lesson 2 also introduces the symbols * and / for multiplication and division.

```
L2-10:  AID SYMBOLS FOR ARITHMETIC OPERATIONS:

          +  ADDITION
          -  SUBTRACTION
          *  MULTIPLICATION
          /  DIVISION

WHICH COMMANDS WILL CAUSE AID TO MULTIPLY 3 by 4?

A.  TYPE (3)(4)
B.  TYPE 3 X 4
C.  TYPE 3 * 4
D.  TYPE 3/4
E.  TYPE 3*4

*A
WRONG

*C
YOU HAVEN'T FOUND ALL OF THEM.  START OVER.

*CE
CORRECT


L2-11:  WHICH COMMAND WILL CAUSE AID TO MULTIPLY 25 BY 5 AND DIVIDE
        BY 3?

A.  TYPE 25 X 5/3
B.  TYPE 25 * 5/3
C.  TYPE 25(5/3)
N.  NONE OF THE ABOVE

*B
CORRECT
```

At the end of each lesson, the student is asked if he wants a summary of the lesson to save as a permanent reference. The summaries are printed in 8 1/2 x 11" format, so that they may be punched and put in a loose-leaf note book. The following summary of Lesson 2 is typical.

```
                    SUMMARY OF LESSON 2
                  USING AID FOR ARITHMETIC

1.  TO START THE AID INTERPRETER, TYPE
      L AID

2.  TO STOP THE AID INTERPRETER, TYPE
      CTRL-C
```

```
3.  THE "TYPE" COMMAND
      ...STARTS WITH THE WORD "TYPE"
      ...THEN A SPACE
      ...THEN AN ALGEBRAIC EXPRESSION
      ...ENDS WITH A RETURN.

      YOU TYPE:       AID ANSWERS:
      ---------       ------------
      TYPE 2+4        2+4 =     6
      TYPE 42/4       42/4 =    10.5
      TYPE 6*1.2      6*1.2 =    7.2

4.  THE SYMBOLS FOR ARITHMETIC OPERATIONS:
      +    ADDITION
      -    SUBTRACTION
      *    MULTIPLICATION
      /    DIVISION
```

After the summary is printed (if the student requests it), the student is asked if he wants to review any of the concepts covered in the lesson. The review, which is about the same length as the lesson, does not cover topics sequentially as in the original presentation, but is instead organized into independent sections, once for each concept so that the student may review only the parts of the lesson that he wishes; also, the student is told which topics are important to ensuing lessons, so that he knows where to concentrate his effort. Here, for example, are a few problems from the review of Lesson 4 (note that the symbol ↑ is used to denote exponentiation, i.e., 5↑2 means 5^2).

```
R4-1:                         REVIEW OF LESSON 4
                       EXPONENTS AND SCIENTIFIC NOTATION

WHICH OF THESE TOPICS DO YOU WANT TO REVIEW NOW?
(BE SURE YOU KNOW THE STARRED TOPICS.)

*A.  EXPONENTS
 B.  USING 0 AND 1 AS EXPONENTS
*C.  ORDER OF ARITHMETIC OPERATIONS
 D.  USING FRACTIONAL EXPONENTS TO FIND ROOTS
*E.  NEGATIVE EXPONENTS
*F.  READING SCIENTIFIC NOTATION
 G.  WRITING SCIENTIFIC NOTATION
 N.  NONE

*C
```

```
R4-17: IF AN EXPRESSION HAS EXPONENTIATION AND ALSO SOME OTHER OPERATION,
SUCH AS MULTIPLICATION, DO THE EXPONENTIATION FIRST.

TO FIND THE VALUE OF
   4*5↑2
DO 5↑2 FIRST, THEN MULTIPLY BY 4.
WHAT IS THE VALUE?

*100
CORRECT


R4-18: DO EXPONENTIATION BEFORE ADDITION, SUBTRACTION, MULTIPLICATION
OR DIVISION. FIND THE VALUE OF EACH EXPRESSION.

50 - 7↑2
*1
CORRECT

3↑3 - 20
*11
WRONG

*-11
WRONG

*7
CORRECT
```

In general, students are expected to have had some previous work with algebra, but it is not assumed that the level of skill is high, or that a student will remember such concepts as the use of zero as an exponent, or the definition of "positive" as contrasted with "non-negative." All such topics are reviewed at appropriate times for the student who needs a refresher. For example, Lesson 15, which introduces the IF clause, reviews relations between numbers in the context of introducing new symbols.

```
L15-1:                          LESSON 15
                          RELATIONS, "IF" CLAUSES

SYMBOLS USED FOR RELATIONS:
   <    FOR "LESS THAN"
   >    FOR "GREATER THAN"
   =    FOR "EQUALS"
   #    FOR "NOT EQUALS"
   <=   FOR "LESS THAN OR EQUALS"
   >=   FOR "GREATER THAN OR EQUALS"
```

```
TYPE THE SYMBOL FOR
"GREATER THAN OR EQUALS"
*>=
CORRECT

"NOT EQUAL"
*#
CORRECT

"LESS THAN"
*<
CORRECT
```

```
L15-2:  RELATIONS BETWEEN NUMBERS CAN BE SHOWN ON A NUMBER LINE.

   -3   -2   -1    0    1    2    3    4
----.----.----.----.----.----.----.----.----
    ↑         ↑                   ↑
    X         Y                   Z

ANY NUMBER TO THE RIGHT OF 2 IS GREATER THAN 2.

ANSWER TRUE OR FALSE (T OR F):
X > 2   *F
Y > 2   *F
Z > 2   *T
X > Y   *F

ANY NUMBER TO THE LEFT OF 2 IS LESS THAN 2.
ANSWER T OR F:
X < 2   *T
Y < 2   *T
Z < 2   *F
Z < X   *F
```

After reviewing the relations between numbers, Lesson 15 proceeds to teach the use of conditional commands using the algebraic notation just introduced.

```
L15-10:  WHICH MEANS "Q IS NON-NEGATIVE"?

   A.  Q > 0

   B.  Q >= 0

   C.  Q < 0

   D.  Q <= 0

   N.  NONE
```

```
*B
CORRECT

L15-11:  NOW THAT YOU KNOW ABOUT THE RELATIONS = # < > <= AND >= I WILL
SHOW YOU HOW TO USE THEM IN AID COMMANDS.

ANY AID COMMAND CAN BE MODIFIED BY AN "IF" CLAUSE.
EXAMPLES:
   SET Z = 2 IF X < 10.
   TYPE X IF X < 0.
   DO PART 5 IF M = N.

COMPLETE THIS COMMAND SO THAT Y = X↑2 IF X IS POSITIVE.
SET Y = X↑2 IF X....0

*>
CORRECT

L15-12:  THE NEXT FEW PROBLEMS ARE ABOUT THESE COMMANDS.
   22.1    SET Y = X IF X > 0.
   22.15   SET Y = -X IF X < 0.
   22.2    SET Y = 0 IF X = 0.
   22.25   TYPE Y.
   DO PART 22 FOR X = -3.15.

AFTER STEP 22.25 WHAT WILL AID ANSWER?

*-3.15
WRONG

*?
LOOK AT STEP 22.15.  IF X IS NEGATIVE, WHAT IS THE VALUE OF Y?

*3.15
CORRECT
```

Although there is considerable variation in the sequence of problem types within a lesson and in the style of presenting new concepts, the general scheme is to review any necessary algebraic ideas, present new symbols and new commands, give sample programs using the new commands, and then give programming problems that can be solved by using the ideas just introduced. The following is an example of a programming problem from Lesson 15.

```
L15-20:  WRITE A PROGRAM THAT WILL PRINT "SAME" IF ALL THREE NUMBERS X,
Y AND Z HAVE THE SAME SIGN.  THE PROGRAM SHOULD PRINT "DIFFERENT" IF THE
NUMBERS DO NOT ALL HAVE THE SAME SIGN.

BEFORE YOU START, TELL ME WHAT YOUR PROGRAM SHOULD PRINT IF X = -2, Y = 3
AND Z = 1?

*DIFFERENT
CORRECT

WHAT SHOULD YOUR PROGRAM PRINT IF X = -2, Y = -3 AND Z = -1?

*DIFFERENT
WRONG

*SAME
CORRECT

L15-21:  OK.  GO AHEAD AND WRITE THE PROGRAM TO FIND OUT IF THE 3 NUMBERS
ALL HAVE THE SAME SIGN.  TEST YOUR PROGRAM FOR THESE VALUES OF X, Y AND Z.

X = 2     Y = 3     Z = 15
X = 2     Y = 3     Z = 0
X = -5    Y = -3    Z = -1
X = -5    Y = -3    Z = 0
```

At this point the student is expected to stop the teaching program and to use the AID interpreter to write and debug his program. When he has completed the program to his satisfaction, he starts the teaching program again.

```
WHAT ANSWER DID YOU GET FOR THE LAST PART?

*DIFFERENT
EXCELLENT

DID YOU USE ANY "DEMAND" COMMANDS IN YOUR PROGRAM?

*NO
YOU COULD HAVE SAVED YOURSELF SOME TYPING IF YOU HAD STARTED THE PROGRAM
WITH THESE COMMANDS:

   DEMAND X
   DEMAND Y
   DEMAND Z
```

The student may request additional information or suggestions about how to write the program either before or after he tries to produce the program. If the student cannot solve the problem, even using the additional help, he is shown a correct solution to the problem and is asked to study it carefully, and to copy and execute it.

There are over 50 programming problems in the course. Many lessons also supply extra-credit programming problems such as the following.

```
X15-1:                 EXTRA-CREDIT PROBLEMS FOR LESSON 15

1.  WRITE A PROGRAM THAT WILL TYPE "1" IF THREE NUMBERS, A, B, AND C, ARE
    DECREASING IN SIZE (I.E. IF A IS LARGEST, B IS NEXT, AND C IS SMALLEST).
    IF A, B, AND C ARE NOT DECREASING, THE PROGRAM SHOULD TYPE "0".

2.  WRITE A PROGRAM THAT WILL TYPE "1" IF B IS BETWEEN A AND C; TYPE "0"
    OTHERWISE.  (NOTICE I DID NOT SAY WHETHER A WAS LARGER OR SMALLER
    THAN C).
```

In the first few programming problems, the program and the values to be used for variables are specified in complete detail, and the student is thoroughly quizzed about the performance of his program. As the course develops, the student is supplied with less and less complete specifications, and he is encouraged to analyze the instructions and to experiment with different solutions. Also, he is gradually given the responsibility for determining whether his program is correct, both in the sense of debugging and in the sense of providing a solution to the stated problem. The aim is not only to encourage analytic ability and creative thinking, but also to introduce the student to the idea that working programmers spend most of their creative effort in defining the problem (and, in many cases, deciding whether there is a problem). Further, they spend much of their programming time satisfying themselves that they have produced a correct program.

Little has been said so far about how a student interacts with the teaching program, and how the teaching program is designed to provide individualized instruction. In order to explain these things, we give some details of the teaching strategy.

One of the basic requirements of a tutorial course is to provide for individualization of instruction, with the aim of optimizing the learning process. The course "Introduction to Programming," which is being developed under NASA Contract NGR-05-020-244, is designed as an application of the results of numerous studies in the techniques of optimizing learning. The variety of optimization routines used in the course and the consequent richness of the curriculum material have never before been attempted in a course of comparable length or scope.

The logic of branching used within problems permits extremely fine discriminations between student responses and thus provides a mechanism for remediation that is appropriate, not only to the specific problem, but also to the specific student response; i.e., gross discrimination of "correct" and "incorrect" are not used as the basis for deciding upon appropriate remediation, as is ordinarly done in drill-and-practice material or in linearly programmed courses. Fine discriminations can also be made between correct responses so that the "correctness" function ranges over a set of positive as well as negative numbers, and the program responds differentially to categories of correct as well as incorrect responses. The analysis of student responses is made by means of twelve basic analysis routines; each of these routines can return from 2 to 4 different values of the correctness function. Furthermore, the analysis routines can be used in any Boolean combination to increase the number of possible values in the range of the correctness function. The maximum size of the range of this function, i.e., the maximum number of correct-incorrect classifications for a given problem, has not yet been fully exploited, since it is limited only by the size of the core buffers in the computer, but we estimate it to be in the neighborhood of 100. Since the probability of receiving a wide variety of distinguishable incorrect responses to a given problem is extremely low, the current course is designed to use from three to ten values for the correctness function, depending upon the content of the problem. Because the system can respond differentially to the students, each problem takes on the aspect of a small "dialogue" between the computer and the student.

The optimization scheme described above is not, however, the only one used in the course. A second major scheme allows the student to initiate the dialogue. In the microbranching logic, the student is allowed two different devices for requesting additional information. The first of these is the HINT command, which may be given by the student at any time simply by typing a question mark. The instructional system provides an unlimited number of hints for each problem; in the current course, two hints are provided for most problems, and as many as six are provided for particularly difficult problems. By allowing for optional additional instruction, we keep the pace fast enough for the more intelligent, better prepared students without penalizing those students whose mathematical background is less adequate.

In addition to the HINT command, there is also a TELL command that may be used by students at any time. This command causes the computer to print the correct answer (or <u>a</u> correct answer) to the problem, providing that such text was coded for the problem, and then branches to the next problem in sequence. In the current course, sample correct answers are provided for about 80 percent of the problems. There is some evidence that adult students learn adequately without being required to make overt responses, so students are in no way penalized, and, in fact, are encouraged to request the answer whenever they wish. In general, the problems that do not include sample correct answers are those for which there is no correct answer ("Do you want a summary of this lesson?"). Two or three representative answers may be provided for problems that have many correct answers.

To illustrate the kind of hints and answers provided in the course, we show a few problems from Lesson 8, with comments.

```
L8-2:  WHAT WILL AID ANSWER?
   LET M(X) = 3*X
   TYPE M(10)

*10                                    [The student makes an incorrect
WRONG                                   response.]

*?                                     [He types a question mark to get
  M(X) = 3*X                            additional information.]
           ↑
SUBSTITUTE 10 FOR X TO FIND M(10)

*30                                    [He then gives the correct answer.]
CORRECT
```

```
SUPPOSE YOU HAD ALREADY GIVEN THE ABOVE COMMANDS, AND
THEN GAVE THIS ONE.  WHAT WOULD AID ANSWER?
   TYPE M(2)

*20                                     [The student gives a wrong answer.]
WRONG

*6                                      [...and corrects himself without
CORRECT                                  additional instruction.]

...AND WHAT WILL AID ANSWER TO THIS COMMAND?
   TYPE M(1)

*3                                      [The student makes a correct response
CORRECT                                  on his first try.]


L8-3:  WHAT WILL AID ANSWER?
   LET Q(X) = 2*X + 7
   TYPE Q(5 - 3)

*?                                      [The student requests a hint.]
     Q(X) = 2*X + 7
              ↑
SUBSTITUTE (5 - 3) FOR X.

*?                                      [He asks for another hint.]
  Q(5 - 3)
= Q (2)
= 2*(2) + 7
= ???

*                                       [...and finally requests the answer
11                                       (by typing Ctrl-T, which is an
                                         invisible character).]
```

The optimization routines described thus far are used in all problems in "Introduction to Programming." An additional scheme is also used for problems that require the student to write and debug a program. Since such problems are necessarily more complex than the kind used in most programmed instruction, there is also a greater need for more highly differentiated remedial material. For each programming problem, a sequence of problems was designed to test the student's understanding of the concepts involved. Additional hints are also available.

Although the most complex of the optimization routines are used within problems, provision is also made for optimization at the lesson level. The number of problems that constitute a lesson for a particular student is dependent upon the responses of that student; for example, in Lesson 3, a student may do only 30 problems, or he may do as many as 74, including the problems in the associated remedial lesson. Further, after every five lessons there is an overview of the preceding material; these lessons consist of five sections (one for each of the preceding lessons), with optional detailed review. Each overview lesson is preceded by an optional self-test, which the student may use to evaluate his progress and which provides him with a basis for deciding which of the sections in the subsequent overview lessons are appropriate.

One indicator of the richness of the curriculum provided by the procedures described above is the number of different messages that can be used in the course of a single lesson; in Lesson 3, for example, one student may see 60 different messages, while another student may see as many as 400. The number of responses required of a student is also an indicator of the richness of the curriculum; for Lesson 3 (to use the same example), only 30 responses are required of the good student, but a student who is giving some incorrect responses and requesting much of the optional material may make as many as 200 responses (there is actually no upper limit, since a student may make any number of incorrect responses per problem).

Notice that a recurring theme in the optimization schemes is the provision for student control. There are strong indications from past research, both in computer-assisted instruction and elsewhere, that the participation of the student in decisions about his course of study significantly affects the rate of learning. The study of motivation in an environment of computer-assisted instruction has not yet been approached in any very rigorous way, but preliminary results do indicate that some factors here may completely overwhelm others in an experimental design. Since curriculum design cannot always wait on firm research results, provision was made in the instructional system for nine student control commands (including the HINT, TELL and GO commands as well as single-character and full-line erase commands, quick sign-off, etc.). These control commands are defined by the coder and may be left undefined if desired. Thus if further testing of the system indicates that there should be less student control, the scheme can be easily modified.

As an illustration of the use of the optimization schemes, a coded problem taken from Lesson 4 is attached as an appendix. There is a top-level problem, followed by eight subproblems which are used as remediation for students who are having difficulty with the concept of hierarchy of operations. The top-level problem requires the student to evaluate the expression

5 * 2↑3

(In the AID programming language, an asterisk is used as the symbol for multiplication, and an up-arrow is used as the symbol for exponentiation, so the expression 5×2^3 would be written 5 * 2↑3 in AID.) If the student does not understand the precedence of exponentiation over multiplication, he will produce the incorrect response "1000" and will then be given the message "Wrong, AID would evaluate 2↑3 first. Try again." If the student produces the correct response (40), he is given the standard correct-answer message CORRECT and then goes to the next top-level problem (Lesson 4, Problem 6), bypassing all of the following subproblems. For the student who fails to produce the correct answer, an algebraic derivation of the correct answer is given, and the student goes to the first subproblem. The first four subproblems lead the student through the evaluation of the expression

32/4↑2

and the fifth subproblem requires the student to evaluate, without detailed help, the expression

10↑3 * 2.

If the student succeeds, he bypasses the remaining subproblems and proceeds to the next top-level problem. The last three subproblems are written for students who are having considerable difficulty with the concept; these last three problems present the concept from a different viewpoint and provide the student with a workable algorithm for solving problems of this type.

The entire sequence of subproblems is tutorial; few remedial sequences in the course consist solely of additional practice without amplification of the ideas. The necessary drill on the concepts presented in the course is attained by introducing the concepts in such a sequence that immediate practice is provided in the context of presenting the next concept. Thus, necessary skills are constantly reinforced without the need for extensive sections of pure drill-and-practice.

III. Preliminary Results

The complete teaching system described above is now in use by NASA personnel, and has been used by a small number of volunteer students from Stanford University and Woodrow Wilson High School in San Francisco, but results are not yet available. The preliminary system, which formed the basis for the present system, was used by ten students in the spring of 1969 and subsequently by another half-dozen who sought out the curriculum designer to request use of the course. The results were extremely encouraging; student motivation was high, performance was good, and in all respects, the preliminary system proved itself both in overall philosophy andin curriculum design. An excerpt from the April-June 1969 progress report is given here.

"A small pilot study was designed during the Spring Quarter, 1969, to supply information for meaningful revisions of the curriculum and the instructional system. Since this was the first trial of the system, the most useful information would be derived from observations of students' reactions to the program. There was no plan to collect detailed data or to do any kind of statistical analysis of data. Ten students were enrolled in the course on a flexible time scheduling basis; some students were scheduled three sessions a week, others two, and others came only once a week, depending upon the wishes of the individual students. The students were allowed to use the course in whatever way they felt best; but they were restricted to taking not more than two lessons per session. Also, immediately after each session, they were to be interviewed briefly.

"The students completed anywhere from three to twenty lessons each, with about half of them getting as far as Lesson 20. In general, the students who did fewer lessons did so because they spent less time on the lessons rather than because of any great difficulties with the material. In fact, the student who had the most difficulty with the course, and made the slowest progress in relation to the time spent, finished Lesson 13 by the end of the quarter and expressed regret that he hadn't been able to spend enough time to have completed the 20 available lessons.

"Students were timed on several lessons in order to get a rough idea of the time which would be necessary for future students to complete the course. The average time per problem for different students ranged from about one minute per problem to three minutes per problem; the assignments for each lesson required about as much time as the lesson itself. [In the preliminary version of the course, programming problems were given as additional assignments rather than being incorporated in the lessons as they are now.]

"Extensive notes were taken during interviews with the students and were summarized in an anecdotal weekly report. Also, the responses to individual problems were tabulated and the percentages of correct and incorrect responses were calculated. The most frequent incorrect response to each problem was also tabulated.

"The students were quite enthusiastic about the course and would have worked for several hours at a time had they not been restricted to taking no more than two lessons per session. Since most of the students' comments were about specific problems, there was no indication that a major revision of the curriculum is needed. The following are a few general observations based on students' comments and behavior.

"Use of student controls. The student control commands, which were explained in detail in Lesson 1, were received with enthusiasm. (A control command is given by holding down the 'CTRL' key while striking a letter key.) The commands used were

Ctrl-H (used to request a hint) [This has been changed to a question mark in the newer version.]

Ctrl-T (used to request the answer)

Ctrl-S (skip to next problem) [This control command is available, but not stressed in the revision.]

Ctrl-G (used to get another problem or lesson. After the student types Ctrl-G he is asked to specify the lesson and problem he desires.)

"Both Ctrl-H and Ctrl-T were used frequently, although there was noticeable tendency for students to use one or the other but not both. Ctrl-S was rarely used; in fact, several students were asked, at the end of Lesson 3, what control commands were available and were not able to recall Ctrl-S.

"Ctrl-G was used much less than anticipated. At the end of the pilot study, the students were queried about this; several students replied that they thought they would not be contributing fully to the experiment (the pilot study) if they skipped any of the lessons; a few students felt that they would not know what they had skipped and that it might be important to them in later lessons (this comment was made even in reference to reviews and self-tests in which there was an explicit statement that no new material would be presented and that it was perfectly acceptable to skip the entire lesson); only one student consistently chose to review previous lessons and he commented that he felt he simply repeated the same mistakes without achieving any noticeable gain in understanding.

"*Language confusion*. Almost all students evidenced some confusion between the language they were learning (the AID programming language) and the language (English) used in the exposition. Part of this confusion undoubtedly arose because the AID language is a subset of English (AID commands are syntactically correct English sentences containing a verb, ending with a period [the newer version of AID does not require a period], etc.); although this is certainly not a complete explanation and it is obvious that the advantages of teaching an English-based programming language far outweigh the disadvantages even if it could be shown to be a significant factor in the language confusion.

"Furthermore, a few students were also puzzled about which program they were using--the teaching program or the AID interpreter (which they used for doing assignments); one student tried to ask the AID interpreter for hints about an assigned programming problem. It is felt that some confusion between languages and between programs is almost inherent in the situation and no satisfactory way of dispelling the confusion has been found.

"*Constructed responses to multiple-choice problems*. The multiple-choice problems used in the course consist of a problem statement or question and a list of possible answers, each of which is labeled with a letter. For example,

WHICH OF THESE ARE CORRECT AID COMMANDS?

A. TYPE 2 × 3.

B. PRINT 2 * 3.

C. TYPE 2 * 3.

N. NONE OF THE ABOVE.

"Students are expected to respond by typing a letter (or list of letters) corresponding to the correct answer (or answers).

"There is a noticeable tendency for students to respond to certain multiple-choice problems by typing the answer itself rather than typing the corresponding letter. In the AID course, a response other than a single letter (or list of letters) is treated as an error, and the message

PLEASE TYPE LETTERS ONLY

is given. This error message has been found to be remarkably ineffective; the probability that a student will repeat the same kind of error after receiving the above error message seems to be greater than one half, possibly as much as three quarters.

"The tendency to make the kind of error described above seems to be influenced by the following factors: [Note: the following remarks were based on observations and suggest future lines of research.]

"1. Answer length. If the number of characters in the answer choice is small (say, two to six characters), there is a strong tendency to type the answer itself.

"2. Context. If the problem is preceded by several problems requiring constructed responses, the tendency to construct a response is somewhat increased. If the preceding constructed responses are closely related to the choices in the multiple-choice problem, there is an even stronger tendency to construct a response; for example, if the six preceding problems require 3-digit numbers as a response, and the choices in the multiple-choice problem are 3-digit numbers, there is a high probability of making an error.

"3. Problem-solving strategy required. There seem to be two distinct kinds of problem-solving strategies used in producing the answer to a multiple-choice problem. One is a 'mental construction' of the correct answer, followed by a search of the choices for that answer, and the other kind is a 'feasibility-elimination' approach in which the student inspects the list of possible answers and chooses that which is most feasible, or eliminates those choices which are least feasible. (Generally, students working on a specific problem will not switch from one strategy to another unless there is a compelling reason; for instance, a student will abandon a 'feasibility-elimination' approach if several choices are equally feasible.) The strategy a student uses is influenced by the problem statement although there is some tendency for individual students to prefer one strategy over another. If the 'mental construction' strategy is used, the student is more likely to produce an overt construction of the answer, thereby producing an 'error.'

"4. Wording used in problem statement. The wording used in instructions to the student seems to have some effect on the tendency to give a constructed answer to a multiple-choice problem. In particular, use of the word 'what' in the problem statement produces more errors than the word 'which.' For example, compare 'What command causes AID to give N a value of 12?' with 'Which command causes AID to give N a value of 12?'

"One additional comment: Although the above remarks may imply that the error of constructing a response in answer to a multiple-choice question is a use-mention error, this may not be the case. There are a number of problems in the course which require a 'partial construction' and there is an observable tendency in students to give a more complete answer than is required; for example, students tend to answer 'Do Part 12' rather than 'Do' in response to this problem:

COMPLETE THIS COMMAND TO EXECUTE PROGRAM 12.

..... PART 12

"The error of constructing a more complete response than required is clearly not a use-mention error, and it seems to be closely related to the error of constructing a response to a multiple-choice problem.

"Answer length, context, required strategy, and wording used in the problem statement are not the only factors which contribute to the kind of use-mention error under consideration here; there are also individual factors, such as age and previous experience. However, the above four factors are the only curriculum-oriented factors which seem to have an effect."

Starting in the summer of 1969, extensive revision of the curriculum and programs was undertaken. The major changes were the provision for multiple hints (in the first version, there was only one hint per problem) and the provision of a multiple-strand structure to provide for review lessons, summaries, and extra-credit problems. The coding language and programs were extended considerably. As mentioned before, detailed results are not available, but all indications are that the revision is extremely successful; both students and teachers were enthusiastic.

IV. Computer Programs and Coding Language

One of the major efforts of the AID project has been in the development of a suitable coding language and a manual explaining the use of that coding language. The necessity for developing a coding language became apparent quite early in the planning stage of the system, since no available high-level language suitable for implementing the kind of optimization schemes was envisioned. The coding language, INSTRUCT, developed for this project, was designed to be learned and used easily by inexperienced coders and writers. Further, the manual, which includes a complete description of the instructional system, is written for readers who are unfamiliar with computers and programming. There are step-by-step instructions on coding, processing, and debugging lessons, as well as instructions for initializing a course, and for defining additional coding commands. The coding commands are summarized in a separate section, so that the manual can serve as a reference source as well as a primer. One of the major reasons for producing such a complete coding manual was to provide an adequate basic document for the instructional system should it be implemented on another computer for use in other places. The manual, which contains 90 pages, cannot be included in its entirety in this report, but excerpts containing a summary of op codes and a BNF definition of the language are included in the appendices. An example of a coded problem sequence (taken from Lesson 4) is also included.

Briefly, the coding language is a high-level computer language designed specifically for writing tutorial computer-assisted instruction. The language contains over 30 different types of commands, such as problem statement commands, response analysis commands, conditional branching commands, that enable a curriculum coder to specify problem statements, hints, sample answers, detailed analyses of student responses and contingent actions to be taken, sequence of problems, and format of all messages.

In order to provide programmed lessons that are highly individualized, there must be nontrivial routines for analyzing student responses and performing appropriate actions contingent upon the results of such analyses. Analysis routines must be highly differential so that specific errors may be isolated and appropriate remedial material presented. A simple correct-incorrect classification of responses is insufficient for an individualized, tutorial system of teaching. There are twelve basic analysis routines: EXACT, KW, EQ, MC, TRUE, YES, and their negations NOTEXACT, NOTKW, NOTEQ, NOTMC, FALSE and NO. The EXACT routine checks the student response for an exact character-by-character match with a coded text string; KW (key word) checks for the occurrence of a coded key word; TRUE checks for a response of TRUE or T; the MC (multiple-choice) routine can be used for multiple-choice problems in which several choices are correct (a correct response may be a list of all correct choices, or a list of a minimum number of correct choices, depending upon how the MC command is used by the coder); the EQ routine checks for a number within a range of numbers, as specified in the coding, or checks for equality with a single number, also as specified in the coding.

The basic analysis routines not only check on the correctness of a student response, they also check on the form of the student response. For example, the EQ routine accepts as a response any number in integer form, decimal form, or scientific notation; any response not in an acceptable form, e.g., a response of the word "four," elicits an error-in-form message: ERROR IN FORM: PLEASE TYPE A NUMBER. Another routine that differentiates between correctly formed and incorrectly formed responses, as well as between correct and incorrect responses is TRUE, which expects either TRUE or T as a correct answer, and either FALSE or F as an incorrect answer. Any other response from the student elicits an error-in-form message: PLEASE ANSWER TRUE OR FALSE. Most other analysis routines (YES, MC, etc.) also contain error-in-form subroutines.

Complex analyses of student responses can be made by using simple Boolean combinations of the basic analysis commands. For example, the coder may specify a check for a number between 1 and 10, but not equal to either 5 or 5.5, by using appropriate combinations of EQ and NOTEQ commands.

Since most of the action performed by the analysis routines is internal, i.e., with no action visible to the student, there are also commands that cause coded messages to be relayed to the student, appropriate branching to take place, etc. These commands, called "action commands," are all contingent upon the results of the analyses performed by the analysis commands, i.e., the actions are contingent upon the correctness of the student response.

In addition to the problem coding described above, the system also allows the coder to specify the number of strands, which of the student control commands are to be made available, and the characters to be used by the student for giving such commands. As a labor-saving device, about 15 "standard messages" can be defined by the coder so that he is not required to code commonly used messages (such as CORRECT, WRONG, TRY AGAIN) more than once.

Because all problems are written in a high-level coding language, any changes needed in the curriculum for research purposes are easily accomplished.

The teaching system described above is implemented for the PDP-10 computer located at the Computer-based Laboratory, operated by the Institute for Mathematical Studies in the Social Sciences of Stanford University. The teaching system consists of a coding language, a lesson processor program that will translate from the coding language into machine-readable code, a lesson interpreter that will interpret the translated code at the time a student is using the system, and a set of auxiliary operational programs. The lesson processor is essentially a compiler for the lesson coding language and is used to translate coded lessons into a form that can be stored efficiently for later use by the lesson interpreter. The program (the lesson interpreter) that will be in operation at the time a student takes a lesson is the most important and largest program in the teaching system. It is a time-sharing program that must be extremely efficient both in terms of core space required and in terms of processing time, since both of these factors affect the response time for all users of the system. Past experience has shown the length of response time as the single, most critical item of concern in the design of a system for computer-assisted instruction. A response time of less than 3 seconds is most desirable, and a response time of more than 10 seconds is totally unacceptable. Response time is affected both by the efficiency of the processing done by the program and by the total size of the program. For these reasons, the lesson interpreter is carefully designed and written in the most efficient available programming language. The auxiliary operational programs include a student enrollment program and a daily teachers' report program.

The lesson processor. The lesson processor is a two-stage processor, the first stage being one of the PDP-10 assemblers. Since the PDP-10 has a macro-assembler, full advantage has been taken of the macro capabilities; the processor consists almost entirely of macro definitions of the op codes used in the coding language, plus a very short load routine, which stores the processed lessons on a disk file (the processor is essentially a zero-length program). The coder is also allowed the advantages of a macro assembler; judicious use of macros can reduce coding time significantly.

The lesson interpreter. The interpreter is written as a reentrant time-sharing program using 2K words (36 bit) of core plus 1K for each of the students concurrently taking lessons. The program is written in one of the assembler languages for the PDP-10. Great care has been taken to ensure fast response time and economical use of core and disk storage. Routines for detecting and compensating for coding errors have been incorporated. In a similar fashion, unexpected responses from students are not allowed to cause errors in the program. This program has been in daily operation for as long as 10 hours per day since the first of February and is operating well; response time is excellent and no bugs have been found in the program. During the month of March, the lesson interpreter handled 1,050 lessons in BASIC and AID without any failures, a more than adequate demonstration of the abilities of the program.

As the students interact with the program, their individual history file is continually updated and written into disk storage. The history file is 100 words long and contains the student's name, the number of the course in which he is enrolled, his current position on each strand (lesson and problem number), the date, and various other information needed by the program. These history files supply information for auxiliary programs such as the daily report program; a sample daily report is included in the appendices. The data found in the individual history files, which are continutally updated as the student progresses through the course, are the only data collection currently done by the program.

The AID interpreter. The course "Introduction to Programming: AID" requires the student to learn to operate two programs that are completely independent: the lesson interpreter (instructional program) and the AID interpreter. The AID interpreter is a commercial program supplied and maintained by Digital Equipment Corporation, the manufacturer of the PDP computers. No changes have been made to date in the AID interpreter for data collection or any other reason, and there is no interrelation between AID and the instructional system other than that it is being implemented on the same computer.

APPENDIX A

Student Manual

INTRODUCTION TO PROGRAMMING: AID

Student Manual

by

Jamesine E. Friend

Revised March, 1971

Institute for Mathematical Studies in the Social Sciences
Stanford University
Stanford, California

TABLE OF CONTENTS

Note: Not all teletypes have the same set of characters.
For shift N, "↑" is equivalent to "∧".
For shift O, "←" is equivalent to "-".

In this manual, ↑ and ← are used; if appropriate, read ∧ and - for these characters.

How to Start the Teaching Program

In this course, you will be taking computer-assisted instruction in programming. The programming language you will learn is called "AID" and the lessons will be given by the PDP-10 computer at Stanford.

Follow these instructions to start the teaching program:

1. Turn on the teletype: the switch on the front of the teletype must be turned to the LINE position.

2. Push the START or BREAK key. (If the teletype doesn't start to hum, get help.)

3. Type a space. The computer will then type

```
     HI
     PLEASE TYPE YOUR NUMBER AND NAME
```

   (If this doesn't happen, get help.)

4. Type Q, your number, your first name, and a space. After you type the space following your first name, the computer should print your last name.

5. If your last name is printed correctly, type a space. (If it isn't, get help.) Then the computer will print the time, the date, and your teletype number.

6. Type

```
     L  INST
```

   and then push the RETURN key. The computer will type

```
     WHERE TO?
```

7. Type the RETURN key.

Steps 1, 2 and 3 are used to establish communication with the computer. Steps 4 and 5 cause you to be "signed on." Steps 6 and 7 start the teaching program.

If the computer does not respond correctly after each step, get help.

Good luck!

How to Stop

When you are through for the day, follow these instructions:

1. Hold down the CTRL key while you type the letter C.
   The computer will print a period.

2. Type the letter K, then push the RETURN key.
   The computer will print the sign-off message.

You do not have to turn the teletype off. It will turn off by itself.

Outline of the Course

Computer-Assisted Instruction in Programming: AID

Lesson 1. How to answer. How to erase. Control commands.

Lesson 2. Signing on and off AID. The TYPE command. Arithmetic operators: + - * / . Decimal numbers.

Lesson 3. Using AID for arithmetic. Use of parentheses. Order of arithmetic operations.

Lesson 4. The operator ↑ for exponentiation. Order of operations. Scientific notation.

Lesson 5. Variables. The SET command. Re-defining variables. The DELETE command used to delete variables.

Lesson 6. Self-test.

Lesson 7. Review.

Lesson 8. The LET command (using function notation). Distinction between LET and SET. Distinction between use of a defined function and display of the formula for a function. Re-defining and deleting functions.

Lesson 9. Some standard AID functions: IP(x), FP(x), SGN(x), SQRT(x).

Lesson 10. Indirect steps.
DO STEP ....
DO STEP ... FOR ....
Re-defining steps and deleting steps.
TYPE STEP ....

Lesson 11. Parts.
DO PART ....
DO PART .... FOR ....
Deleting parts.
TYPE PART.

Lesson 12. The DEMAND command.
DO PART ..., ... TIMES.
Termination by refusal to answer a DEMAND command.

Lesson 13. Self-test.

Lesson 14. Review.

Lesson 15. Relations between numbers.
Relational symbols: < > <= >= = #
Number line.
The IF clause.

Lesson 16. Branching. The TO command.
TO STEP ...
TO PART ...

Lesson 17. Traces.

Lesson 18. The indirect use of DO.

Lesson 19. How to write and debug a program.

Lesson 20. Self-test.

Lesson 21. Review.

Lesson 22. The FORM statement.

Lesson 23. Loops.

Lesson 24. Loops with variable bounds.

Lesson 25. Loops compared with FOR clauses.

Lesson 26. Loops with a DEMAND command.

Lesson 27. Self-test.

Lesson 28. Review.

Lesson 29. Absolute value.

Lesson 30. Trigonometric functions: SIN(x), COS(x).

Lesson 31. EXP(x), LOG(x).

Lesson 32. Lists.

Lesson 33. Using loops with lists of numbers.

Lesson 34. Self-test.

Lesson 35. Review.

Lesson 36. Nested loops.

Lesson 37. Iterative functions: SUM, PROD, MAX, MIN.

Lesson 38. Arrays.
LET S BE SPARSE.

Lesson 39. More about lists and arrays.

Lesson 40. Conditional definition of functions.

Lesson 41. Self-test.

Lesson 42. Review.

Lesson 43. Recursive functions.

Lesson 44. AND, OR and NOT.
Truth tables.

Lesson 45. TV(x). The function FIRST.

Lesson 46. LET used to define propositions.

Lesson 47. More standard AID functions.

Lesson 48. Self-test.

Lesson 49. Review.

47

GLOSSARY

Absolute value

The absolute value of a number is the size of that number disregarding the sign of the number. In AID, exclamation points are used to denote absolute value:

Examples:

```
!-2.7! = 2.7
!2.7!  = 2.7
```

See Lesson 29. Also see Operational Symbols.

****

AID

AID is the computer programming language being taught in this course. "AID" stands for Algebraic Interpretive Dialogue. See AID Interpreter.

****

AID commands

All AID commands have a similar form.

Each command must be on one line and must end with a RETURN. The form of the commands is as follows:

1. An optional step number, like 2.1 or 37.54 or 16.165.
2. A verb such as TYPE, SET, DELETE.
3. An argument whose form depends upon the preceding verb. The argument for TYPE is an algebraic expression:
   ```
   TYPE X + 2/Y
   ```
   The argument for SET is an equation with a single variable on the left of the equal sign:
   ```
   SET C = 72/B + 3.134
   ```
   Etc.
4. An optional IF clause.
   ```
   TYPE X + Y IF Z < 0
   SET Q = 3 IF P = 15
   DO PART 3 IF X < 27
   ```

In addition to the above four parts, certain commands may contain FOR clauses, or IN FORM clauses.

The AID commands taught in this course are

DELETE	Lessons 5,11
DEMAND	Lessons 12,26
DISCARD	Lesson 19
DO	Lessons 10, 11, 12
FILE	Lesson 19
FORM	Lesson 22
LET	Lesson 8
RECALL	Lesson 19
SET	Lesson 5
TO	Lesson 16
TYPE	Lesson 2
USE	Lesson 19

See Direct Steps, Indirect Steps

****

AID functions

AID functions are the functions already defined by AID.
These functions are

ARG, COS, DP, EXP, FIRST, FP, IP, LOG, MAX, MIN, PROD,
SGN, SIN, SQRT, SUM, TV, XP.

Each of these functions is separately defined in the glossary.
See Lessons 9, 30, 31, 45 and 47.

****

AID Interpreter

The AID Interpreter is the program used when you want AID to solve a problem for you. After you start AID, you can type any AID commands. The AID Interpreter interprets your commands and executes them. To start the AID Interpreter (after you are signed on), type CTRL-C then type "L AID".

To stop the AID Interpreter, type CTRL-C.

To stop a runaway AID program, type CTRL-C twice.

****

AND

"AND" is a logical operator used in propositions. All elements connected by "AND" must be true for the entire expression to be true. If any one element is false, the expression is false.
Examples: Assume A = TRUE, B = TRUE, C = FALSE

```
X = A AND B              X = TRUE
Z = A AND B AND C        Z = FALSE
```

See Lessons 15 and 44. Also see Proposition.

****

Answer, How to

To answer a problem in the teaching program type your answer, then type the key labeled "RETURN." For multiple-choice problems, there may be more than one correct answer; you may type the letters in any order (with spaces or commas between them, if you wish), for example,

```
ABC
CBA
A, C, B
B C A
```

For TRUE-FALSE questions, you may type "T" for "TRUE" and "F" for "FALSE." For YES-NO questions, you may type "Y" for "YES" and "N" for "NO."
See Lesson 1.

****

Answer, How to Get

To get the correct answer to a problem, hold down the "CTRL" key while you type the letter "T" (for "Tell me the answer").

****

ARG

ARG(x,y) is the argument function. AID finds the angle between the +x axis of the x,y plane and the line joining 0,0 and x,y. The result is in radians.

****

Arithmetic symbols

See Operational symbols

****

Array

An array is a set of numbers identified by a single letter and from 1 to 10 subscripts (indices). The subscripts may be any integers from -250 to 250.

Examples:

The following are all members of the same array A:

```
A(-10,2,5) = 2.789
A(-10,1,0) = -45
A(1,20,59) = 0
```

You can set all undefined members of an array (for example X) to be 0 with this command:

```
LET X BE SPARSE.
```

See Lessons 38 and 39. Also see List.

****

Asterisk (*)

Both the teaching program and the AID Interpreter print an asterisk when ready for a response from the user. The asterisk is also used as the multiplication symbol (6 * 7 means 6 times 7).

****

Base

(See also Exponent, Exponential function.) In an exponential function the base is the number multiplied by itself as often as specified.

Example:

X is the base: X↑2 = X*X

The base may be either a number or a variable.

See Lessons 4 and 31.

****

Boolean expression

See Proposition.

****

Branch

To branch means to go from one part of a program to another part of the program out of sequence. To do this use the DO command or the TO command.

See DO, TO.

****

50

Command

See Control commands, AID commands

****

Control commands

CTRL stands for the key marked "CTRL." Whenever you see a command with CTRL- and a letter, you are supposed to hold down the CTRL key while you type the letter. ("CTRL" stands for "control.")

CTRL-A. The "repeat" or "again" command causes the retyping of a problem.

CTRL-C. This is the call command. It is used to stop a program that is running. Use CTRL-C to stop either the teaching program or the AID Interpreter. If you have written an AID program that is endlessly looping, type CTRL-C, then type REENTER to start AID again without restarting the program which was looping. See Lessons 1, 2 and 16.

CTRL-G. This is the "go" command. You use this command only in the teaching program to go to the lesson or problem you choose. After you type CTRL-G, the computer asks "WHERE TO?" Then you specify the lesson or problem you want. See Lesson 1.

CTRL-H. This is the "hop" command. It causes the teaching program to skip the problem you are working on and go to the next one. Use this command whenever you want to go on to the next problem without doing or finishing the current one.

CTRL-O. This is the "Oh, shut up" command. It will stop the computer from typing. The computer will then wait for a response from the user.

CTRL-T. This is the "tell" command. If you are using the teaching program and want the answer to a problem, type CTRL-T and the computer will print the answer and then go on to the next problem. See Lesson 1.

CTRL-U. This is the "undo" command. It will cause the computer to erase the line you have just typed.

?. This is the hint command. If you are using the teaching program and want a hint about the problem you are working on, type a question mark, ?. The computer will then give you a hint. See Lesson 1.

****

Conditional definition of functions

A function is said to be defined "conditionally" if the value of the function depends upon some condition such as "...IF X > 0" or "...IF 2 < X AND X < 7." For example, the absolute value function can be defined in this way:

```
    For x > = 0, A(x) = x.
    For x < 0,   A(x) = -x.
```

In AID, this conditional function is defined by the command

```
    LET A(X) = (X > = 0: x;  x < 0: -x)
```

The form of a conditional definition in AID is

```
    (condition: value;  condition: value; ...;  condition: value)
```

Generally, the last condition (and last colon) may be omitted, in which case the last value listed is used for "everything else," i.e., for all cases not covered by one of the preceding conditions. The absolute value function may be written without the last condition:

```
    LET A(X) = (X > = 0: X;   -X)
```

****

Counter

A counter is a variable used for counting. The counter is usually set to some initial value, say 0, and then increased by some amount, say 1, at regular intervals. One common use of a counter is to count the number of times a loop is used. One of the commands inside the loop should change the value of the counter (usually by adding or subtracting a given number). Somewhere inside the loop there is an "exit condition," in which the counter is compared with another number to decide if AID should repeat the loop or if it should exit from the loop and go on to some instruction outside the loop. See Lessons 23, 24, 25, 26 and 36.

****

COS

COS(x) is the cosine function. AID will give the cosine of the number you give. X must be given in radians and the absolute value of X must be less than 100.
Example:

```
    COS(0) = 1
```

****

CTRL

See Control commands

****

Debug

(See also Trace) To debug a program, you must find and correct all the errors in it, whether they are logical errors or simply typing errors. A trace is an effective method for finding precisely where an error is. See Lesson 19.

****

DELETE

Use DELETE to remove a variable, a specific element in an array, or an entire array, along with the values belonging to them from computer storage. You may also DELETE a step, a part, a formula, or a form. One DELETE command may be used to DELETE several items.
Examples:

```
    DELETE Z
    DELETE A(2)
    DELETE FORM 71
    DELETE Y, FORMULA B, PART 7
```

See Lessons 5, 8, 10, and 11. Also see FILE commands.

****

DEMAND

DEMAND X causes the computer to stop and wait for the user to type a value. DEMAND can only be used as an indirect command.

Examples:

```
AID command:                   output:
1.3 DEMAND B                   B =    *
7.12 DEMAND M(2,4)             M(2,4) =     *
4.1 DEMAND P AS "POUNDS"       POUNDS =     *
```

See Lessons 12 and 26.

****

Direct step

An AID command not preceded by a step number is called a "direct step." AID interprets and executes a direct step as soon as you type the RETURN key. You must type a direct step each time you want it executed. DEMAND and TO may not be used as direct steps.

Examples:

```
AID command:        output:
TYPE 2*7            2*7 = 14
SET X = -3          no output (stores -3 in location X)
```

****

DISCARD

See FILE commands. Also see DELETE.

****

DO

The DO command is used to execute an indirect step or part. You may specify how many times the step or part is executed (if you don't specify, it will be executed only once). You may also use a FOR clause and specify a range of values for which the step or part is to be executed.

Examples:

```
DO STEP 10.1.
DO PART 6, 2 TIMES.
DO STEP 8.2 FOR X = 12(2)20.
```

See Lessons 10, 11, 12, and 18. Also see FOR clause.

****

DP

DP(x) is the digit part function. This function uses the scientific notation form of a number and finds the new form of the digit part of the number you specify.

Examples:

```
241.37 in scientific notation is 2.4137*10↑2, so
    DP(241.37) = 2.4137
.24137 in scientific notation is 2.4137*10↑(-1), so
    DP(.24137) = 2.4137
```

The DP function is introduced in Lesson 47.
See Scientific Notation, XP.

****

Erase

To erase a line, hold down the CTRL key while you type the letter U. To erase one character at a time, type the RUBOUT key once for each letter you want erased. See DELETE, DISCARD. See Lesson 1.

****

Errors

In writing AID programs you may make two kinds of errors:

1. Semantic errors. A semantic error is the kind that occurs when you leave out a necessary command or use a valid AID command when you intended to use another. AID will execute the commands just as you wrote them. This means that the only way to detect this kind of error is to see if you are given a wrong answer. A program may keep running indefinitely if an infinite loop is introduced. Type CTRL-C twice to escape, then type "REENTER."
2. Syntax errors. These are the errors that occur when you type something which is meaningless to AID. Because AID does not understand, it will stop and print an error message, then wait for you to do something (such as correcting the mistake and starting again!).

See Lesson 19. Also see Erase.

****

Execute

To execute a program, you make the computer do the commands in the program. This is done by writing the program and then giving AID a command to execute the program (for example, DO PART 5). Indirect steps and parts are stored and you must use a DO command to cause AID to execute them. Direct steps are always executed immediately.

****

Exit condition

An exit condition is a command within a loop which tells AID whether to repeat the loop or to quit looping. One kind of exit condition compares a counter with another number to decide. When the condition of the comparison is not met, AID exits from the loop and goes to the next step. No exit condition is needed if the loop contains a DEMAND command, since you can stop the loop at any time by typing only a carriage return when AID waits for you to give a value.

Examples:

```
    1.4   TO STEP 1.25 IF X > 25.
    9.34  TO STEP 9.1 IF SQRT (X) < 10.
```

See Lessons 23, 24, 25, 26 and 36. See Counter.

****

EXP

EXP(x) is the exponential function, E↑X, where E is Euler's number (2.71828183).

Example:

EXP(3) = 20.0855369

See Lesson 31.

****

Exponent

In an exponential function the exponent tells how many times the base is multiplied by itself. The exponent may be either a number or a variable.

Examples:

3 is the exponent: X↑3
Z is the exponent: 7.43↑Z

The AID function EXP(X) is equivalent to 2.71828183↑X, so X is the exponent. A fractional (or decimal) exponent indicates which root of a number is being calculated. For example, the square root of X may be written either

X↑(1/2)

or

X↑(.5).

If the exponent is negative you first do whatever is indicated by the numerical value of the exponent (find the proper root or multiply the base by itself the correct number of times). Then take the reciprocal of the result.

Examples:

4↑(-3) = 1/4↑3
10↑(-6) = 1/10↑6

If the exponent is 0, the value of the expression is 1, regardless of the value of the base.

Examples:

2↑0 = 1
5.5↑0 = 1
0↑0 = 1

See Lessons 4 and 31. See Base, Exponential Function.

****

Exponential function

An exponential function is a function in which the variable appears as an exponent.

Examples:

F(X) = 2↑X
G(X) = 1.2↑(3*X)
H(X) = X↑X

The AID function EXP(X) is an exponential function which is equivalent to 2.71828183↑X. Also see Base, Exponent.

****

FILE commands

Programs, formulas, forms, etc., may be filed for later use by using the AID file commands. The commands

```
USE FILE 100
FILE PART 3 AS ITEM 5
```

will cause PART 3 to be permanently stored as item 5 on disk file 100. The PART may be fetched from the file at a later date by using the commands

```
USE FILE 100
RECALL ITEM 5
```

Item numbers can be from 1 to 25.

Examples of file commands:

```
USE FILE 100
FILE F AS ITEM 6
FILE FORM 70 AS ITEM 10
FILE PART 2 AS ITEM 12
```

An item is erased from a file by a DISCARD command:

```
DISCARD ITEM 17
```

See Storage. See Lesson 19.

****

FIRST

FIRST is an AID function that finds the first value in an array which satisfies the specified proposition.

Example:

```
FIRST(I = 1(1)30: A(I) > 700)
```

I is the index of the array A so I = 1(1)30 tells which elements of the array are to be considered. A(I) > 700 is the proposition which must be satisfied. The result of the FIRST function will be the index of the first element in the array A which is greater than 700. See Lesson 45.

****

FOR

A FOR clause can be used after a DO command. The FOR clause specifies the values for which the DO command must be executed. There are two ways to specify the values in a FOR clause:

1. The values can simply be listed:

   ```
   DO STEP 1.3 FOR X = 1,2,3,10.
   ```

   Step 1.3 is done one time for each of the four values of x listed.

2. The values may be specified by giving the range:

   ```
   DO STEP 1.3 FOR Y = 3(2)13.
   ```

   Step 1.3 will be done for Y = 3, 5, 7, 9, 11, and 13. 3 is called the initial value, 2 is the step size, and 13 is the final value. (See Range.)

See Lessons 10, 11, and 25.

****

FORM

FORM is the command used to tell AID to type an answer in some form other than the standard form. To specify the form, first type the word "FORM," then give it a number, and follow it with a colon. On the next line type the form you want AID to print your answer in, including any words you want. Where AID is to fill in the number, use back arrows to represent digits. Put the decimal in the appropriate place. Caution: use only one line.

Example:

```
    FORM 73:
    THE ANSWER IS ←←←←.←←
```

Then when you want AID to use your form, use a command like

```
    TYPE X IN FORM 73.
```

See Lesson 22.

****

FP

FP is the fraction part function. AID answers with the fraction part of the number or variable you specify.

Examples:

```
    FP(132.576) = .576
    FP(-8.543) = -.543
```

The FP function is introduced in Lesson 9.

****

Function

See AID functions.

****

Go

See CTRL-G, WHERE TO?

****

Hint

In the teaching program, hints are provided for most problems. To get a hint, type a question mark, ?. There are usually several hints with each problem; the first time you type a question mark you will get the first hint, the second question mark will give you the second hint, etc.

****

IF clause

An IF clause may be added to any AID command so that the command will be executed only if the proposition in the IF clause is satisfied.

Example:

```
    1.1 SET B = 50 IF A > 100.
```

AID will set the value of B equal to 50 only if A is greater than 100. See Lesson 15.

****

Index

An index is a reference number for a list or an array. The index is the number in parentheses. Since all the members of a list or an array have the same letter, each member has its own index to distinguish it from the others.

Example:

```
    L(16) = 10 means the 16th number in the list L is 10.
        L is the label for the list.
        16 is the index of a particular element.
        10 is the value of that element of the list.
```

The plural of "index" is "indices." An index is also called a subscript. See Lesson 32.

****

Indirect step

An indirect step is an AID command preceded by a step number. Indirect steps are stored for later use, rather than executed immediately. When you use a DO command or a TO command, the step will be executed.

Example:

```
    1.3 TYPE 3*2.
```

AID will not print anything until you give an indirect DO or TO command or one of these direct commands:

```
    DO PART 1.
        or
    DO STEP 1.3.
```

Step numbers must be decimal numbers containing both an integer portion and a decimal portion; a step number can contain a maximum of nine significant digits. Some commands may only be used in indirect steps; those commands are DEMAND and TO. See Lesson 10. Also see Part, Step number.

****

Initial value

The term initial value may refer to two different things. It is the first value given to a counter (see Loops, Exit conditions). It also refers to the first value of a range of values in a FOR clause using this form:

```
    initial value (step size) final value
```

In the command

```
    DO PART 3 FOR X = 6(2)20
```

the initial value is 6.
See Range.

****

Input

Input commands assign values to the variables in a program. Most programs must provide for input. The SET and DEMAND commands are used for input. See Lesson 19.

****

INST

See Teaching program.

****

IP

IP(X) is the integer part function. AID will give the integer part of the number or variable you specify.

Examples:

```
IP(.723) = 0
IP(72.8) = 72
IP(-6.9) = -6
```

The IP function is introduced in Lesson 9.

****

L AID

See AID Interpreter.

****

L INST

See Teaching Program.

****

Lesson

To get a specific lesson using the teaching program, you must

First, sign on (see page 3)
Second, start the teaching program (Type "L INST")
Third, specify the lesson (Type "L5" for Lesson 5, "L36" for Lesson 36, etc.)

Also see CTRL-G.

****

LET

LET is used to define functions and propositions.

Examples:

```
LET A(W,L) = W*L        (formula for area of a rectangle)
LET B = X AND Y    (B will be true only if X and Y are both true.)
LET T(A) = SIN(A)/COS(A)     (tangent function)
```

See Lessons 8 and 46.

****

Line number

See Step Number, Indirect Step.

****

List

You may use one letter to represent a list of numbers. Each number in the list must have an index to distinguish it from the other members of the list.

```
Examples:  L(1) = 10    (The first number in list L is 10.)
           L(2) = 6     (The second number in list L is 6.)
           L(3) = 29    (The third number in list L is 29.)
```

See Lessons 32 and 33. Also see Array.

****

LOG

LOG(X) is the natural logarithm function. LOG(X) gives the logarithm to the base E of X. E is Euler's number (2.71828183). X must be greater than 0.
Example:

```
LOG(650) = 6.47697236
```

The LOG function is introduced in Lesson 31.

****

Logical operator

The logical operators in AID are AND and OR. Operations involving AND are done before operations involving OR. See Lesson 44. Also see Propositions.

****

Loop

A loop is a portion of a program that is repeated. The number of times a loop is executed depends on the counter and on the exit condition. Loops are first discussed in Lesson 23.

****

MAX

MAX is the AID function that finds the largest value in a list.
Example:

```
MAX(5, -4, 3, y, x↑2)
```

You may also specify the list as a part of a sequence. You must specify which numbers in the sequence are to be considered and what the formula for the sequence is.
Examples:

```
MAX(I = 1,2,3,4: I*3) is the same as MAX(3, 6, 9, 12)
MAX(J = 10(-2)0: 2↑J) is the same as
    MAX(2↑10, 2↑8, 2↑6, 2↑4, 2↑2, 2↑0)
```

See Lesson 37.

****

MIN

MIN is the AID function that finds the smallest value in a sequence. You must tell AID which numbers in the sequence are to be considered and what the formula for the sequence is. For short sequences you may simply type the list of numbers.
Examples:

```
MIN(i = 1(1)5: i*3)
MIN(j = 3,0,-2: 2↑j)
MIN(4,8,-7,z)
```

See Lesson 37. Also see MAX.

****

Mistakes

See Errors, see Erase.

****

Multiple choice problems

See Answer.

****

NOT

See Propositions.

****

Numbers

Numbers may be expressed in either decimal form (2348.25) or in scientific notation (2.34825*10↑3). Numbers are limited to 9 significant digits. See Lesson 4.

****

Number line

The number line is a line divided into equal parts. One dividing point is labeled 0 and all the dividing points to the right are labeled consecutively 1,2,3,... . All the dividing points to the left of 0 are labeled -1,-2,-3,..., consecutively.

Example:

```
—.—.—.—.—.—.—.—.—.—
 -2 -1  0  1  2  3  4  5  6
```

****

Operational symbols

The AID symbols for arithmetic operation are these:

! !	absolute valuc
↑	exponentiation
*	multiplication
/	division
+	addition
-	subtraction

The order of priority of the operations is this:

! !
↑
* / evaluated from left to right
+ - evaluated from left to right

See Lessons 2, 3, 4 and 29.

****

OR

OR is a logical operator used in propositions. If any element connected by OR is true, then the entire expression is true, otherwise the expression is false.

Examples: assume A = TRUE, B = FALSE, C = FALSE

X = B OR C	X = FALSE
Z = A OR B OR C	Z = TRUE

See Lessons 15 and 44. Also see Propositions.

****

Output

An output command causes AID to print the results of processing. Most programs should provide for output. The only AID output command is TYPE. See Lessons 2 and 19.

****

PART

A PART consists of all the indirect steps with the same value in the integer portion. For example, these steps all belong to PART 2.

```
    2.001 SET X = 1
    2.99  SET X = X + 1
    2.4 TYPE X
```

See Lesson 11.

****

PROD

PROD multiplies all the specified numbers in a sequence together. You must tell AID which members of the sequence are to be used and what the formula for the sequence is. For short sequences you may simply type the list of numbers.

Examples:

```
    PROD(j = 1,2,3,4: j + 3)
       ...this is equivalent to (1+3)*(2+3)*(3+3)*(4+3)
    PROD(i = 5(5)30: j/4)
       ...this is equivalent to (5/4)*(10/4)*(15/4)*(20/4)*(25/4)*(30/4)
    PROD(2,4,Z,.8,-2)
       ...this is equivalent to 2*4*Z*.8*(-2)
```

See Lesson 37. Also see SUM, MAX, MIN.

****

Proposition

A proposition is a mathematical sentence made up of arithmetic or logical statements that use the relational operators (>,=,etc.), NOT, and the logical operators (AND, OR). The value of a proposition is either true or false. The order of execution within a proposition is

1. evaluate expressions
2. relational operations
3. NOT
4. AND
5. OR

Examples: assume X = TRUE, Y = FALSE, Z = TRUE

```
    B = X AND Y               B is FALSE
    A = X AND Y OR Z          A is TRUE
    C = (2 < 3) OR (7 > 10)   C is TRUE
```

Propositions are discussed in Lessons 44-46. See TV.

****

Range

In a number of different AID commands a list of numbers can be specified by defining the range of the numbers in this way:

```
i(s)f
```

where i = the initial value, s = the step size, and f = the final value.

Examples:

```
DO PART 7 FOR X = 15(5)40
```

(The initial value is 15, the step size is 5, and the final value is 40, so the list of numbers is 15, 20, 25, 30, 35, 40.)

```
TYPE MAX(N = 1(7)29: N/3)
```

(The initial value is 1, the step size is 7, and the final value is 29, so the list of values for N is 1, 8, 15, 22, 29.)

A range specification may also be used with MIN, SUM, PROD, and FIRST.

****

RECALL

See FILE Commands.

****

Reciprocal

The reciprocal of a number, say A, is found by dividing 1 by the number A.

Examples:

number	reciprocal
3	1/3
2.5	1/2.5 = .4
.5	1/.5 = 2
1/3	1/(1/3) = 3

****

Recursion

Recursion is a way of defining a function on the integers by (1) specifying the value of the function for the integer 1, and (2) defining the value of the function for integers greater than 1 in terms of the value of the function for smaller integers. For example, the factorial function F(X) may be defined by these two equations:

```
F(1) = 1
```

(this specifies the value of the function for the integer 1.)

```
F(X) = X*F(X-1) for X > 1
```

(this defines the value of the function for X in terms of integers less than X.)

In AID, the above two equations are combined in a single conditional expression, as follows:

```
F(X) = (X=1: 1;  X > 1: X*F(X-1))
```

****

REENTER

To stop a runaway program, type Ctrl-C twice, then type "REENTER." AID does the next step and then stops and tells you where it is so you can decide what to do next. See Ctrl-C.

****

Relational symbols

These are the relational symbols used in AID:

=	equal	>	greater than
#	not equal	<=	less than or equal to
<	less than	>=	greater than or equal to

The relational symbols are discussed in Lesson 15.

****

Repeat

To have a garbled problem retyped, type CTRL-A, for "again."

****

Scientific notation

Scientific notation is used to write very large and very small numbers.

		scientific notation
30000	=	3.0 * 10↑4
4560000	=	4.56 * 10↑6
0.0025	=	2.5 * 10↑(-3)
0.00000071	=	7.1 * 10↑(-7)

See Lesson 4.

****

Semantic errors

See Errors.

****

SET

The SET command assigns values to variables.

Examples:

```
SET X = 5.25
SET Z = A*B   (A and B must already have values.)
```

The SET command is introduced in Lesson 5.

****

SGN

SGN(X) is the sign function. It gives 1 if X is a positive number, 0 if X is 0, and -1 if X is a negative number.

Examples:

```
SGN(25) = 1
SGN(0)  = 0
SGN(-762.4) = -1
```

The SGN function is introduced in Lesson 9.

****

Sign-on

See Page 1 of this manual.

****

Sign-off

To sign off use these commands:

```
CTRL-C   (to stop the program)
K        (to sign off)
```

****

Significant digits

The significant digits of a number are the digits beginning with the first non-zero digit on the left and ending with the last non-zero digit on the right.

Examples:

number	significant digits
0.2030	203
100	1
.00976	976

In AID, numbers are limited to 9 significant digits.

****

SIN

SIN(X) is the sine function. AID finds the sine of X. X must be expressed in radians. The absolute value of X must be less than 100.

Example:

```
SIN(0) = 0
```

The SIN function is introduced in Lesson 30.

****

SQRT

SQRT(X) is the square root function. AID finds the positive square root of X. X cannot be negative.

Examples:

```
SQRT(9) = 3
SQRT(60 + 40) = 10
```

The SQRT function is introduced in Lesson 9.

****

Start

To start using the computer, you must sign on (see Page 1).

To start the AID Interpreter type:

```
L AID
```

To start the teaching program type:

```
L INST
```

See Lessons 1 and 2. Also see AID Interpreter, Teaching Program.

****

65

STEP

Every AID command is called a "step." There are indirect steps, which are saved for later execution, and direct steps, which are executed immediately.
See Lesson 10. See AID Commands, Indirect Steps.

****

Step number

Any AID command may be preceded by a step number to make the command into an indirect step (which is stored, rather than executed immediately). Step numbers must be decimal numbers containing both an integer portion and a decimal portion; a step number may contain a maximum of nine significant digits. For example, the following are all valid step numbers:

1.2
1.3
10.678
10.6781233

See Indirect Step.

****

Stop

To stop a runaway AID program, type CTRL-C twice, then type "REENTER." To stop either the AID Interpreter or the teaching program, type CTRL-C (see Control commands). To stop for the day, you must sign off: Type "K" after you have typed CTRL-C.

****

Storage

Storage locations are in the short-term memory (core) of the computer. AID gives each variable, each member of a list, etc., its own storage location. If you change the value of a variable, AID finds its storage location, takes out the old value and puts in the new value. The SET command is used to store numbers and lists of numbers. The LET command is used to store function definitions and definitions of propositions. Indirect steps (steps with a preceding step number) are automatically stored. Anything in short-term memory may be changed simply by redefining it, or it may be erased by using a DELETE command. For long-term storage, see FILE Commands.

****

Subscript

See Index.

****

SUM

SUM is the AID function that adds the specified members of a sequence. You must tell AID which members of the sequence to consider and what the formula for the sequence is. For short sequences you may simply list the numbers.

Examples:

```
    SUM(j = 1,2,3,4: j*3)
        ...equivalent to (1*3) + (2*3) + (3*3) + (4*3)
    SUM(i = 1(3)25: i↑2)
        ...equivalent to 1↑2 + 4↑2 + 7↑2 + ... + 25↑2
    SUM(10,X,Z,-42.1)
        ...equivalent to 10 + X + Z + (-42.1)
```

See Lesson 37. Also see PROD, MAX, MIN.

****

Syntax errors

See Errors

****

Teaching program

The teaching program is the one that teaches you how to write programs using the AID language. After you are signed on, you may start the teaching program by typing:

```
    L INST
```

For complete instructions, see page 3 of this manual.

****

Tell

See CTRL-T.

****

TO

TO is a branching command used to tell AID to go to a step or part out of sequence. TO must be used indirectly only.

Examples:

```
    2.75 TO STEP 2.3.
    17.4 TO PART 15.
```

TO is introduced in Lesson 16.

****

Trace

A trace is a table used to find errors which are difficult to spot otherwise. To make a trace, list the steps in a program in the order they are done. For each step also list the values of the variables after the step is done. Sometimes output is listed for each step. Traces are discussed in Lesson 17.

****

67

Trigonometric functions

The only trigonometric functions in AID are SIN(X) and COS(X). You must define your own functions if you want to use any other trigonometric functions. For example, the tangent function can be defined by

```
    LET T(X) = SIN(X)/COS(X)
```

See SIN, COS.

****

Truth tables

See Lesson 44.

****

TV

TV(X) is the truth value function, where X is a proposition. If the proposition is true, TV(X) will be 1. If the proposition is false, TV(X) will be 0.

Examples: assume A = -5 < 3 and B = (2 < 0) OR (2 < 1)

```
    TV(A) = 1
    TV(B) = 0
```

The TV function is discussed in Lesson 45.

****

TYPE

The TYPE command causes AID to print out the specified information.

```
      command:          output:
    TYPE 2*3           2*3 = 6
    TYPE ←             (a blank line)
    TYPE "VALUES"      VALUES
    TYPE F             F(X): 3*X↑2
    TYPE X             X = 3.47
    TYPE STEP 17.2     17.2 SET X = 2/Y
```

One TYPE command may be used for several things:

```
    TYPE FORMULA F, SQRT(12),3 + 2.7.
```

See Lesson 2.

****

USE

See FILE Commands.

****

Variable

In AID, variables are used to designate storage locations for numbers, formulas, lists of numbers, arrays, etc. AID variables are the single letters A, B, C, ..., Z.

Examples:

```
    SET A = 2              (A is a number)
    LET F(X) = X↑2 + 3     (F is a formula)
    SET A(2) = 7.05        (A is a list)
    SET B(3,7) = 21.76     (B is an array)
    SET M = A AND B        (M is a proposition)
```

****

WHERE TO?

In the teaching program "WHERE TO?" is typed by the computer to indicate that the user can specify a lesson or problem to do next.

To continue your lessons, type the RETURN key.
To start Lesson 19, type "L19"
To do Lesson 45, Problem 6, type "L45-6"
To get Summary of Lesson 21, type "S21"
To get a Review of Lesson 26, type "R26", etc.

See Lesson 1.

****

XP

XP(X) is the exponent part function. This function takes the number you give and finds the value of the exponent when your number is expressed in scientific notation.

Examples:

```
24137 in scientific notation is 2.4137*10↑4 so
    XP(24137) = 4
.0024137 in scientific notation is 2.4137*10↑(-3) so
    XP(.0024137) = -3
```

See Lesson 47.

****

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
OPERATING NOTE NO. 32

October 12, 1967

AID FOR ON-LINE COMPUTATION

adapted from RAND documentation
by S. Russell and R. Gruen

1. INTRODUCTION

AID[+] is an on-line, time-shared computing service that is designed to appear to each user as a personal "computing aide," interacting with the user and responding to instructions couched in a simple language and transmitted over communication lines from the user's Teletype.

This memorandum describes the programming language for requesting computations of AID. Prior experience with other programming languages (e.g., FORTRAN) is neither necessary nor applicable; indeed, reliance upon such experience may be misleading.

The section below is an overview and should be read carefully. Section 3 is a fairly complete description of the language, designed as a reference. The examples, however, should be studied; they are positive rather than negative, showing what is permitted rather than what is not allowed.

2. OVERVIEW

Users request actions of AID by typing single-line commands called steps. A numerical label prefixed to the step is an implied command to AID to retain the step as part of a stored program. AID files away labelled steps in sequence according to the numeric value of the label or step number. The step number, therefore, determines whether an addition, insertion, or deletion is required.

Steps are organized into parts according to the integer parts of the step numbers. Steps and parts are units that may be introduced, edited, typed out, or filed in long-term storage. In addition, they are natural stored-program units for specifying, in a hierarchial manner, procedures to be carried out by AID.

[+] AID - Algebraic Interpretive Dialog is derived from JOSS, a system developed by The RAND Corporation. JOSS is a trademark and service mark of the RAND Corporation for its computer program and services using that program. We are indebted to The RAND Corporation for the use of the program and its documentation.

Decimal and logical values may be assigned to any of the 26 letters admitted as identifiers. Values may be organized into vectors and arrays by using indexed letters, and the letters themselves may be used to refer to arrays for purposes of deletion, typing, filling in long-term storage, and as actual parameters of *formulas* (see below).

In addition to values, arbitrarily complex expressions for values and letters may be assigned to a letter, which may then be used as an abbreviation for the expression; expressions so assigned are called *formulas*. Formulas involving formal parameters (identified by letters) may also be assigned to a letter. The letter and expressions for actual parameters, in functional notation, may then be used as an abbreviation for the formula with the actual parameters substituted for the formal ones. The letter itself may be used to refer to the formula for purposes of deletion, typing, filing, and as an actual parameter of a formula.

Programs for evaluating the sum, product, largest, and smallest of a set of decimal values--and for evaluating the first in a range of decimal values for which a condition holds--can be expressed succinctly and used as expressions for values:

```
  SUM( I = 1(1)N : A(I) )
 PROD( X, Y, Z/2 )
  MAX( I = 1(1)N : A(I)*B(I) )
  MIN( X, Y/3, Z*2 )
FIRST( I = X(1)Z : P(I) )
```

Either of the two notational styles may be used, except for FIRST which finds the first I for which P(I) is TRUE. Programs for determining the *conjunction* or *disjunction* of a set of logical values can also be expressed in either style, and used as expressions for logical values.

Short programs for choosing expressions differentially on the basis of a set of conditions can also be expressed succinctly and used as expressions. The notation chosen abbreviates phrases such as:

if x = y use x + y, if x > y use x, otherwise use y

by (X = Y: X + Y ; X > Y: X ; Y)

Such *iterative functions* and *conditional expressions*, together with formulas, lead to powerful, direct expressions for complex procedures, particularly recursive ones.

AID represents decimal numbers in scientific notation: nine digits of significance and a base-ten scale factor in the range -99 through +99. Addition, subtraction, multiplication, division, and square root are carried out to give true results rounded to nine significant digits; zeroes are substituted on underflow while overflow yields an error message. In other elementary functions, care is taken to provide reasonable significance and continuity of approximation, to factor out error conditions, and to hit certain "magic" values on the nose.

The six numerical relations together with AND, OR, NOT, and a set of elementary logical functions may be used to express logical values and conditions (which may be attached to any step).

A single, general rule governs the formation and use of expressions for values: with the exception of step labels, which must be decimal numerals, wherever a decimal (logical) numeral is allowed in a command, an arbitrarily complex expression for a decimal (logical) value may be used.

AID types answers one-per-line, identifying answers by the expression used in the step calling for the output; in the event of conditional expressions, AID uses only the chosen sub-expression for identification. Decimal points and equal signs are lined up, and fixed-point notation is used whenever possible. For more formal output, the user can create full-line FORMS to specify literal information and blank fields to be filled in with answers. A string of up arrows with an optional decimal point is used for fixed-point fields; a string of periods specifies a tabular form of a scientific notation (floating point).

Users can request AID to file, in long-term storage, identifiable units and collections of units--steps, parts, forms, formulas, and values. Users may then request AID to recall such filed items, discard them from the files, or type out a list of items in a file.

Users start AID off on the task of carrying out a stored program by directing AID to DO a step or part--iteratively (for a range-of-values) or a specified number of times, if desired. AID cancels all outstanding tasks before beginning a direct (i.e., initiated from the console) task, begins the interpretation of a part at the first step of the part, and then interprets each step in sequence. Each subsequent indirect (i.e., initiated by a step of a stored program) DO causes AID to retain the status of the current task, pause to carry out the new task, and then return to continue the suspended one. If the user wishes AID to behave in the same manner for a directly initiated task, the DO command must be enclosed in parentheses.

AID modifies this general behavior whenever it encounters: a) an error; b) a branching command; c) a stopping command; d) a command for terminating a task or a portion of a task; e) an interrupt-signal from the user. The deep and involved hierarchy of tasks and formulas that can occur (recursion is allowed) demands that AID's status be perfectly clear each time control is transferred to the user, for any reason. In addition to error messages, interrupt messages, and stopping messages, AID transmits status messages on completion of parenthetical tasks to distinguish this state from the state of having finished a direct, nonparenthetical task. The user is able to proceed in every situation, in the event of errors, he can take corrective action, and then direct AID to continue with a GO command.

3. DESCRIPTION

EDITING INPUT LINES

AID indicates that it is ready to receive input by typing out an asterisk (*). Characters may be deleted sequentially backward by striking the RUBOUT key. Typing asterisk (*) at the beginning or end of an input line cancels the line.

RULES OF FORM

One command per line, one line per command.

Commands begin with a verb and end with a period.

Words, variables, and numerals may neither abut each other nor contain embedded spaces; spaces may not appear between an identifier (of an array, a formula or a function) and its associated grouped argument(s); otherwise, spaces may be used freely.

Asterisk typed by AID	Step number	Verb	Arguments	Modifiers
→	*1.23	TYPE	X, Y, Z+3	IN FORM 3 IF X+Y > 1Ø.
	*1.4	DO	PART 6	FOR X = 1(1Ø)1Ø1, 1ØØØ.

DIRECT COMMAND: Step number omitted; command is executed immediately.

STORED COMMAND: Step number present; command is stored in order of step number.

STEP: A stored command; step number is limited to 9-digit numbers $\geq$ 1.

PART: A group of steps whose step numbers have the same integral part.

FORM: A pictorial specification of literal information and fields to be filled with values, for formal output. Fields are denoted by strings of left arrows (with optional point) or strings of dots (for a tabular form of scientific representation).

```
*FORM 7:
*I = ←←.←←← AMPS.          V = ......... VOLTS
```

NUMBERS: Range: $\pm 10^{-99}$ to $9.99999999 \cdot 10^{99}$
Precision: 9 significant digits

SYMBOLS: Single letter identifiers. May identify decimal values, logical values (true, false), formulas, and arrays of values.

ARRAYS: Up to 10 indices having integer values in the range [-250,250].

DECIMAL OPERATIONS:

+ - * / ** ↑

Single asterisk for multiplication, double asterisk or up arrow (↑) for exponentiation.

RELATIONS: < > <= >= = #

Extended relations (e.g., $a < b \leq c$) permitted. Number sign for not equal.

LOGICAL OPERATIONS:

AND OR NOT

GROUPERS: ( ) [ ] (used interchangeably in pairs)

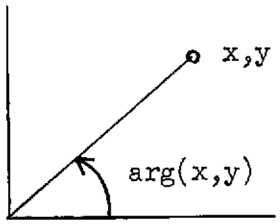
3 + 1/2 + 1/4.5	=	$3 + (1/2 + (1/4 \cdot 5))$
-2↑3*4-5	=	$(-(2^3) \cdot 4)-5$
2**3**4	=	$(2^3)^4$
A OR B AND NOT C OR D	=	a or (b and not c) or d

BASIC FUNCTIONS

SQRT(X)	square root, $x \geq 0$
SIN(X)	$\lvert x \text{ in radians} \rvert < 100$
COS(X)	
LOG(X)	natural log, $x > 0$
EXP(X)	e^x
ARG(X,Y)	angle of point x,y in radians, arg(0,0)=0.

NUMBER DISSECTION FUNCTION

SGN(X)	-0,0,+1 for $x < 0, x = 0, x > 0$		
IP(X)	integer part	ip(3.2)	= 3
FP(X)	fraction part	fp(3.2)	= .2
DP(X)	digit part	dp(100.2)	= 1.002
XP(X)	exponent part	xp(100.2)	= 2
!x!	absolute value for decimal values $\lvert\text{true}\rvert = 1$, $\lvert\text{false}\rvert = 0$		



SPECIAL FUNCTIONS

SUM[I=A(B)C:F(I)]	SUM(X,Y,Z+10)
PROD[I=A(B)C:F(I)]	PROD(A+B,C+D,E+F)
MIN[I=A(B)C:F(I)]	MIN(A,B,C,D)
MAX[I=A(B)C:F(I)]	MAX(B,1,X+Y)
FIRST[I=A(B)C:P(I)]	gives first I for which P(I) is true
TV(P) = 0,1	IF P = FALSE, TRUE
= FALSE, TRUE	IF P = 0, P ≠ 0
CONJ[I=A(B)C:P(I)]	CONJ(X=1, Y > 3,P)
DISJ[I=A(B)C:P(I)]	DISJ(A=B=C,A > Y ≥ 10)

CONDITIONAL EXPRESSIONS

$(P_1:E_1: P_2:E_2: E_3)$

where: P_i are expressions for logical values,

means: If P_1 is true use E_1, if P_2 is true use E_2, otherwise use E_3.

```
*SET X = ( Ø < Y < = 5: Ø ; Y < 1Ø: 2*2 ; 5 ).
*LET P(X) = [ X = Ø: 1 ; PROD( 1 = 1(1)X : 1 ) ].
```

AID VERBS

SET Assigns value. SET and final period may be omitted on direct commands.

```
*SET X = 3.
*SET A(5,X) = Y+3*X-X*2.
```

LET Defines a formula of 9 or fewer parameters.

```
*LET F(X,Y) = X**2+1Ø*X-6*Y.
*LET H = (B-A)/2.
*LET D(F,X) = [ F(X+D)-F(X) ]/D.
*LET Q(R) = [ R=Ø: 1 ; FP(R)=Ø: R*Q(R-1) ]
```

DELETE Erases values, parts, steps, forms, formulas.

```
*DELETE A, PART 3, ALL FORMS.
*DELETE ALL VALUES, ALL FORMULAS.
```

TYPE Types quoted text or values, blank lines (←), parts, forms, etc.

```
*TYPE "THE QUICK BROWN FOX."
*TYPE X+3, D( SIN,Ø ), ←, ALL STEPS.
```

DEMAND Requests an input value from user. Executing:

```
1.4 DEMAND A( 3,I+1Ø ).
```

with I = 59 causes AID to respond with:

```
A(3,69) =*
```

The desired value for A(3,69) may then be typed, followed by a carriage return.

DO Executes or "does" part or step. FOR clause gives range of values. Returns to user if direct, to next step if indirect.

```
*DO PART 6 FOR X = .1, 3(2)1Ø, 1ØØ*A+2*B.
```

TO Sends AID to indicated part or step.

```
*1.3 TO STEP 3.5.
```

STOP Interrupts program. Console control returns to user.

GO Restarts program after interrupt, error message, or STOP command.

DONE Signals completion of DO for current FOR value.

QUIT Signals completion of DO for all FOR values.

CANCEL Signals completion of all DO's.

(DO) Executes part or step without disturbing interrupted calculation.

```
*(DO PART 3.)
```

(CANCEL) Signals completion of last (DO).

LINE Types a blank line.

FORM After form number, colon, and carriage-return pauses for user to enter format for output. Fields are strings of left arrows or dots.

```
*FORM 3:
*X = ←←←←.←← Y = ←←←←.← Z = ..........
```

USE User file in dictionary.

```
*USE FILE 1Ø5 (DTA7).
```

FILE Stores an ITEM in the files.

```
*FILE PART 3, A, Z, AS ITEM 7 (CODE).
```

RECALL Retrieves an ITEM from files.

```
*RECALL ITEM 7 (CODE).
```

DISCARD Erases a filed ITEM.

```
*DISCARD ITEM 3 (FOO).
```

AID MODIFIERS

IF Precedes a logical expression conditioning any command.

```
*TYPE X IF Ø < = X < 5.
*SET Y = 3 IF X < = 1Ø AND X*Y#1Ø.
```

FOR Used on DO only. PART or STEP is executed repeatedly for specified set of values.

```
*DO PART 3 FOR X = 1(1)1Ø(1Ø)1ØØ, 1ØØØ.
*DO STEP 1.2 FOR X = .Ø1, .Ø3, .1(A)B.
```

TIMES Used on DO only. Causes repeated execution of PART or STEP.

```
*DO PART 4, 43 TIMES.
*DO STEP 7.3, N+1 TIMES.
```

IN FORM — Modifies TYPE only. Causes values to be typed in fields of specified FORM.

```
*TYPE X, Y, Z*2 IN FORM 3.
```

AID NOUNS

TIME — Gives 24-hour time.

```
*TYPE TIME.
```

SPARSE — Declares undefined array elements to have zero values; they require no storage.

```
*LET A BE SPARSE.
```

$ — The current line number. Maximum is 54.

EXAMPLE OF A COMPLETE AID TYPEOUT

```
*TYPE ALL.

1.1 LINE.
1.13 TYPE FORM 2.
1.15 DO PART 2 FOR B = .1(.1)4.

2.Ø5 SET A = - B.
2.1 LINE IF FP($/5) = 1/5.
2.6 TYPE B, EXP(B), LOG(EXP(B)), C*I(F) IN FORM 1.

FORM 1:
←←←.←← ....... ←←.←←←← ←.←←←←

FORM 2:
  X    EXP(X)     LOC     PROB

        I(F): H/2*SUM(I=1(1)3Ø:SUM[J=1(1)2:F(Y(I,J))])
        F(X): EXP(-X**2/2)
           H: (B-A)/3Ø
      Y(I,J): A+H/2**[T(J)+2**I-1]

           C =       .398942281

        T(1) =       .57735Ø268
        T(2) =      -.57735Ø268

*
```

INSTRUCT

Coders' Manual
(excerpts)

by

Jamesine E. Friend

K. Summary of Op Codes

Note: If an op code has more than 1 argument, separate the arguments by commas.

Op Code	No. of Arguments	Kind of Argument	Comments
LESSON	2	1. Strand identifier (1 to 6 letters) 2. Lesson number	Pseudo op code. Marks beginning of a lesson.
EOL	none		Pseudo op code. Marks end of lesson.
PROB	1	text string	Displays problem number and problem text. Pauses for student response.
QUES	1	text string	Displays problem text. Pauses for student response.
SPROB	1	text string	Displays problem text. Pauses for student response.
TELL	1	text string	Displays text of correct answer, when requested by student. Branch to next problem. Default routine causes branch to pause student response.
HINT	1	text string	Displays text for hint when requested by student. Pause for student response.
EXACT	1	text string	Analyzes student response for exact match. Sets SCORE.
MC	1	text string containing list of letters	Analyzes response to multiple-choice problems. Sets SCORE to 1 if completely correct, -1 if completely wrong, -2 if partially wrong, -3 if partially correct. Checks form of response.

80

EQ	1	text string containing: 1. number 2. optional number, giving tolerance	Analyzes response for equality with coded number, within tolerance specified by second number. Sets SCORE. Checks form of response.
KW	1	text string	Analyzes response for existence of coded text string. Sets SCORE.
NO	0		Analyzes response for "no" or "n". Sets SCORE. Checks form of response.
YES	0		Similar to NO.
TRUE	0		Checks for "true" or "t". Sets SCORE. Checks form of response.
FALSE	0		Similar to TRUE.
LIST	*undefined*		
SET	*undefined*		
NOTEXACT . . . NOTKW		Similar to op codes described above, with negation of SCORE.	
CA	1	optional text string	Executes only if SCORE > 0. Displays message. Branch to next problem.
C1	1	optional text string	Executes only if SCORE = 1. As for CA.
C2	1	optional text string	Executes only if SCORE = 2. As for CA.
C3	1	optional text string	Executes only if SCORE = 3. As for CA.
WA	1	optional text string	Executes only if SCORE < 0. Branch to pause for student response.

W1	1	optional text string	Executes only if SCORE = -1. As for WA.
W2	1	optional text string	Executes only if SCORE = -2. As for WA.
W3	1	optional text string	Executes only if SCORE = -3. As for WA.
BRCA	4	1. strand identifier. 2. lesson number 3. problem number 4. optional text string	Executes only if SCORE > 0. Displays message. Branch to specified problem.
BRWA	4	1. strand identifier 2. lesson number 3. problem number 4. optional text string	Executes only if SCORE < 0. Displays message. Branch to specified problem.
WS	1	optional text string	Executes only if SCORE < 0. Displays message. Branch to next problem.

L. *BNF Definition of Coding Language*

```
<strand> ::= <lesson> EOL<CR><strand>|<empty>

<lesson> ::= <lesson><prob>|<lesson identifier><cr>

<prob> ::=<PROB command><non-PROB commands>|
          <SPROB command><non-PROB commands>|
          <QUES command><non-PROB commands>

<non-PROB commands> ::= <HINT series><non-PROB commands>|
                        <TELL command><non-PROB commands>|
                        <analysis command><non-PROB commands>|
                        <action command><non-PROB commands>|
                        <empty>

<HINT series> ::= <HINT command><HINT series>|<empty>

<analysis command> ::= <EXACT command>|
                       <MC command>|
                       <EQ command>|
                       <KW command>|
                          <NO command>|
                          <YES command>|
                          <TRUE command>|
                          <FALSE command>|
                          <NOTEXACT command>|
                                 .
                                 .
                                 .
                          <NOTKW command>

<action command> ::= <CA command>|
                     <C1 command>|
                     <C2 command>|
                     <C3 command>|
                     <WA command>|
                     <W1 command>|
                     <W2 command>|
                     <W3 command>|
                     <BRCA command>|
                     <BRWA command>|
                     <WS command>
```

Problem Statement Commands:

```
<PROB command> ::= PROB <space><text string><CR>

<SPROB command> ::= SPROB <space><text string><CR>

<QUES command> ::= QUES <space><text string><CR>

<HINT command> ::= HINT <space><text string><CR>

<TELL command> ::= TELL <space><text string><CR>
```

Analysis Commands:

```
<EXACT command> ::= EXACT <space><text string><CR>

<MC command> ::= MC <space><left superquote><letter list>
                    <right superquote><CR>

<letter list> ::= <letter><comma><letter list>|
                  <letter><space><letter list>|
                  <letter>

<EQ command> ::= EQ <space><left superquote><decimal number>
                  <right superquote><CR>|
                 EQ <space><left superquote><decimal number>
                  <decimal number><right superquote><CR>
```

```
<KW command> ::= KW <space><text string><CR>

<NO command> ::= NO <CR>

<YES command> ::= YES <CR>

<TRUE command> ::= TRUE <CR>

<FALSE command> ::= FALSE <CR>

<NOTEXACT command>
      .            Similar to EXACT...EQ commands
      .
      .
<NOTEQ command>
```

Action Commands:

```
<CA command> ::= CA <space><text string><CR>|CA <CR>

<C1 command> ::= C1 <space><text string><CR>|C1 <CR>

<C2 command> ::= C2 <space><text string><CR>|C2 <CR>

<C3 command> ::= C3 <space><text string><CR>|C3 <CR>

<WA command> ::= WA <space><text string><CR>|WA <CR>

<W1 command> ::= W1 <space><text string><CR>|W1 <CR>

<W2 command> ::= W2 <space><text string><CR>|W2 <CR>

<W3 command> ::= W3 <space><text string><CR>|W3 <CR>

<BRCA command> ::= BRCA <space><strand identifier>,
                 <lesson number>,<problem number><CR>|
                 <BRCA command>,<text string><CR>

<BRWA command> ::= BRWA <space><strand identifier>,
                   <lesson number>,<problem number><CR>|
                   <BRWA command>,<text string><CR>

<WS command> ::= WS <space><text string><CR>|WS<CR>

<strand identifier> ::= *1 to 6 letters*

<lesson number> ::= *natural number 1 to 999*

<problem number> ::= *natural number 1 to 128*
```

Miscellaneous and "Primitives":

```
<text string> ::= <left superquote><character string>
                  <right superquote>

<character string> ::= <character><character string>|<empty>

<letter> ::= *a - z, upper or lower case*

<decimal number> ::= *any number in decimal form with not more than
                     9 significant digits; includes integers*

<left superquote> ::= *Philco: less-than-or-equal sign
                      *Teletype: Ctrl-Shift-L

<right superquote> ::= *Philco: greater-than-or-equal sign
                       *Teletype: Ctrl-Shift-M
```

SAMPLE CODED PROBLEM
(taken from Lesson 4)

```
PROB
"AID WILL DO EXPONENTIATION BEFORE IT DOES MULTIPLICATION, DIVISION,
ADDITION OR SUBTRACTION.
WHAT WOULD AID ANSWER?
   TYPE 5 * 2↑3"

TELL
"5 * 2↑3 = 5 * 8 = 40"

HINT
"AID WOULD EVALUATE 2↑3 FIRST."

HINT
"DO 2↑3 FIRST, THEN MULTIPLY BY 5."

NOTEQ "1000"

WA
"WRONG.  AID WOULD EVALUATE 2↑3 FIRST.  TRY AGAIN."

EQ "40"

WS
"WRONG.  5 * 2↑3 = 5 * 8 = 40"

BRCA L,4,6



SPROB
"LET'S GO THROUGH THIS PROBLEM STEP-BY-STEP.
WHICH EXPRESSION IS EVALUATED FIRST IN THIS COMMAND?
   TYPE 32/4↑2

A.  4*2
B.  32/4
C.  4↑2
N.  NONE"

TELL
"C (EXPONENTIATION IS DONE BEFORE DIVISION.)"

HINT
"EXPONENTIATION IS DONE FIRST."

EXACT
"4↑2"
```

```
CA

MC "C"

CA

WA


SPROB
"...AND WHAT IS THE VALUE OF 4↑2?"

TELL
"4↑2 = 4*4 = 16"

HINT
"4↑2 = 4 * 4 = ???"

EQ "16"

CA

WA


SPROB
"SO THE VALUE OF 32/4↑2 IS THE SAME AS THE VALUE OF 32/???"

TELL
"16 (4↑2 = 16)"

HINT
"WHAT ANSWER DID YOU GET FOR 4↑2?"

HINT
"32 DIVIDED BY 4↑2 IS THE SAME AS 32 DIVIDED BY WHAT NUMBER?

EQ "16"

CA

WA
```

```
SPROB
"THEN WHAT WOULD AID ANSWER TO THIS COMMAND?
   TYPE 32/4↑2."

TELL
"32/4↑2 = 32/16 = 2"

HINT
"WHAT IS THE VALUE OF 32/4↑2"

HINT
"AID WILL DO EXPONENTIATION BEFORE DIVISION."

EQ "2"

CA

WA



SPROB
"WHAT WOULD AID ANSWER?
   TYPE 10↑3 * 2"

TELL
"10↑3 * 2 = 1000 * 2 = 2000"

HINT
"AID WOULD DO EXPONENTIATION BEFORE MULTIPLICATION."

HINT
"Hint: 10↑3 = 10 * 10 * 10."

NOTEQ "10000"

WA
"WRONG.  AID WOULD DO EXPONENTIATION BEFORE MULTIPLICATION."

EQ "2000"

WS
"WRONG.  10↑3 * 2 = 1000 * 2 = 2000"
BRCA L,4,6
```

```
SPROB
"THERE IS AN EASY WAY TO DO PROBLEMS THAT HAVE EXPONENTIATION AND
ALSO SOME OTHER OPERATION: IMAGINE THAT THERE ARE PARENTHESES AROUND
THE TERM WITH THE EXPONENTIATION.
FOR EXAMPLE,
TO DO 3↑4 + 2, DO (3↑4) + 2.
TO DO 625/5↑2, DO 625/(5↑2).
TO DO 4↑2 * 2↑4, DO (4↑2) * (2↑4).
WHAT IS THE VALUE OF 5↑2/2?

TELL
"5↑2/2 = (5↑2)/2 = 25/↓ = 12.5"

HINT
"REWRITE THE EXPRESSION WITH PARENTHESES.  THEN TRY TO DO IT."

HINT
"5↑2/2 = (5↑2)/2 = ???"

EQ "12.5"

CA

WA



SPROB
"WHAT WOULD AID ANSWER?
   TYPE 10↑3/10↑2"

TELL
"10↑3/10↑2 = (10↑3)/(10↑2) = 1000/100 = 10"

HINT
"REWRITE THE EXPRESSION WITH PARENTHESES (USE TWO PAIRS).
THEN FIND THE VALUE."

HINT
"10↑3/10↑2 = (10↑3)/(10↑2) = ???"

EQ "10"

CA

WA
```

```
SPROB
"WHAT WOULD AID ANSWER?
   TYPE 10↑3 0 10↑2"

TELL
"10↑3 - 10↑2 = (10↑3) - (10↑2) = 1000 - 100 = 900"

HINT
"REWRITE THE EXPRESSION WITH PARENTHESES BEFORE YOU DO IT."

HINT
"10↑3 - 10↑2 = (10↑3) - (10↑2) = ???"

EQ "900"

CA

WA
```

N74-26706

STUDENT PERFORMANCE IN COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING

by

J. E. Friend, J. D. Fletcher, and R. C. Atkinson

May 10, 1972

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

STUDENT PERFORMANCE IN COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING

J. E. Friend, J. D. Fletcher, and R. C. Atkinson

Stanford University

Stanford, California 94305

1. Introduction

In 1967 the Institute for Mathematical Studies in the Social Sciences received a three-year grant* from the National Aeronautics and Space Administration to do exploratory research in the optimization of instruction and to develop a practical course of study using computer-assisted instruction (CAI). The course that was developed was a one-quarter college course in computer science (Friend & Atkinson, 1971). Developmental testing of the course was accomplished using NASA personnel and Stanford students. The course was not the only product of this grant; an entire computer-assisted instructional system (Friend, 1971) was developed, with this course as the first application.

The following year, the National Science Foundation provided funds** for a second, and very similar, application of the instructional system developed under the NASA funding. The second course was an introduction to computer programming for culturally-deprived high school students. This course taught the programming language, BASIC, whereas the first course taught AID. Both AID and BASIC are higher-order algebraic languages analogous to ALGOL and FORTRAN. The main difference between the

* NASA Grant NGR-05-020-244
**NSF Grant No. GJ-443X

two courses is that the AID course was written for college students with a good background in algebra, and the BASIC course was written for high school students with low reading ability and little or no background in algebra. The BASIC course was adequately tested with a group of over 100 students in an inner-city high school in San Francisco.

In 1970, the Office of Naval Research awarded a research contract to the Institute to continue its program of basic reseach in instructional strategies using the already developed instructional system and courses as research tools. The main subject of study was to be the AID course, but data collected in connection with the use of the BASIC course was also to be used if appropriate.

This report is a preliminary discussion of the research conducted under the ONR contract and is concerned only with the AID course.

2. The Instructional System

The course "Computer-assisted Instruction in Programming: AID" is an introductory course in computer science for community or junior college students with some background in high-school algebra. The course is completely self-contained and requires no supervision from a qualified instructor of programming. A brief student manual is supplied to supplement the instruction given by computer.

The computer used is a Digital Equipment Corporation PDP-10 located at Stanford University and owned and operated by the Institute for Mathematical Studies in the Social Sciences. Connected to this computer by telephone lines are "Model-33" teletypewriters located in the schools and used for communicating with students. Instructions are printed on the teletypewriter terminal, and the student responds by typing his replies on the same terminal. Teletypewriter operation is simple and can be learned from short instructions printed in the student manual.

Once the student has the teletypewriter in operation, all further instruction is given by computer under the control of a program known as INST. This program, which is the major component of the INSTRUCT system, interprets coded lessons providing individualized, tutorial instruction to the student. This instructional system and the method of programming lessons for it are described adequately by Friend (1971), and a detailed description will not be repeated here.

The AID course uses most of the features of the INSTRUCT system. The course contains 50 lessons organized into seven "lesson blocks." Each lesson block contains five tutorial lessons, followed by a self-test and a general review. The 50th lesson is a concluding lesson independent of the lesson blocks. The structure of the main strand is shown in Figure 1. The lessons vary in length from 10 to 60 exercises depending upon the content. Lessons of average length require about one hour to complete. Lesson length is completely under student control, and a student may take a few exercises or several lessons in one sitting.

One of the primary teaching strategies used in the course is the provision for student control of the sequence of instruction. Students may skip from any exercise in the course to any other exercise at any time, retracing their steps if they wish, or skipping lessons entirely. This strategy is intended to encourage the student to take responsibility for learning the concepts, not simply for progressing through a given set of exercises. Most college students are capable, and desirous, of assuming this responsibility, and the provision for student control of

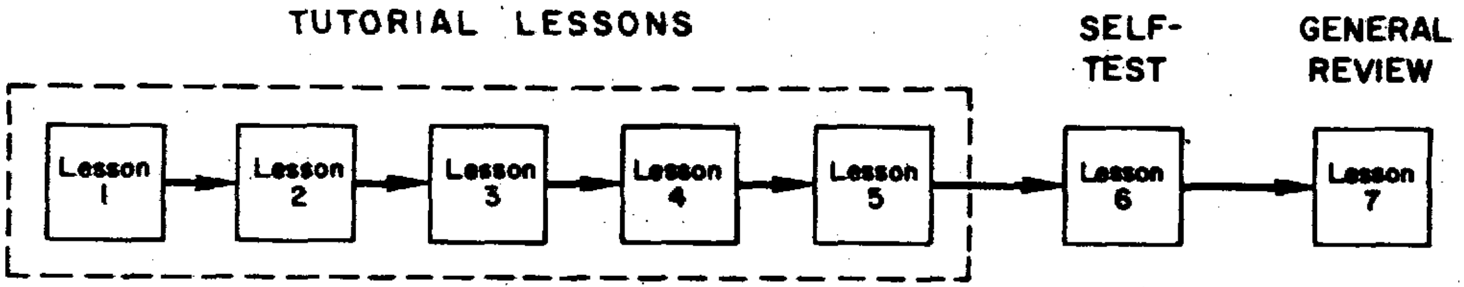


Figure 1. Structure of main lesson strand.

instruction is assumed to provide motivation. Whether or not all students are motivated by this treatment and whether or not it is an effective strategy in terms of the amount of learning taking place remains to be tested.

Because of this allowance for student control, the 50 lessons may be taken in any sequence. If the student does not exercise his prerogative for choosing the sequence, the lessons are automatically sequenced for him; and it is assumed that most students will, in fact, do the lessons in the order indicated.

Besides the main strand of lessons, the course also contains review lessons, one for each of the tutorial lessons in the seven lesson blocks. These review lessons are also tutorial and cover the same concepts as do the lessons they are associated with. However, they present each concept from a slightly different viewpoint providing additional practice in the skills to be learned. In general, each lesson covers five or six related concepts. In review lessons, the student may review whichever concepts he wishes, in any order he chooses. In fact, he must choose the order; there is no automatic sequencing provided by the program. At the end of each tutorial lesson, the student is asked if he wants to review any of the ideas covered in the lesson he has just completed. The student need not wait for these reminders, of course, since he can call for any review, or any exercise in any review, whenever he wishes.

Also associated with each tutorial lesson is a summary of the lesson, and the student is reminded at the end of each lesson that summaries are available. Each summary is printed in an 8-1/2" × 11" form that can be torn off and saved by the student as a permanent record.

In addition to the main strand of lessons, the reviews, and the summaries, there is a strand of "extra-credit" problems containing more difficult programming problems to be solved by the more capable students. It is recommended to instructors that the solutions of these problems be submitted for extra-credit if the course is graded. Not every lesson has associated extra-credit problems because they are not always appropriate to the subject matter. If there are such problems, the student is asked if he wants to try them at the end of the lesson.

The relationship and sequence of lessons, summaries, reviews and extra-credit problems are illustrated in Figure 2.

As described above, the main strand of lessons is divided into seven-lesson blocks, five tutorial lessons followed by a self-test and a general review. Lessons numbered 6, 13, 20, 27, 34, 41, and 48 are self-tests. Lessons numbered 7, 14, 21, 28, 35, 42, and 49 are general reviews. The self-tests are optional and students are told at the beginning of each test that the test is for their information only and may be skipped. The tests cover the concepts taught in the preceding five lessons and provide a good indication of weaknesses and areas requiring review. The general review which follows the self-test is also optional and is recommended for students who do poorly in the self-test. The general review is programmed to call the reviews for individual lessons as subroutines. For example, the student may review parts of Lessons 1, 3, and 4 and skip the review of Lessons 2 and 5. If he wishes to review Lesson 2, he will take the same review he would have taken if he had chosen to review the lesson immediately after taking it, and he can decide which concepts from that lesson to review and in which order.

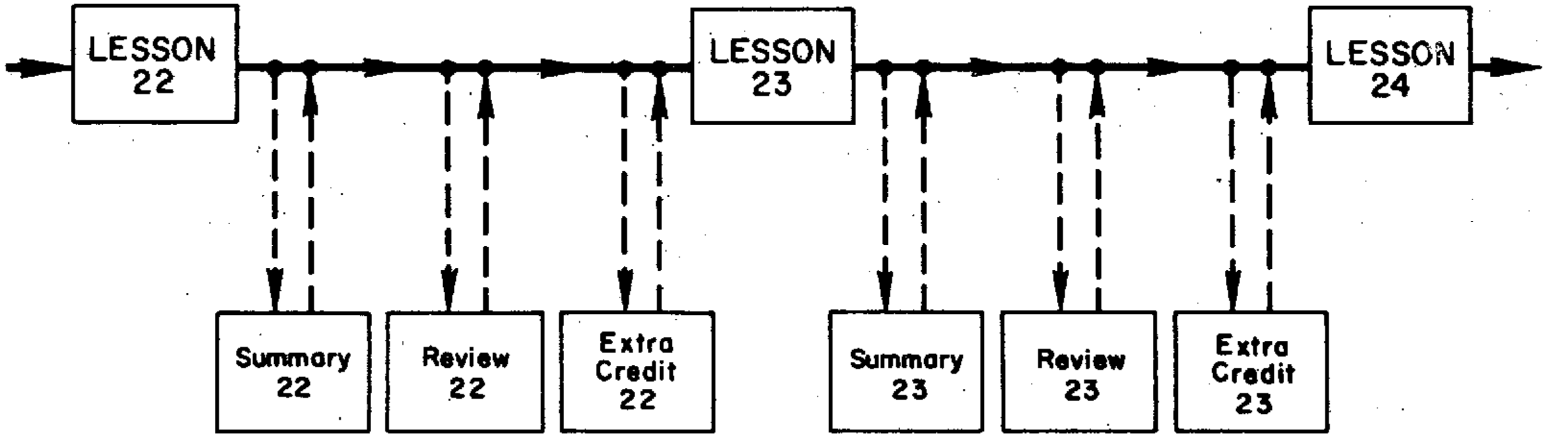


Figure 2. Relationship and sequence of tutorial lessons, summaries, reviews and extra-credit problems.

This review system is a gross method for providing individualized remediation. A more sensitive means of individualizing remediation is used within the lessons themselves where remedial sequences of exercises are given immediately to students who demonstrate an inadequate understanding of the material being taught. The remedial sequences that are imbedded within the lessons are not optional and are automatically provided for students whose responses indicate a lack of understanding. Because of this automatic remediation, different students may receive different numbers of exercises in a given lesson.

A student who makes an incorrect response to an exercise may not need an entire sequence of remedial exercises. He may profit from a single specific corrective message, pointing out the error and allowing him another try at the same problem. This kind of specific correction is used for most exercises in the course. Messages are provided, not for all possible incorrect responses, but for those incorrect responses judged to be most likely to occur.

In line with the provision for student control of the sequence of instruction, there is also provision for student control of the amount of instruction. This is done by providing additional instruction for almost every exercise in the form of "hints" and sample correct answers. After the problem statement is typed, the student is free to request a hint by typing a question mark. For many problems, a sequence of three or four hints is provided. The student may request one hint, make an attempt to answer the exercise, then ask for a second hint, and so on. Also, the student may ask for the correct answer at any time, either before or after he makes a try at the problem. He does this by typing two keys simultaneously, the CTRL (control) key and the letter T. The answer will be printed, and he will proceed to the next exercise.

In the tutorial lessons, there is no limit on the number of attempts that a student may make for an exercise. As soon as he makes a correct response he will be given the next exercise in sequence, but if he fails to respond correctly, he is given another chance. If he cannot do the problem at all, he can judge for himself whether to continue trying or whether to go on with the lesson; whenever he decides to proceed without giving a correct response he can do so by typing CTRL-T, thereby getting the correct answer and the next exercise.

The CTRL key is also used in several other student controls. CTRL-G is used when the student wants to jump to a different exercise; after he types CTRL-G, the computer will ask "Where to?" and he can type the lesson and problem number that he desires. CTRL-U is used to erase entire lines, and CTRL-Z is used to stop the instructional program and end the session.

3. The AID Interpreter

The AID language was chosen as the subject of the course, not only because it is a useful algebraic programming language, but also because it is easily learned by a beginner. AID was developed, under the name of JOSS (Mark & Amerding, 1967; Shaw, 1965), by the Rand Corporation for use by scientists and engineers at Rand who needed an easily learned programming language that was capable of performing complex algebraic tasks. JOSS was later implemented for several other computers under a variety of names.

One of the advantages of AID as a beginner's language is that it is an interpreted, rather than compiled, language, which will act immediately on direct commands given by the user. Because of this feature the syntax and use of many AID commands can be taught to the student before he is taught about stored programs. In the AID course, the use of AID is introduced in Lesson 2 and the concept of stored programs is not introduced until Lesson 10.

A second advantage of AID, as compared to, say, FORTRAN or LISP, is that the syntax is a subset of English; commands are easily read as English imperative sentences. As an example, here is a simple AID program:

```
1.1  SET X = 1.3074.
1.2  SET Y = X/37.5.
1.3  TYPE Y IF Y < .029.
1.4  TYPE Y - .01 IF Y > .029.
1.5  TYPE "EUREKA" IF Y = .029.
```

As with most programming languages, AID is easier to read than to write, and considerable practice is needed by most students before they can produce a complete simple program like the above without error. However, they can begin to produce simple direct commands like

```
SET Y = X/37.5
```

or

```
TYPE Y - .01
```

on their first day.

Though simple to learn, AID is nevertheless a powerful tool for algebraic tasks. Definitions of conditional functions and recursive functions are relatively simple, and there are a variety of useful standard functions, such as the basic trigonometric, exponential, and logarithmic functions. Lists and matrices can be easily defined, although AID does not provide the standard matrix manipulation functions found in some other programming languages.

Basic programming features, common to all programming languages, are available in AID. Variables and functions can be labeled, as can stored commands. There are branching commands, subroutine calls, and conditional clauses, and there are input and output commands for both disk and teletype.

AID commands and programs are interpreted by a program called, aptly, the AID interpreter. The interpreter used by the students of this course is a program written by Digital Equipment Corporation, the manufacturer of the Institute's PDP-10 computer.

The use of the AID interpreter is taught in the course, and the students are expected to use it frequently to solve problems given them in the lessons. Thus, the students taking this course will use two programs: INST, the instructional program which talks about the AID languages, and AID, the interpreter which uses the AID language. In a previous version of the course, the two programs were completely independent and the students were required to learn to start and stop both programs so that they could switch back and forth to do the programming problems. This method was feasible but awkward and time consuming. For the present version of the course, both programs were modified to permit easy access from the instructional program to the AID interpreter. Students can call the AID interpreter at any time simply by typing the word "AID," and can return to the instructional program by typing "INST." This interface provides an additional advantage for research in that it supplies an easy way to cross-reference data collected by the two programs. It is now possible to pass information, invisibly to the student, from one program to the other; every time the student calls the AID interpreter, the student's identity and current position in the course are passed to the AID interpreter so that data collected by AID can be keyed to data collected by INST..

4. Data

Both the instructional program and the AID interpreter contain data-collection subroutines that enable them to store information about student responses as the student is working. These data are stored temporarily on the disk and later transferred to permanent tape storage.

The instructional program records the following information with each student response.;

1. Student number. (Each student is assigned a unique number when he first enrolls for any computer-assisted instruction offered by the Institute.)
2. Date.
3. Time of day.
4. Lesson identifier.
5. Problem number.
6. Subproblem number.
7. Trial number.
8. Student response. (This is an exact character-by-character record of the response made by the student, excluding erasures made by the student.)

9. Analysis of correctness. (This is a record of how the instructional program scored the student's response to this exercise.)

10. Lesson score. (This is a cumulative score of the student's first responses to exercises within this lesson.)

11. Number of hints. (This is a record of the number of hints requested by the student before he made this reponse.)

12. Answer provided by program? (This switch records whether or not the student response was a request for the correct answer.)

It is estimated that about 3,000 such blocks of individual response data are collected for each student taking the course.

The AID interpreter also collects data but the form is simpler since it contains no routines to analyze the student input. The AID interpreter collects the following information:

1. Student number.

2. Date.

3. Lesson identifier. (This is information sent to the AID interpreter by the instructional program.)

4. Problem number. (This also is sent by the instructional program.)

5. Subproblem number. (Again, this is sent by the instructional program.)

6. Student input. (This is an exact character-by-character duplicate of everything typed by the student, excluding erasures.)

It is estimated that about 300 such blocks of data are collected for each student taking the course.

A variety of students have been enrolled in the AID course since the data collection routines were added to the programs. One of the largest groups is comprised of students in the "High Potential" program at the University of California at Los Angeles. These students are entering freshmen who do not meet the usual entrance requirements for UCLA but who, for one reason or another, are suspected to have a higher potential than revealed by their high school records or by entrance examinations. These students may be described as "culturally deprived," and are products of inner-city schools where average achievement is quite low.

A second large group of students are from DeAnza College in Cupertino, California. DeAnza is a community college located a short distance from Stanford. The DeAnza students who have enrolled in the course were all unprovisionally admitted to DeAnza and most of them have a better high school background than the UCLA students.

A number of NASA personnel, from the Ames Research Center at Moffett Field, California, and from the Manned Spacecraft Center in Houston, Texas, have also enrolled for the course.

The Institute is providing CAI for hearing impaired students in several schools, and a half-dozen of these handicapped youngsters have also enrolled for the course during the last year.

5. The Curriculum: Lessons 1 to 21

The content of the course "Computer-assisted Instruction in Programming: AID" has been described elsewhere (Friend & Atkinson, 1971), so a complete description of the entire course will not be repeated here. The research reported in this paper concerns only the main strand Lessons 1 to 21, and these lessons are described below in considerable detail.

Lessons 1 to 21 contain three "blocks" of lessons; each of the blocks contains five tutorial lessons, one self-test and one general review:

Lessons 1 to 5 - Tutorial

Lesson 6 - Self-test of Lessons 1 to 5

Lesson 7 - General Review of Lessons 1 to 5

Lessons 8 to 12 - Tutorial

Lesson 13 - Self-test of Lessons 8 to 12

Lesson 14 - General Review of Lessons 8 to 12

Lessons 15 to 19 - Tutorial

Lesson 20 - Self-test of Lessons 15 to 19

Lesson 21 - General Review of Lessons 15 to 19

These lessons constitute a brief introduction to programming, covering such concepts as stored programs, use of variables, fundamentals of input and output, the syntax of algebraic expressions and Boolean statements, definitions of functions, conditional clauses and branching, core and disk storage, use of subroutines, and some debugging techniques. Some of these concepts (input and output, core and disk storage, subroutines) are discussed very briefly, while others (syntax of algebraic expression, syntax and meaning of Boolean statements) are covered more extensively. One major programming essential that is not introduced in the first 21 lessons is the loop, which is introduced in Lesson 23. Lists, arrays, trigonometric and exponential functions, recursive functions and the truth function are also introduced in later lessons. A brief outline of Lessons 1 to 21 is given in Table 1.

The 15 tutorial lessons in the first 21 lessons vary in length, depending upon the concepts covered by the lesson. Lesson 12, for example, covers two very simple commands and contains only 14 exercises, whereas Lesson 15 introduces Boolean statements and conditional clauses and contains 62 exercises. Also, the kinds of exercises in the lessons vary

Table 1

Brief Outline: Lessons 1 to 21

Lesson 1	Using the Instructional Program
Lesson 2	Using AID for Arithmetic
Lesson 3	Order of Arithmetic Operations
Lesson 4	Exponents and Scientific Notation
Lesson 5	The SET and DELETE Commands
Lesson 6	Test of Lessons 1 to 5
Lesson 7	General Review of Lessons 1 to 5
Lesson 8	The LET Command
Lesson 9	Some Standard AID Functions
Lesson 10	Indirect Steps, the DO Command, the FOR Clause
Lesson 11	Parts
Lesson 12	The DEMAND Command
Lesson 13	Test of Lessons 8 to 12
Lesson 14	General Review of Lessons 8 to 12
Lesson 15	Relations and the Use of the IF Clause
Lesson 16	The TO Command
Lesson 17	Debugging Techniques
Lesson 18	The Indirect Use of the DO Command
Lesson 19	Debugging, Permanent Storage
Lesson 20	Test of Lessons 15 to 19
Lesson 21	General Review of Lessons 15 to 19

with the subject matter. Lesson 15 has 24 true-false exercises, reflecting the content of the lesson (Boolean statements). Lesson 8, in contrast, has no true-false exercises, instead it has 32 exercises that require the student to predict the result of using given AID commands. Table 2 shows the number and type of exercises in Lessons 1 to 21, excluding the general review Lessons 7, 14 and 21.

The exercises are categorized into the 13 different problem types listed in Table 2. The first four types are multiple-choice exercises. In the AID course, multiple-choice exercises may have more than one correct choice, and the student response is not correct unless all correct choices are listed. The multiple-choice exercises in each lesson are all classified according to the number of correct choices except for the few that include the choice,

N. NONE OF THE ABOVE,

and are classified separately. Of the 675 exercises in the first 21 lessons there are 80 multiple-choice exercises, 56 of which have a single correct choice.

There are a number of exercises that appear to be constructed-response exercises in that the student is not presented with a list of possible answers from which to choose. However, closer inspection reveals that there are actually a limited number of choices of a form clearly implied in the statement of the problem. The following exercises, for example, imply only two choices:

Lesson 19, Exercise 8.

> SUPPOSE AID FOUND A SYNTAX ERROR IN STEP 17.2. DO YOU
> HAVE TO DELETE STEP 17.2 BEFORE YOU RETYPE IT?

Table 2

Number of Exercises, by Type, in Lessons 1 to 21.
(Excluding General Reviews: Lessons 7, 14, 21)

Lesson Number	1	2	3	4	5	6	8	9	10	11	12	13	15	16	17	18	19	20	Total
Multiple-Choice 1 Correct Choice	3	4	7	6	5	2	1	1	2			2	5	5	1	6	1	5	56
Multiple-Choice 2 Correct Choices		4	1		1	2			1	1		1						1	12
Multiple-Choice 3 or More Correct Choices	2	2	2			1			1										8
Multiple-Choice Correct Choice: NONE		1	1	1		1													4
Total Multiple-Choice Exercises	5	11	11	7	6	6	1	1	4	1	0	3	5	5	1	6	1	6	80
Yes-No Exercises (except opinion questions)	2													1			7	1	11
True-False						9							24					1	34
Other Implied-Choice		1		1									2				1		5
Total Implied Choice	2	1	0	1	0	9	0	0	0	0	0	0	26	1	0	0	8	2	50
Predicted AID Response		4	19	25	5	12	32	17	1			13	6					1	135
Constructed AID Command		2	7	1	9	5	8		2	6	1	8		1			7	6	63
Reported Result of AID Use		3	3	3	13	2	4	6	4	4	3	1	1						47
Other Constructed-Response Exercises	3		1	15	17	4	8		13	5	2	3	10	10	18	22	4	10	145
Total Constructed Response	3	9	30	44	44	23	52	23	20	15	6	25	17	11	18	22	11	17	390
"Use AID"		3	3	3	9	2	4	3	5	6	5	2	6	2		1	4		58
Opinion (or preference)	5	4	5	6	4	3	5	4	4	8	3	3	8	8	4	6	8	2	90
Unclassified Exercises	3														4				7
Total	18	28	49	61	63	43	62	31	33	30	14	33	62	27	27	35	32	27	675

Lesson 6, Exercise 8.

```
ANSWER TRUE OR FALSE:
CTRL-Z WILL STOP THE AID INTERPRETER.
```

Lesson 4, Exercise 27.

```
IF YOU USED THIS COMMAND
      TYPE 1/100
WOULD AID GIVE THE ANSWER IN DECIMAL FORM OR IN
SCIENTIFIC NOTATION?
```

Exercises of this type are labeled "implied choice" exercises and are classified as yes-no exercises, true-false exercises, or "other implied-choice" exercises. In the first 21 lessons there are 50 such exercises.

Three-hundred ninety-two of the exercises in the Table 2 lessons fall into the third major group comprising true constructed-response exercises. This group is subdivided into four classes: predicted AID responses, constructed AID commands, reported result of AID use, and "other constructed-response" exercises. The predicted AID responses contain questions like "What would ADD answer if you gave this command ....?" One-hundred and five of the constructed-response exercises are of this type.

A smaller class of constructed response exercises includes the 63 exercises that require the student to construct a complete AID command. This class does not include all, or even most, of the AID commands that the student must construct. It includes exercises that pass the constructed AID commands to the instructional program; excluded are exercises that require direct communication with the AID interpreter. The instructional program is more capable than the AID interpreter of analyzing constructed AID commands in detail and of giving meaningful messages to students who make errors when they first learn a new command. The usual sequence of instruction in introducing a new AID command is as follows:

> First, an example of the command is shown, and its use is explained. In this step, students are usually required to determine which of several forms of the command are syntactically correct.
>
> Second, the student is shown several examples and asked to determine what AID would respond if such a command were given to the AID interpreter.
>
> Third, the student is asked to construct commands that would result in a specified action.
>
> Fourth, the student is asked to start the AID interpreter and give commands of the new kind directly to AID.
>
> Fifth, after using the AID interpreter as directed, the student is requested to report on the results given him by AID.

Exercises in this fifth step are classified in Table 2 as "reported result of AID use." In these exercises, the instructional program can infer the kind of errors a student is making and give him remedial instruction if needed. The device of asking students to switch to the interpreter for computation and to switch back to the teaching program to report results is one of the weakest features of the course, and it is used only from necessity. Clearly, an efficient means of interfacing the two programs is needed to pass information invisibly, and provide meaningful instruction for the students who need it. Only 47 of the 390 constructed-response exercises are of the type needed in the fifth step, because an overall teaching strategy used in the course is to encourage students to assume responsibility for ferreting out their own errors and for determining if they have a correct program.

The fourth class of constructed-response exercises, called "other constructed-response exercises," contains 145 exercises of considerable variety. This class will be further subdivided in a later analysis.

The miscellaneous exercises contain a class designated in Table 2 as "use AID." These exercises require the student to use the AID interpreter in solving a problem. The problems range in complexity from copying commands in order to observe their consequences to solving complex problems by writing and debugging complete programs. Many of these exercises contain three or four problems, and the student is asked to solve all of the stated problems before he switches back to the instructional program and reports the results.

Also in the miscellaneous group are those exercises that elicit an opinion or preference. These exercises either ask the student to express a preference for the sequence of instruction (Do you want a summary of this lesson?) or to give a self-evaluation of his competence (Do you remember how to start the AID interpreter?). The exercises are fairly evenly distributed over lessons and there are 90 of them in the first 21 lessons.

In the following paragraphs, each of the first 21 lessons is described and characteristic exercises from the lesson are given.

Lesson 1: Using the Instructional Program

Lesson 1 is a set of 18 short exercises explaining how to use the instructional program. The mechanics of typing and erasing responses are explained, and instructions are given for starting and stopping the program. The student is taught how to use optional control keys to get additional instruction (HINT and TELL commands) or to alter the sequence of exercises (the GO command).

The style of instruction in Lesson 1, as in succeeding lessons, is informal, and it does not always give explicit directions, requiring the student to attend the instructions carefully and learn by induction from the examples given.

Two of the exercises in Lesson 1 are given here.

Lesson 1, Exercise 3:

> IF MULTIPLE CHOICE PROBLEMS HAVE MORE THAN ONE CORRECT ANSWER, YOU CAN LIST THE CORRECT CHOICES IN ANY ORDER. SUPPOSE B, C AND D ARE THE CORRECT CHOICES FOR A PROBLEM. WHICH OF THESE WOULD BE CORRECT WAYS TO ANSWER?
>
> A. D, B, C, A
> B. B, D, C
> C. B, C, D
> D. D, B, C

Lesson 1, Exercise 14:

> FROM LESSON 1, YOU SHOULD HAVE LEARNED HOW TO SIGN ON AND OFF, HOW TO START AND STOP THE TEACHING PROGRAM, HOW TO GET A HINT, AND HOW TO USE CTRL-G. DO YOU WANT TO REVIEW ANY OF THESE TOPICS?

Exercise 14 illustrates an instructional strategy that is used in many lessons. At the end of a lesson, its content is briefly summarized, and the student is given the option of reviewing topics he is unsure of. If the student responds affirmatively, he is given the review associated with that lesson. In this way, the student is made responsible for the material covered in each lesson and is forced to judge if he is competent to proceed with the course or if he needs additional instruction and practice.

In Lesson 1, five of the 18 exercises are multiple choice, similar in form to Exercise 3 shown above. This proportion of multiple-choice exercises is fairly typical of the course. The multiple-choice format is chosen over a constructed response format depending on the objective of the exercise. For example, if the purpose is to teach students to discriminate between syntactically correct and incorrect AID commands, a multiple-choice exercise is used; if the purpose is to teach the construction of an AID command, a constructed response will be requested.

Of the 13 exercises remaining in Lesson 1, seven could appropriately be labeled "implied multiple choice" since the statement of the exercise clearly implies only a small number of possible responses. In order to keep the terminology straight, however, exercises that are not in the conventional multiple-choice format, will be referred to as "implied choice" exercises, and the term, "multiple choice," will be reserved for exercises that list and label a set of choices and require that the student respond by typing the label or labels.

One of the most frequently used implied-choice exercises is the yes-no exercise. Many of these are designed to elicit opinion rather than information and have no "correct" answer. In Lesson 1, five of the seven implied-choice exercises are of this type and in Table 2 are classified as "opinion exercises" rather than "implied-choice exercises." This proportion is greater than that of later lessons, but still, indicates the style of the lessons.

Lesson 2: Using AID for Arithmetic

The 28 exercises in Lesson 2 teach the student how to start and stop the AID interpreter and how to use it for simple arithmetic by employing the "TYPE" command. The AID symbols for the four simple arithmetic operations (+, -, *, and /) are taught, and the use of parentheses in arithmetic expressions is introduced. By the end of the lesson, the student should be able to start the AID interpreter and give simple, direct commands such as:

```
TYPE  5/25
TYPE  3.25 + 17.4 - 3.12
TYPE  15 * 17 + 25 * 19
```

One of the most persistent errors made by students learning any algebraic programming language is the omission of the multiplication operator in algebraic expressions. The source of this difficulty is the convention of using juxtaposition to indicate multiplication. For example, we ordinarily write

$$a(b+c)$$

without an explicit multiplication operator, but AID, like other algebraic programming languages, demands that the multiplication be indicated explicitly. AID has an asterisk as the symbol for multiplication, as in

```
A * (B+C) .
```

In Lesson 2, there are a number of exercises aimed specifically at overcoming this error.

The following are exercises from Lesson 2.

Lesson 2, Exercise 15:

```
WHAT WOULD AID ANSWER TO THIS COMMAND?
TYPE 72/12
```

Lesson 2, Exercise 19:

USE AID TO DO THESE PROBLEMS:

1. FINT THE AREA OF A RECTANGLE WITH WIDTH 1.72375 AND LENGTH 12.001325.
2. SUPPOSE A SQUARE OF WIDTH .637825 IS CUT FROM THE ABOVE RECTANGLE. FIND THE AREA OF THE SQUARE.
3. FIND THE AREA OF THE REMAINING PART OF THE RECTANGLE.

Of the 28 exercises in Lesson 2, 11 are multiple choice, five are implied-multiple choice, and 12 are true constructed response exercises. The constructed response exercises vary in difficulty, from the simple problem given in Exercise 15, above, to the problem given in Exercise 19.

Lesson 3: Order of Arithmetic Operations

The arithmetic used in Lesson 2 was relatively simple, but in Lesson 3 the complexity increases with the addition of the concept of hierarchy of operations and the use of parentheses. Because each AID command must be typed entirely on a single line, horizontal division bars for grouping cannot be used. Thus, an expression like

$$\frac{xy}{z+3}$$

using more than one line of type, must be translated into the AID expression

$$X * Y / (Z+3)$$

with parentheses to show grouping. The AID expression is more difficult to construct, since it requires a conscious decision about the desired order of evaluation.

Lesson 3 teaches the student how to force an order of evaluation by using parentheses. To do this, the student must recognize the difference between expressions like (16 - 4) - 3 and 16 - (4 - 3). After a few exercises on parentheses, rules are given for the hierarchy of the four basic arithmetic operators (+, -, *, and /), and a number of exercises are given in which the task is to determine the order of evaluation if no parentheses are used.

The following are two examples from Lesson 3.

Lesson 3, Exercise 5:

```
WHAT WILL AID ANSWER TO THIS COMMAND?
     TYPE 1/(100/10)
```

Lesson 3, Exercise 10:

```
LOOK AT THESE THREE COMMANDS.  AID WILL GIVE THE SAME
ANSWER TO TWO OF THEM.  WHICH TWO?

  A. TYPE  3 + (2*4)
  B. TYPE  (3+2) * 4
  C. TYPE  3 + 2 * 4
```

Since Lesson 3 is primarily a review of algebraic notions that may be better understood by some students than by others, it provides more opportunity for individualized branching. A good student can complete Lesson 3 in 27 exercises, but a poor student will be given additional practice and may do up to 49 exercises.

The exercises in Lesson 3 are of medium difficulty, and most can be done quickly. Eleven of the exercises are multiple choice, and an additional five are implied-choice exercises. Three of the exercises, including one with three parts, require the student to use the AID interpreter. Nineteen of the exercises are similar to Exercise 5 above.

Lesson 4: Exponents and Scientific Notation

Lesson 4 is longer than average and extends the work on arithmetic expressions to include exponentiation. First, the concept of exponentiation is reviewed, with the introduction of the AID symbol (↑). The rules for the hierarchy of operations are extended to include exponentiation, and the AID form of scientific notation is introduced. Negative exponents, fractional exponents, and zero as an exponent are also covered. Lesson 4, like Lesson 3, is largely review of algebraic principles and may have been forgotten. The exercises also provide practice in reading and constructing expressions in the form required by the AID interpreter.

Some examples from Lesson 4 follow.

Lesson 4, Exercise 6:

```
WHAT IS THE VALUE OF 5↑2/2?
```

Lesson 4, Exercise 12:

```
USE AID TO EVALUATE EACH OF THE FOLLOWING.

1.  4 SQUARED TIMES 3.1416
2.  THE SUM OF 4 CUBED AND 6.
3.  THE SUM OF THE SQUARES OF 1, 2, 3, 4, 5, 6, 7 AND 8
```

Lesson 4 contains 61 exercises, of which 48 require constructed responses. Most of the constructed responses require arithmetic calculations to answer questions like those illustrated in Exercise 6 above. Three exercises require the student to use the AID interpreter, and he is encouraged to use AID throughout the lesson.

Lesson 5: The SET and DELETE Commands

After the sizeable dose of arithmetic given in Lessons 3 and 4, Lesson 5 provides relief by returning to the mainstream of instruction with the introduction of two new AID commands: the SET command and the DELETE command. SET is used in AID to assign values to real variables; DELETE is used to delete a previous assignment or definition. In AID, variables are single letters, and the SET and DELETE commands are easily learned by most students. A number of word problems are given to illustrate the use of the new commands. Lesson 5 also introduces the multiple-argument form of the TYPE command in which several TYPE commands can be combined into one by separating the arguments with commas (TYPE X,Y,X+Y).

The following exercises are from Lesson 5.

Lesson 5, Exercise 3:

> WHAT WILL AID ANSWER AFTER THESE COMMANDS?

```
    SET B = 1·5
    TYPE 3*B
```

Lesson 5, Exercise 31:

> TO FIND THE NEW AMOUNT IN A SAVINGS ACCOUNT, CALCULATE THE INTEREST AND ADD IT TO THE LAST BALANCE. START AID AND CALCULATE THE INTEREST AND THE NEW BALANCE AFTER ONE YEAR FOR AN ACCOUNT WITH AN INTEREST RATE OF 4.5 PERCENT PER YEAR AND A PREVIOUS BALANCE OF $3274.86. (ASK FOR A HINT IF YOU NEED ONE.)
>
> WHAT IS THE INTEREST ON THE ABOVE ACCOUNT TO THE NEAREST PENNY?
>
> WHAT IS THE NEW BALANCE IN THE ACCOUNT?

Lesson 5, with its 63 exercises, is fairly long and has nine exercises that ask the student to use AID to solve problems. The most difficult of these is shown in Exercise 31 above. The trend toward a higher proportion of constructed responses continues here, with Lesson 5 having only six multiple-choice problems. There are four implied-choice exercises, but all of these are requests for opinions from the student.

Lesson 6: Test of Lessons 1 to 5

Lesson 5 concludes the first five-lesson tutorial block and is followed by a self-test in Lesson 6 and a general review in Lesson 7. Both Lessons 6 and 7 are optional. Lesson 6 contains 40 test questions and problems covering the material in Lessons 1 to 5. Like other self-tests, Lesson 6 supplies no hints, and students are allowed only one try on each exercise. However, the student may request the correct answers at any time by typing CTRL-T. Whenever a student misses an exercise, he is given a review reference and advised to review that topic before proceeding with the course.

The exercises in Lesson 6 are classified according to which lesson they are testing:

Lesson Number	Exercises in Lesson 6 Testing Given Lesson
1	2, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15, 16
2	8, 11, 17, 18, 19, 20, 38-1, 39-1
3	21, 22, 23
4	24, 25, 26, 27
5	28, 29, 30, 31, 32, 33, 34, 35, 36, 37

Exercises 1, 1-1 and 40 are "opinion" exercises and are not listed. Exercises 38 and 39 are "use AID" exercises and are not listed. Of the 38 exercises in the list, 23 are constructed-response exercises.

Lesson 7: General Review of Lessons 1 to 5

The student is allowed to skip Lesson 7, but he is advised to take it if he missed more than five problems in the Lesson 6 self-test. Lesson 7 uses a more complex branching scheme to allow students to review only selected portions of the preceding lessons.

The following is an example from Lesson 7.

Lesson 7, Exercise 5:

```
LESSON 4 WAS ABOUT EXPONENTS AND SCIENTIFIC NOTATION.
FRACTIONAL EXPONENTS AND NEGATIVE EXPONENTS WERE
DISCUSSED, AND ALSO THE USE OF 0 AND 1 AS EXPONENTS.
THE ORDER OF ARITHMETIC OPERATIONS + - * / and % WAS
COVERED.

DO YOU WANT TO REVIEW ANY OF THESE THINGS?
```

If a student answers "yes," he is sent to the review lesson for Lesson 4, where he may review any of the lesson topics in any order he wants.

This first general review also reminds students that they can control the sequence of instruction by using the CTRL-G key and that the student manual provides an outline of the course.

Lesson 8: The LET Command

Lesson 8 is the beginning of a new lesson block, and introduces the powerful LET command, used in AID to define functions. The difference between LET and SET commands is discussed, and a variety of algebra problems is given. A major emphasis in Lesson 8 is on substituting arithmetic expressions for variables in algebraic expressions. Lesson 8 is the first lesson that has optional "extra-credit" problems.

The following are two exercises from Lesson 8.

Lesson 8, Exercise 4:

```
WHAT WILL AID ANSWER?

     LET P(M) = M↑2
     TYPE P(6/2)
```

Lesson 8, Exercise 28:

```
USE AID TO DO THIS PROBLEM. DEFINE A FUNCTION TO CONVERT
DEGREES FAHRENHEIT TO DEGREES CENTIGRADE. THEN CONVERT
THESE TEMPERATURES TO CENTIGRADE:

          0, 10, 32, 72, 212
```

Of the 62 exercises in Lesson 8, over half are similar to Exercise 4 above. There are four multi-part problems that require the use of the AID interpreter. There is only one multiple-choice exercise in Lesson 8, and none of the exercises are more difficult than Exercise 28 above.

Lesson 9: Some Standard AID Functions

In Lesson 9, the student is introduced to four functions already defined in AID. These are:

```
SQRT(X) - the square root function,
  IP(X) - the "integer part" function,
  FP(X) - the "fraction part" function,
 SGN(X) - the sign function.
```

These functions, together with those defined by the student, are used in several problems requiring the use of the AID interpreter.

The following are two exercises from Lesson 9.

Lesson 9, Exercise 2:

```
WHAT WILL AID ANSWER?

TYPE 3 * SQRT(100)
```

Lesson 9, Exercise 14:

```
YOU CAN USE THE AID FUNCTION FP(X) TO FIND OUT IF ONE
NUMBER CAN BE DIVIDED BY ANOTHER WITHOUT A REMAINDER....
IS 2976 EVENLY DIVISIBLE BY 3?
```

Lesson 9 has 31 exercises of which 17 are similar to Exercise 2 above.

Lesson 10: Indirect Steps, the DO Command, the FOR Clause

In Lesson 10, the concept of a stored program is introduced. Up to this point, students have been using AID as a desk calculator, doing all exercises with direct commands, i.e., commands that are executed immediately. In this lesson, the students are taught that TYPE commands can be stored for later execution by prefacing the command with a step number, as in the following examples:

```
2.1  TYPE F(16)
4.7  TYPE X↑2,X↑3 .
```

They are also taught how to execute these stored commands by using a DO command. Two variants of the FOR clause are used to modify DO commands. In the first variant, values for the iteration variable are given by a simple listing:

```
DO STEP 17.3 FOR Y = 1,2,7,14.3.
```

In the second variant, values for the iteration variable are given in a range specification that specifies an initial value for the variable, a step size, and a final value:

```
DO STEP 5 FOR X = 4(2)9.
```

This command specifies that X will assume the value 3, then be incremented by 2 after each iteration of step 5 until X > 9. This is equivalent to the FORTRAN form,

```
      DO 5  X = 4,9,2
    5 < statement 5 >,
```

and the ALGOL form

```
FOR X ← 4 STEP 2 UNTIL 9 DO
< statement 5 >.
```

The following are some examples from Lesson 10.

Lesson 10, Exercise 12:

> USING AID, WRITE AN INDIRECT STEP THAT WILL CONVERT MILES PER HOUR TO FEET PER SECOND. THEN CONVERT ALL OF THESE TO FEET PER SECOND:
>
> 10 MILES PER HOUR
> 100 MILES PER HOUR
> 65 MILES PER HOUR
> 1023 MILES PER HOUR

Lesson 10, Exercise 17:

> WHAT VALUES OF A WILL BE USED IF THIS COMMAND IS GIVEN?
>
> DO STEP 73.7 FOR A = 5(10)35
>
> A. 5, 10, 15, 20, 25, 30, 35
> B. 10, 15, 20, 25, 30, 35
> C. 5, 15, 25, 35

Of the 33 exercises in Lesson 10, four are multiple choice and another four are implied choice. Most of the constructed-response exercises are quite simple, and Exercise 12 above is the only one that requires any problem-solving skills.

Lesson 11: Parts

Lesson 11 explains how indirect (stored) steps are grouped into "parts," Steps 12.1, 12.7, and 12.8, for example, are grouped as

"Part 12," and can be executed by a single command:

```
DO PART 12.
```

The sequence of execution depends only upon the numerical order of the step numbers, and not upon the sequence in which they were written. Thus, steps 3.75, 3.2 and 3.8 will be executed in the order: 3.2, 3.75, 3.8. This concept is clear to most students. The only difficulty is caused by step numbers with trailing zeros; some students fail to order correctly a sequence like 3.5, 3.8, 3.10 (the correct order is 3.10, 3.5, 3.8).

Here are two examples from Lesson 11.

Lesson 11, Exercise 5:

```
YOU CAN TYPE THE STEPS IN ANY ORDER, BUT AID WILL ALWAYS
DO THEM IN NUMERICAL ORDER.  WHICH STEP WILL BE DONE FIRST?

    17.4  TYPE X↑Y
    17.5  SET N=5
    17.2  SET X=10
    17.3  SET Y=2
```

Lesson 11, Exercise 11:

```
A PART (SET OF INDIRECT STEPS) IS ALSO CALLED A PROGRAM.
USE AID TO WRITE A PROGRAM THAT WILL LIST THE RADIUS,
DIAMETER, CIRCUMFERENCE, AND AREA OF A CIRCLE OF RADIUS
R.  THEN USE THE PROGRAM FOR R = 10, 20, 30, 40 AND 50.
```

Lesson 11 has 30 exercises, with an unusually high proportion of "opinion" questions (8 out of 30). Only one of the 30 exercises is multiple choice, and six of the 30 require the student to use the AID interpreter, a slightly higher proportion than was found in earlier lessons.

Lesson 12: The DEMAND Command

In Lesson 12 the DEMAND command is introduced. The DEMAND command is used in AID programs for teletype input. The DEMAND command, unlike previously introduced commands, can be used only indirectly, as a stored command. This lesson also introduces a variant of the DO command:

```
DO PART 7, 5 TIMES
```

The following are exercises from Lesson 12.

Lesson 12, Exercise 4:

> START AID AND WRITE A PROGRAM THAT WILL ASK YOU FOR 3 NUMBERS, A, B, AND C, AND THEN GIVE YOU THE AVERAGE OF THE 3 NUMBERS. AFTER YOU HAVE TESTED YOUR PROGRAM, USE IT TO FIND THE AVERAGE OF
>
> A = 179.053
> B = 23.7
> C = 271.0015

Lesson 12, Exercise 5:

> WHAT COMMAND WOULD YOU USE IF YOU WANTED PART 2 DONE 7 TIMES?

Lesson 12 is very short, containing only 14 exercises. None is multiple choice, and five require the student to use the AID interpreter.

Lesson 13: Test of Lessons 8 to 12

Lessons 8 to 12 constitute the second five-lesson tutorial block of instruction and are followed by an optional self-test (Lesson 13) and a review (Lesson 14). Lesson 13 is structured like other self-tests; the student is given only one try for each exercise, and there are no hints provided. A student who cannot do an exercise can request the correct answer by typing CTRL-T.

The following classifies the exercises in Lesson 13 according to which lesson is being tested.

Lesson Number	Exercises in Lesson 13 Testing Given Lesson
8	1-1, 2, 3, 4, 5, 6, 7, 8
9	9, 10, 11, 12, 13, 14, 16
10	17, 18, 19, 20, 21, 22, 23, 24
11	25, 26, 27
12	28, 29-1

Five of the 33 exercises in Lesson 13 are not included in the list. Exercises 1, 15-1 and 30 are "opinion" questions, and Exercises 15 and 29 are "use AID" exercises. Of the 28 exercises in the list, 25 are constructed-response exercises.

Lesson 14: General Review of Lessons 8 to 12

Lesson 14, the general review of Lessons 8 to 12, is optional but is recommended for students who missed more than three problems in the preceding self-test. Here is one example from Lesson 14.

Lesson 14, Exercise 3:

```
LESSON 8 WAS ABOUT THE "LET" COMMAND AND HOW TO USE IT
TO DEFINE A FUNCTION. FUNCTIONS OF 2 AND 3 VARIABLES
WERE DISCUSSED. INSTRUCTIONS FOR PRINTING AND DELETING
A FUNCTION WERE GIVEN.

DO YOU WANT TO REVIEW ANY OF LESSON 8?
```

The structure of Lesson 14, like that of other general reviews, allows a student who answers "yes" to branch to the review for the lesson and to review any of the lesson topics in any order.

Lesson 15: Relations and the Use of the IF Clause

Lesson 15 begins a new lesson block and introduces the most powerful programming tool: the conditional clause. The conditional (IF) clause may be appended to any of the commands so far introduced. The following AID symbols for arithmetic relations are introduced:

<	less than,
>	greater than,
<=	less than or equal,
>=	greater than or equal,
#	not equal.

The terms "positive," "negative," and "non-negative" are reviewed. The Boolean connectives "and" and "or" are also introduced. Students are required to write several programs using conditional branching.

The following are examples from Lesson 15.

Lesson 15, Exercise 14:

STUDY THIS PROGRAM.

```
49.5  TYPE X IF X > Y
49.6  TYPE Y IF X < = Y
DO PART 49.
```

IF X = 12.1 AND Y = 6, WHAT WILL AID ANSWER?

Lesson 15, Exercise 20:

WRITE A PROGRAM THAT WILL PRINT "SAME" IF ALL THREE NUMBERS X, Y, AND Z HAVE THE SAME SIGN. THE PROGRAM SHOULD PRINT "DIFFERENT" IF THE NUMBERS DO NOT ALL HAVE THE SAME SIGN.

Of the 62 exercises in Lesson 15, a large number (24) are true-false exercises. The students are required to use AID in six exercises similar to Exercise 20 above.

Lesson 16: The TO Command

Lesson 16 reviews the use of conditionals and introduces conditional branching.

The following are two examples from Lesson 16.

Lesson 16, Exercise 3:

HERE IS A PROGRAM THAT CALCULATES THE AREA OF A RECTANGLE OF LENGTH L AND WIDTH W. IF EITHER L OR W IS NEGATIVE, PART 15 IS USED TO GIVE AN "ERROR" MESSAGE.

```
14.1  DEMAND L
14.2  TO PART 15 IF L < 0
14.3  DEMAND W
14.4  TO PART 15 IF W < 0
14.5  TYPE L * W
15.1  TYPE "DO NOT USE NEGATIVE NUMBERS."
```

WHICH STEPS WILL BE DONE IF L = 5 AND W = - 3?

Lesson 16, Exercise 4:

WRITE A PROGRAM THAT WILL DEMAND A RADIUS R AND THEN CALCULATE THE AREA OF A CIRCLE WITH THAT RADIUS. USE TWO PARTS, ONE FOR THE MAIN PROGRAM AND ONE FOR AN "ERROR" ROUTINE TO BE USED IF R IS NEGATIVE.

There are 27 exercises in Lesson 16, with five multiple choice and eight "opinion" questions.

Lesson 17: Debugging Techniques

Lesson 17 concentrates on debugging techniques, showing the student how to trace a program by making a table listing the steps in order of execution.

Here is one example from Lesson 17, Exercise 3:

FOR PRACTICE, LET'S MAKE A TRACE OF THIS PROGRAM, ASSUMING A = 3.

```
31.3  DEMAND A
31.2  SET B = A↑2 - 10
31.3  SET C = A IF A > B
31.4  SET C = B IF A < = B
31.5  TYPE B
31.6  TYPE C
```

FILL IN THE VALUES OF C IN THIS TRACE (STARTING AT STEP 31.3).

STEP	A	B	C
31.1	3	-	-
31.2	3	-1	-
31.3	3	-1	?
31.4	3	-1	?
31.5	3	-1	?
31.6	3	-1	?

Of the 27 exercises in Lesson 17, 18 are similar to the exercise above. Four exercises require the student to write a complete trace with paper and pencil and check his answer by typing CTRL-T.

Lesson 18: The Indirect Use of the DO Command

In Lesson 18, the indirect use of the DO command is introduced. Up to this point, the student has been using DO commands directly to execute programs or single steps. The DO command can also be given indirectly to execute subroutines. A conditional clause is frequently used with indirect DO commands.

The following is an exercise from Lesson 18.

Lesson 18, Exercise 2:

WHEN AID COMES TO AN INDIRECT "DO" COMMAND, IT WILL DO THE STEP OR PART INDICATED AND THEN RETURN TO THE STEP AFTER THE "DO" COMMAND.

```
16.1  DO STEP 2.1 IF Q < 0
16.2  TYPE Q
 2.1  SET Q = - Q
DO PART 16
```

IF Q = 3, THE STEPS WILL BE DONE IN WHICH ORDER?

A	B	C	D
16.1	16.1	16.1	16.1
2.1	16.2	2.1	16.2
16.2	2.1		

There are 35 exercises in Lesson 18, with only one programming problem requiring the use of AID.

Lesson 19: Debugging, Permanent Storage

The first half of Lesson 19 is an optional section of tips for writing and debugging programs. This section is primarily for students who have been having difficulty with the programming problems in preceding lessons. The second half of the lesson describes the difference between core memory and disk storage and teaches the students how to store programs on the disk by using the AID file commands: USE, FILE, RECALL, and DISCARD.

Here are two examples from Lesson 19.

Lesson 19, Exercise 13:

WHAT COMMAND WOULD YOU USE TO FILE PART 29 AS ITEM 3?

Lesson 19, Exercise 17:

```
IF A NUMBER B WAS FILED AS ITEM 6, YOU WOULD RECALL IT
BY TYPING
            USE FILE 100
AND THEN WHAT?
```

Of the 32 exercises in Lesson 19, eight are requests for opinions, and four ask the student to use AID.

Lesson 20: Test of Lessons 15 to 19

Lesson 19 completes the third lesson block. Lesson 20 is a test of Lessons 15 to 19, and is structured like the tests in Lessons 6 and 13. Lesson 20 contains 27 exercises, of which two are requests for opinions. The other 25 can be grouped according to which lessons they test:

Lesson	Exercises in Lesson 20 Testing Given Lesson
15	1-1, 2, 3, 4, 5, 6
16	7, 16
17	8, 9, 10
18	11, 12, 13, 14, 15
19	17, 18, 19, 20, 21, 22, 23, 24, 25

There are six multiple-choice exercises, two "implied-choice" exercises, and 17 constructed responses.

Lesson 21: General Review of Lessons 15 to 19

Lesson 21 is a general review of the lessons tested by Lesson 20. Like other general reviews, it is optional and is recommended for students who missed more than three problems in the test.

Here is one example from Lesson 21:

Lesson 21, Exercise 8:

> LESSON 19 EXPLAINED HOW TO PLAN, WRITE, AND EDIT A PROGRAM; WHAT KINDS OF ERRORS THERE ARE AND HOW TO CORRECT THEM; AND HOW TO USE PERMANENT STORAGE.
>
> DO YOU WANT TO REVIEW LESSON 19?

A student who responds "yes" will be given the review lesson for Lesson 19.

6. The Daily Report

A daily report program was provided to inform teachers of the progress of individual students. The report lists the students in given classes, shows their current position in the curriculum, and indicates if they used the curriculum on the day of the report. The position of each student is indicated by printing the number of the last problem he completed in the tutorial lessons, in the reviews, in the summaries, and in the extra-credit problems. By comparison with previous daily reports, the teacher (or researcher) can judge about how much an individual student has progressed, and he can compare the positions of different students.

However, the daily report provides only a rough measure of progress since the students themselves control the sequence of instruction. A student may be on Lesson 5 one day and on Lesson 12 the next, either by diligently working through all the intervening lessons or by simply skipping directly to Lesson 12. Also, a student may decide to go back to review a previous lesson, and his daily report will show that he was on Lesson 12 one day and had regressed to Lesson 5 the next. Even if one assumes that students are working their way straight through the course, it is hard to get a precise measure of progress from the reports because of the varying lengths of lessons and the uneven dispersion of time-consuming programming problems. Nevertheless, the daily reports can provide an adequate indication of average rate of progress through the course and of variation in the rates of progress among students.

To illustrate use of the daily reports, one class of 39 students was selected as a sample. All these students were enrolled in the UCLA "High Potential" program. The daily report information for the class is summarized for each student and presented in Table 3. Number of days worked, number of lessons completed, and rate of progress in number of lessons per day are listed in Table 3. This particular class was chosen because of its large enrollment, but it is atypical because attendance for AID was voluntary. Of 48 possible workdays, the average number of days worked by the 39 students is 10.4 with a range from 1 to 36 days. This average is high for strictly voluntary attendance. Average number of lessons completed is 14.1 with a range from 2 to 36 lessons, indicating that the students completed slightly less than one-third of the course in the time allotted.

To illustrate the individuality of the students, progress in number of lessons completed for three students (Nos. 2, 7, and 11) from the class tabulated in Table 3 is graphed in Figure 3. Student 2 can be characterized as slow and steady, Student 7 can be characterized as fast and steady, and Student 11 can be characterized as persistent but erratic.

7. Item Analysis of INST Data

In this preliminary report, only data from first responses for Lessons 1 to 21 are analyzed. The data used were collected by the INST

Table 3

Daily Report Summary for One Class of 39 Participating Students

Student	Number of days worked	Number of lessons completed	Number of lessons completed per day
1	22	23	1.04
2	17	11	.65
3	17	27	1.59
4	2	5	2.50
5	13	18	1.38
6	13	8	.62
7	15	24	1.60
8	17	26	1.53
9	1	9	9.00
10	1	2	2.00
11	36	36	1.00
12	6	4	.67
13	3	2	.67
14	2	4	2.00
15	6	5	.83
16	2	3	1.50
17	8	20	2.50
18	13	13	1.00
19	16	22	1.38
20	9	9	1.00
21	4	8	2.00
22	19	22	1.16
23	20	31	1.55
24	18	22	1.22
25	8	20	2.50
26	11	11	1.00
27	5	8	1.60

Table 3 (cont'd)

Student	Number of days worked	Number of lessons completed	Number of lessons completed per day
28	1	2	2.00
29	6	8	1.33
30	8	11	1.38
31	2	4	2.00
32	17	21	1.24
33	10	23	2.30
34	18	17	.94
35	7	15	2.14
36	6	8	1.33
37	20	20	1.00
38	1	7	7.00
39	5	21	4.20
Mean	10.38	14.10	1.86
S.D.	7.80	9.02	1.62
N	39	39	39

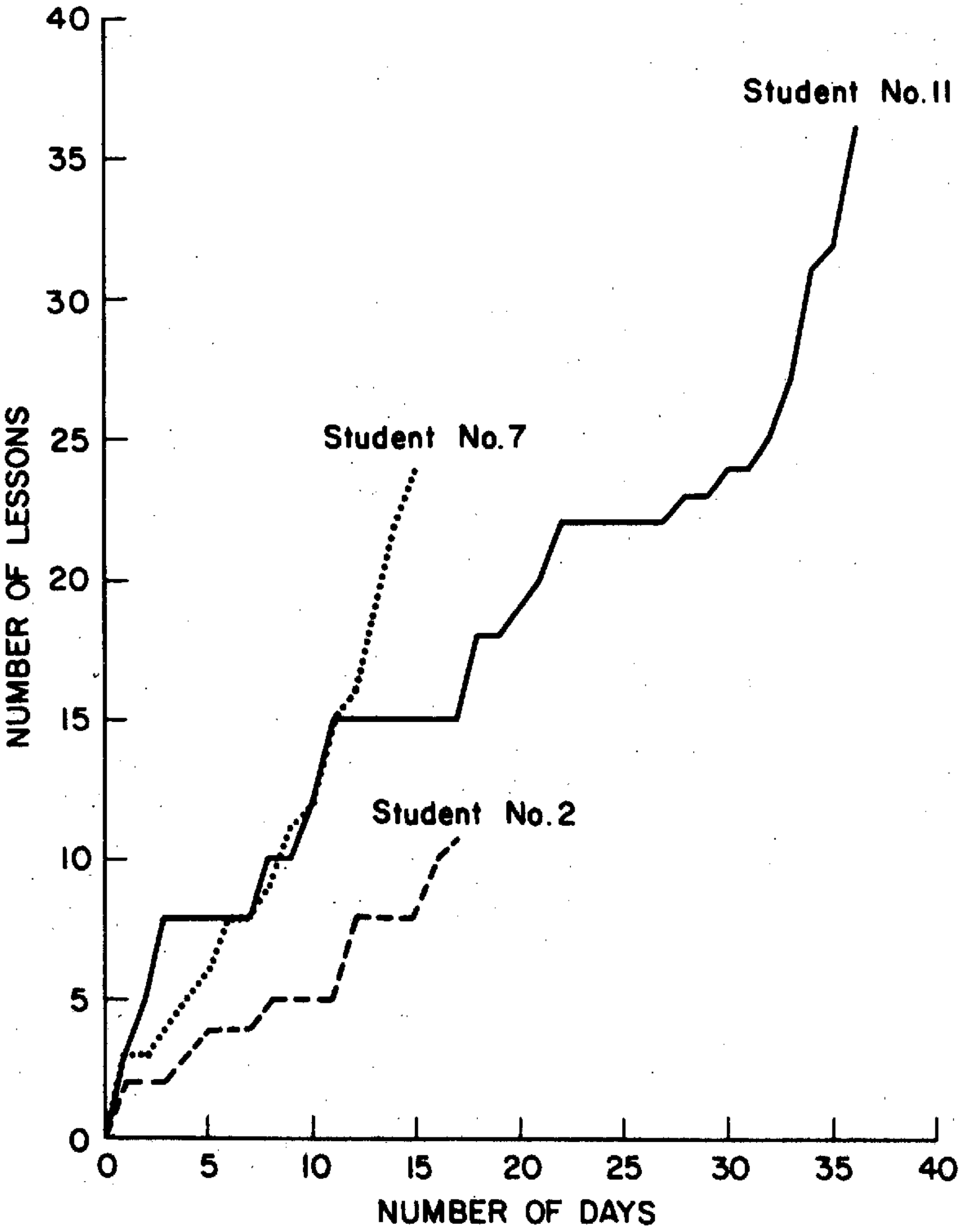


Figure 3. Student progress in number of lessons completed after each day of work in AID for three students.

instructional program. No data collected by the AID interpreter are included in this section.

Because of the unique branching structure used in the general review, Lessons 7, 14 and 21 are not included in the analysis. The self-test Lessons 6, 13 and 20 are included for comparison.

To describe the statistics reported, it is necessary to explicate the definitions of "first response" and of "correct response" used by the data collection and analysis routines. "First response" is defined to be the first of a set of contiguous responses made in any one session to a single exercise, with the following two exceptions. First, a student may, through his own volition or by automatic action of the program, repeat an exercise. In such a case, the first response made by the student the second time he encounters the exercise is counted again as a first response to that exercise. Thus a total of 35 first responses to a given exercise may be the work of only 34 students. Second, if a student terminates a session after responding incorrectly to an exercise, he will commence the next session with that same exercise; his first response in the subsequent session will be tallied as another first response to the exercise. In actual practice, the effect of these "extra" first responses is negligible.

Some responses are not considered at all in the tally of first responses. These are responses that cannot be classified as correct or incorrect. The only responses that fall into this category are question mark (student request for a hint), CTRL-G (student request for a change of problem sequence), CTRL-A (student request for a repeat of the problem), and CTRL-Z (student request for sign-off). If any of these occur as a first response, they are ignored.

Excluding the lessons and exercises not graded, 6,512 first responses made by the 68 students who completed one or more AID lessons during the first semester of the 1972-73 school year were analyzed for this report. Of the 68 students, 39 were drawn from the UCLA "High Potential" program, 26 from Ames-NASA, and three from the network of schools for the deaf.

The other critical definition used is that of "correct response." In general, the definition of "correct" is supplied by the programming and is what one would expect. If the correct answer to an exercise is "true," the student response may be "true" or "t," and any other response such as "false" or "help" or "yes" is classified as incorrect.

Some exercises have no clearly defined correct answer, however. First, requests for sequencing, "Do you want to try the extra-credit problems for this lesson?", cannot be said to have a right or wrong answer. Second, the programming of a few exercises precluded easy classification; there were seven such exercises in the 675 exercises considered. Third, some exercises ask the student to use AID to solve a stated problem. We could consider the response to be correct if the student did indeed use AID, but on a less superficial level we need some analysis of the student's use of the AID interpreter. This implies a routine that can judge the correctness of a student-written program. Unfortunately, such a routine is not available. The general solution to the problem of proving the correctness of a computer program has been shown to be recursively unsolvable (Davis, 1958). While many particular cases of this problem are solvable, it remains a deep and non-trivial problem to construct an algorithm that will prove correctness for a comprehensive set of student solutions. Some data collected by the AID interpreter have been hand-graded and will be described later in this report.

Problem Types

A summary of correct first responses by problem type is shown in Table 4. Number correct, number of first responses, and percent correct are shown for each problem type.

This summary reveals several interesting results from the item analysis. First, the proportion correct for all exercises is only 65.7%, a figure at least 10% lower than predicted after developmental testing of the program. This result is probably explained by the fact that most of the data for this analysis was obtained from students in a "High Potential" program comprising students who do not meet regular college entrance requirements but who are suspected to have higher ability than indicated by their achievement records.

In later analyses, the data used will be drawn from community college students who have been admitted unprovisionally. The original goal of the AID project was to prepare a curriculum for community college students, and these new data will provide a better assessment of the curriculum.

Second, an unexpected result is the proportion correct for multiple-choice exercises as compared with constructed responses. The proportion correct is markedly lower, 54.9%, for multiple-choice exercises compared with 66.6% for constructed responses. In presenting a lesson-by-lesson comparison of the proportion correct for multiple-choice and constructed responses Table 5 shows this result to be quite stable across the lessons.

Table 4

Number and Percent of Correct First Responses in Lessons 1 to 21 (Excluding Lessons 7, 14, 21)

	Total correct first responses	Total first responses	Percent correct
Multiple-Choice 1 Correct Choice	436	718	60.7
Multiple-Choice 2 Correct Choices	60	160	37.5
Multiple-Choice 3 or More Correct Choices	59	120	49.2
Multiple-Choice Correct Choice: NONE	23	55	41.8
Total Multiple-Choice Exercises	578	1053	54.9
Yes-No Exercises (except opinion questions)	99	133	74.4
True-False	215	254	84.6
Other Implied-Choice	47	59	79.7
Total Implied-Choice	361	446	80.9
Predicted AID Response	1282	1861	68.9
Constructed AID Command	497	798	62.3
Reported Result of AID Use	557	878	63.4
Other Constructed-Response Exercises	1003	1476	68.0
Total Constructed Response	3339	5013	66.6
Totals	4278	6512	65.7

Table 5

Percent of Correct First Responses for All Multiple-choice and All Constructed-response Exercises in Lessons 1 to 21 (Excluding Lessons 7, 14, 21)

Lesson	Percent correct multiple-choice	Total responses multiple-choice	Percent correct constructed response	Total responses constructed response
1	66.7	81	68.0	50
2	67.3	153	58.2	146
3	32.7	101	65.1	373
4	57.4	61	71.5	368
5	46.7	75	70.2	601
6	44.7	76	69.5	298
8	30.8	13	67.2	862
9	26.3	19	70.1	472
10	76.3	76	81.8	357
11	70.6	17	73.6	208
12	--	0	77.8	99
13	54.3	35	58.1	296
15	61.2	85	55.3	237
16	47.6	84	65.2	178
17	66.7	12	0.0	55
18	48.1	81	50.6	77
19	25.0	12	67.4	135
20	61.1	72	51.7	201
Totals	54.9	1053	66.6	5013

Third, the proportion correct for implied-choice exercises (80.9%) is higher than either multiple-choice or constructed-response exercises. This result might be expected since all of the implied-choice exercises imply only two choices, and about 50% of the answers would be correct if they were selected by random guessing.

Fourth, the order of percent correct among the different kinds of multiple-choice exercises is not what would be predicted from the number of correct choices available. Exercises in which there are more than one correct choice are more difficult (119/280 = 42.5%) than are exercises in which exactly one choice is correct (459/773 = 59.4%). However, exercises in which none of the listed choices is correct are more difficult (41.8%) than the other single choice exercises (60.7%), and exercises in which there are three or more correct choices are easier (49.1%) than those in which there are two correct choices (37.5%).

A more detailed analysis of the data for multiple-choice exercises will be discussed later in this report.

Student Control of Amount of Instruction

Two features of the course that allow the student to control the amount of instruction he receives are "hints" and "tells." Most of the exercises have one or more optional hints which can be requested by the student at any time, either before or after making a response. Also, in all exercises for which there is a correct answer the student may request the correct answer, i.e., a tell, at any time.

How these two controls are used is as yet undetermined, and will be the subject of future research. The item analyses of first responses, on which this report is based, does provide some evidence, however. The number of hints and the number of tells requested before first responses were tallied.

In some exercises, the hints provide information vital to the solution of the problem. This information was put into hint messages rather than into the problem text itself whenever it could be assumed that a fairly large proportion of the students would already know it. For example, if the exercise is to use AID in calculating the volume of a cylinder of given radius and height, it is essential to use the formula that gives volume in terms of radius and height. Many students can be assumed to know this formula so it was not included in the problem statement but instead was included as the first hint. In other cases, the hints suggest a strategy to be used in solving the problem. It was assumed that hints would be used freely by as many as a quarter of the students.

Table 6 shows the number of hints requested before the first response by problem type. Only 89 hints preceding first responses were requested for Lessons 1 to 21. Since there were over 6,500 first responses for these lessons, this number of requests for hints is surprisingly low. Variation between different lessons and different problem types should be viewed with skepticism because of the small number of requests.

It may be that students do not request hints until they have made at least one try, or they may believe they would be penalized for requesting hints, or they may not understand how to ask for them. Because the low number of hint requests is so unexpected it is worth pursuing in later research on use of control features.

Table 6

Number and Percent of Hints Requested in Lessons 1 to 21
(Excluding Lessons 7, 14, 21)

[illegible]	Total hints requested	Total responses	Percent hints
Multiple-Choice 1 Correct Choice	4	718	0.6
Multiple-Choice 2 Correct Choices	0	160	0.0
Multiple-Choice 3 or More Correct Choices	0	120	0.0
Multiple-Choice Correct Choice: NONE	0	55	0.0
Total Multiple-Choice Exercises	4	1053	0.4
Yes-No Exercises (except opinion questions)	7	133	5.3
True-False	0	254	0.0
Other Implied-Choice	1	59	1.7
Total Implied-Choice	8	446	1.8
Predicted AID Response	30	1861	1.6
Constructed AID Command	19	798	2.4
Reported Result of AID Use	17	878	1.9
Other Constructed-Response Exercises	11	1476	0.7
Total Constructed Response	77	5013	1.5
Totals	89	6512	1.4

Requests for the correct answer were expected to be relatively low as a first response, about 5% as a first response and about 10% as a second, or later, response. Table 7 shows the number of first responses requesting the correct answer by problem type. Two hundred forty-two of the 6,512 first responses, or 3.7%, were requests for the answer. It is interesting that there was a greater proportion of such requests for multiple-choice exercises (4.6%) than for constructed-response exercises (3.8%), indicating that multiple-choice exercises are more difficult.

In the analysis of structural variables affecting problem difficulty, proportion correct is taken as the measure of difficulty. An interesting comparison could be made by using the number of requests for answers as a measure of difficulty. This was not done in the current analysis because the number of subjects was too small to permit meaningful application of the stepwise multiple linear regressions used below.

Lesson to Subtest Relationship

The percentages correct by lesson listed in Table 8 show little variation, with the exception of Lesson 17, and it can be inferred that the curriculum is reasonably consistent in difficulty. Lessons 10 and 12 appear to be the easiest, and Lessons 17 and 18 seem to be the most difficult.

Two of the more difficult lessons are tests (Lessons 13 and 20), and the average percent correct for the three tests (Lessons 6, 13, and 20) is 60.7%, which is slightly lower than the average for tutorial lessons, 63.3%. To compare tutorial lessons and tests, each test was divided into five subtests each consisting of the exercises associated with one of the five preceding tutorial lessons, and the percent correct

Table 7

Number and Percent of Requests for the Correct Answer (Tells) in Lessons 1 to 21 (Excluding Lessons 7, 14, 21)

	Total tells requested	Total responses	Percent tells
Multiple-Choice 1 Correct Choice	39	718	5.4
Multiple-Choice 2 Correct Choices	5	160	3.1
Multiple-Choice 3 or More Correct Choices	3	120	2.5
Multiple-Choice Correct Choice: NONE	2	55	3.6
Total Multiple-Choice Exercises	49	1053	4.7
Yes-No Exercises (except opinion questions)	1	133	0.8
True-False	0	254	0.0
Other Implied-Choice	0	59	0.0
Total Implied-Choice	1	446	0.2
Predicted AID Response	82	1861	4.4
Constructed AID Command	34	798	4.3
Reported Result of AID Use	14	878	1.6
Other Constructed-Response Exercises	62	1476	4.2
Total Constructed Response	192	5013	3.8
Totals	242	6512	3.7

Table 8

Comparison of Scores in Lessons with Scores in Related Exercises in Subtests

Lesson number	Proportion correct in lesson (%)	Proportion correct in test (%)
1	67.9	85.0
2	63.3	75.0
3	58.2	71.4
4	69.4	46.7
5	67.6	52.7
8	66.7	56.0
9	68.4	56.2
10	80.8	54.4
11	73.3	80.6
12	77.8	52.4
15	66.6	61.7
16	61.0	59.1
17	11.9	51.5
18	49.4	52.7
19	67.4	54.6
Mean	63.3	60.7
S.D.	16.1	11.7
N	15	15

Correlation coefficient (R) = .165

$R^2 = .027$

on each subtest was compared with the proportion correct on the lesson tested by that subtest. Percentages correct on these subtests are given in the second column in Table 8 and are associated with the appropriate lesson. There is essentially no correlation between percent correct on lessons and percent correct on the associated internal tests (R^2 = .027). The predictive power of lesson scores for individual students will be explored in a later analysis of individual performance.

Multiple-choice Exercises

In the first 21 lessons of the AID course there are 80 multiple-choice exercises. In all of these exercises the choices are listed and labeled with letters, and if there is more than one correct choice, the students are expected to type the labels for all of the correct choices. Two formats are used:

```
(Vertical)          A.  TYPE
                    B.  DELETE
                    C.  SET
                    D.  DO
                    N.  NONE OF THE ABOVE

(Horizontal)

                      A                     B

                1.2  SET X=1          1.2  DEMAND X
                1.3  TYPE X↑2         1.3  TYPE X↑2
                1.4  TYPE X↑3         1.4  TYPE X↑3
```

Generally, the vertical form is used if each choice can be printed on one line, and the vertical form is used if several lines are required to print single choices.

The data for multiple-choice exercises are divided into four classes, according to the number of correct choices (one, two, more than two, none). The number of correct first responses, total number of first responses, and percent correct on first responses for each problem type are shown in Table 4 presented earlier. The class of multiple-choice exercises with one correct choice is sufficiently numerous (56 exercises) to warrant a more detailed inspection of the data. These exercises were subdivided into the 10 following classes, and proportion correct was calculated for each of these classes.

(a) Algebraic Equivalence I. The student is given an algebraic expression and is asked to choose an equivalent expression.

Example:

```
        TYPE 10/7 - 5 - 2
  COULD BE WRITTEN
        A.  TYPE 10/(7-5) - 2
        B.  TYPE (10/7) - (5-2)
        C.  TYPE (10/7) - 5 - 2
        N.  NONE
```

(b) Algebraic Equivalence II. The student must choose a described algebraic expression.

Example:

```
        WHICH COMMAND WILL CAUSE AID TO MULTIPLY 25
        BY 5 AND DIVIDE BY 3?
        A.  TYPE 25 × 5/3
        B.  TYPE 25 * 5/3
        C.  TYPE 25(5/3)
        N.  NONE OF THE ABOVE
```

150

(c) Choice of AID Programs. The student must choose which of two routines will produce a specified result.

Example:

WHICH PART TYPES THE SMALLEST OF TWO NUMBERS?

```
A.  3.1  TYPE R IF R > S
    3.2  TYPE S IF R <= S

B.  4.1  TYPE R IF R < S
    4.2  TYPE S IF R >= S
```

(d) Mechanics. The student is asked about the mechanics of using the instructional program or the AID interpreter.

Example:

WHICH COMMAND WILL STOP THE AID INTERPRETER, AND RETURN YOU TO YOUR LESSON?

A. CTRL-H
B. RETURN
C. INST
D. CTRL-T

(e) Syntax of AID Commands. The student must decide which of a list of commands is syntactically correct.

Example:

WHICH OF THESE COMMANDS WILL CAUSE AID TO STOP AND WAIT FOR YOU TO TYPE A VALUE FOR S?

```
A.  3.7  ASK S = *
B.  3.7  DEMAND S
C.  3.7  REQUEST S =
D.  3.7  DEMAND S
```

(f) Semantics of AID Commands. The student must choose from among syntactically correct commands those that effect a specified action.

Example:

WHICH COMMAND(S) WILL NEVER GIVE A NEGATIVE
NUMBER NO MATTER WHAT THE VALUE OF X IS?

```
A.  TYPE X/5
B.  TYPE SGN(X)*(X/5)
C.  TYPE X/(-5)
D.  TYPE SGN(X)*(X/(-5))

N.  NONE
```

(g) Boolean Equivalence I. A Boolean statement is given, and the student must choose an equivalent statement.

Example:

```
X <= M
```

MEANS THE SAME AS WHICH OF THESE?

```
A.  M # X
B.  M <= X
C.  M >= X
N.  NONE
```

(h) Boolean Equivalence II. The student must choose a Boolean statement, given a description of it in English.

Example:

WHICH MEANS "Q IS NON-NEGATIVE?"

```
A.  Q > 0
B.  Q >= 0
C.  Q < 0
D.  Q <= 0
N.  NONE
```

152

(i) Sequence of Execution. The choices are lists of step numbers, and the student must choose the list that executes the commands in a specified order.

Example:

```
16.1  DO STEP 2.1 IF Q < 0
16.2  TYPE Q
 2.1  SET Q = - Q
DO PART 16
```

IF Q = 3, THE STEPS WILL BE DONE IN WHICH ORDER?

A.	B.	C.	D.
16.1	16.1	16.1	16.1
2.1	16.2	2.1	16,2
16.2	2,1		

(j) Miscellaneous

The number of exercises, number of first responses, and percent correct on first responses for each of the above classes of multiple-choice exercises are given in Table 9.

It is difficult to draw firm conclusions from this subdivision of problems but these data present some clues to student behavior. The two classes with the highest proportion correct are (d) Mechanics, and (g) Boolean Equivalence I. The two classes with the lowest proportion correct are (c) Choice of AID Programs and (h) Boolean Equivalence II. The proportion correct for class (i) Sequence of Execution is only 51.6% over 13 exercises. It may be that the students have difficulty understanding order of execution of commands and that the curriculum should emphasize this area. The data from later lessons on loops will be studied to see if they also suggest some revision in this aspect of the course.

Table 9

Number of Exercises, Number and Percent of Correct First Responses in the 10 Classes of Single Choice Multiple-Choice Exercises in Lessons 1 to 21 (Excluding Lessons 7, 14, 21)

Exercise type	Number of Exercises	Total correct first responses	Total first responses	Percent correct
(a) Algebraic Equivalence I	6	38	62	61.3
(b) Algebraic Equivalence II	2	12	25	48.0
(c) Choice of AID Programs	3	13	38	34.2
(d) Mechanics	6	74	88	84.1
(e) Syntax of AID Commands	7	61	91	67.0
(f) Semantics of AID Commands	2	16	31	51.6
(g) Boolean Equivalence I	3	38	49	77.6
(h) Boolean Equivalence II	2	13	34	38.2
(i) Sequence of Execution	13	97	188	51.6
(j) Miscellaneous	12	74	112	66.1
Total	56	436	718	60.7

In multiple-choice exercises, several variables other than problem type may contribute to difficulty. One of these variables is the number of choices given; presumably the larger the set of possible answers, the more difficult the choice. The range of choices in these exercises is from two to five. To estimate the effect of the number of choices, the correlation between proportion correct and number of choices was calculated, using only those exercises (49 of the 56) for which there were 10 or more first responses. This correlation was .000, indicating that the number of choices bears no linear relationship to problem difficulty. This result is not conclusive since number of choices and whether an exercise is in Class (c) or (d) are statistically dependent. Further, Class (c) is the most difficult class but has an average of only two choices per exercises, whereas Class (d) is the least difficult class but has an average of 4.3 choices per exercises.

A further analysis was made by using multiple step-wise linear regression with observed proportion correct, x_1, as the dependent variable, and using the following independent variables:

x_2 number of choices

x_3 1 if one of the choices is
N. NONE OF THE ABOVE
0 otherwise

x_4 1 if the exercise is in Class (a) Algebraic Equivalence I
0 if not

x_5 1 if the exercise is in Class (b) Algebraic Equivalence II
0 if not

x_6 1 if the exercise is in Class (c) Choice of AID Programs
0 if not

x_7 1 if the exercise is in Class (d) Mechanics

0 if not

x_8 1 if the exercise is in Class (e) Syntax of AID Commands

0 if not

x_9 1 if the exercise is in Class (f) Semantics of AID Commands

0 if not

x_{10} 1 if the exercise is in Class (g) Boolean Equivalence I

0 if not

x_{11} 1 if the exercise is in Class (h) Boolean Equivalence II

0 if not

x_{12} 1 if the exercise is in Class (i) Sequence of Execution

0 if not

x_{13} 1 if the exercise is in Class (j) Miscellaneous

0 if not.

This regression was computed for the 49 exercises for which there were 10 or more first responses. In Table 10, the variables are listed in the order in which they were "stepped" into the regression equation, and the values of R and R^2 are given. The variables in the regression accounted for 38% of the variance of the observed proportion correct. This leaves 62% of the variance unaccounted for, indicating that there are other variables, as yet unidentified, that affect problem difficulty in this group of exercises.

The two variables that entered into the regression first, x_7 and x_6, are the variables for membership in Classes (d) and (c), the same two classes that are statistically dependent on number of choices, and that the third variables to enter into the regression is x_2, the number of

Table 10

Order in Which Independent Variables Were Entered into a Multiple Stepwise Regression with the Associated Correlation Coefficients (R)

Order	Variable	R	R^2	Increase in R^2
1	x_7	0.33	0.11	0.11
2	x_6	0.40	0.16	0.05
3	x_2	0.52	0.27	0.11
4	x_5	0.54	0.29	0.02
5	x_8	0.55	0.31	0.02
6	x_3	0.56	0.32	0.02
7	x_{10}	0.59	0.35	0.03
8	x_{12}	0.60	0.36	0.01
9	x_{13}	0.61	0.37	0.01
10	x_{11}	0.61	0.38	0.01
11	x_4	0.61	0.38	0.00
12	x_9	0.61	0.38	0.00

choices. x_7 and x_6 together accounted for 16% of the variance in the observed proportion correct, whereas the addition of x_2 increased that figure by 11%, indicating that the number of choices is of significant effect even after Classes (c) and (d) are taken into account. The fact that x_2 entered into the regression before any of the other variables (except x_7 and x_6) supports the hypothesis that the number of choices contributes to problem difficulty.

Constructed-response Exercises

The constructed-response exercises form by far the most numerous class of exercises in the course (390 out of 675), and represent more diversity than either multiple-choice or implied-choice exercises. One reasonably homogeneous subclass of the constructed-response exercises, the "predicted AID response" exercises, was chosen for more detailed study. This class contains 135 exercises, and the data contains 1,861 responses to be analyzed.

In the "predicted AID response" exercises, the student is asked to predict the result if a given command were to be executed by the AID interpreter. For these exercises, a simple classification scheme is not possible because of the interaction of variables that contribute to the relative difficulty of each exercise. For this reason, multiple stepwise linear regression was used to find which of these variables had the greatest influence on problem difficulty. Only the 104 exercises with 10 or more responses were used in this study. The dependent variable, x_1, to be predicted by the regression equation was observed proportion correct. Eleven independent variables were identified and values assigned to each exercise. The independent variables were the following:

x_2 Number of arguments for TYPE commands. The problem may contain several TYPE commands, each of which can have one or more arguments. The number of arguments for all TYPE commands are summed to give the value of x_2.

x_3 Number of AID commands. All of the commands displayed in the problem are counted for x_3. If the exercise is a continuation of a preceding exercise or exercises, the commands displayed in the preceding exercise may also be counted into this variable. Variables x_3 to x_{11} are concerned with this extended set of commands.

x_4 Proportion of AID commands to be used. Some commands may be extraneous. x_4 is obtained by dividing the number of commands needed by the total number of commands displayed.

x_5 Number of function calls. x_5 is the total number of function calls in the displayed commands, including both defined functions and standard AID functions.

x_6 Number of clauses. The number of IF, FOR, and TIMES clauses are counted.

x_7 Number of substitutions required. x_7 is a count of the substitutions required for the correct solutions, including substitution of numbers for real variables and substitution of expressions for real variables. Substitution of function definitions is not counted. As an example, suppose these commands were given:

```
LET F(X) = X+2
SET A = 5
TYPE F(27/A)
```

The substitution of 5 for A is counted, and the substitution of 27/A for X in the expression X+2 is counted, so the value of x_7 is 2.

x_8 Imbeddedness of arithmetic expression. The student may be asked to evaluate some arithmetic expression, possibly after performing one or more substitutions. To measure imbeddedness, we used a "completed" expression, which is obtained from the expression displayed by making all required substitutions and by inserting all implicit parentheses. This completed expression is the basis for calculating variables x_8 and x_9. For x_8, imbeddedness is measured by finding the maximum number of left parentheses that can be counted before a right parenthesis is encountered. As an example, consider these commands:

```
SET X = 2
TYPE X + X↑3
```

The completed expression derived from this would be

$$2 + (2\uparrow3)$$

and the imbeddedness variable x_8 would have the value 1 since there is only one pair of parentheses.

x_9 Number of operations. The value of x_9 is found by counting the number of operation symbols in the completed arithmetic expression.

x_{10} Number of exponentiation operators. The occurrences of the symbol ↑ in the extended set of displayed commands are counted for x_{10}.

x_{11} Number of implied parentheses. Here the completed arithmetic expression is not used for a basis for calculation. What is counted is the number of pairs of parentheses that are used implicitly. For example, in this command,

SET X = 2↑3/4,

there is one pair of implied parentheses, (2↑3)/4, and the value of x_{11} is one.

x_{12} Number of preceding "predicted AID response" exercises in the same lesson.

Table 11 shows the order in which the variables were "stepped" into the regression equation and the values for R and R^2. The variables used accounted for only 21% of the variance in proportion correct. The variable with the greatest effect was x_9, the number of operations. This variable is related to, but not identical with, the operations variable used by Loftus (1970) and by Jerman (1971) in similar models of problem-solving difficulty. In both cases, the operations variable was found to be significantly related to difficulty. The second most effective variable was x_6, the number of clauses, and the third was x_5, the number of function calls.

The variables x_{11}, x_2, x_7, x_3 and x_8 all added essentially nothing to the value of R^2. In fact, the first six variables accounted for 20% of the variance and the next four together added only 1% more.

8. Analysis of Selected Aid Data

Although this report is concerned primarily with data collected by INST, a large body of data was also collected by the AID interpreter, and a sample of this data was selected and analyzed by hand. These data

Table 11

Order in Which Independent Variables Were Entered into a Multiple Stepwise Regression with the Associated Correlation Coefficients (R)

Variable	R	R^2	Increase in R^2
x_9	0.30	0.09	0.09
x_6	0.41	0.17	0.08
x_5	0.43	0.18	0.01
x_{10}	0.43	0.19	0.01
x_{12}	0.44	0.19	0.00
x_4	0.44	0.20	0.01
x_8	0.45	0.20	0.00
x_3	0.45	0.20	0.00
x_7	0.45	0.21	0.00
x_2	0.46	0.21	0.00
x_{11}	0.46	0.21	0.00

were collected by the AID interpreter as students used it to solve problems given by the instructional program. The data consists simply of the characters typed by the student together with necessary bookkeeping information such as student number, problem identifiers, and date.

The problem of analyzing these data from the AID interpreter is basically unsolvable. As was discussed earlier, no effective procedure exists that can determine if a student-written program is equivalent to a given, correct program. An approximation of this procedure may be possible, but for the moment data collected by the AID interpreter must be analyzed by hand. However, no evaluation of the course will be complete without an examination of the programs written by the students. In a later report, more sophisticated and extensive analyses of the AID interpreter data will be made.

In the first 21 lessons of the course, there are 58 exercises that require the student to use the AID interpreter. These exercises vary greatly in difficulty. Some simply ask the student to call the AID interpreter and copy a single command; others ask the student to apply recently learned principles to solve complex problems. Of the 58 exercises requiring the use of AID, 28 are essentially copying tasks. Of the remaining 30 exercises, there are approximately 20 that can serve as genuine tests of the students' ability to use the AID interpreter. Eight of these exercises were chosen for analysis, the results of which are summarized in Table 12.

All these exercises follow the same form. The instructional program presents a problem and asks the student to use the AID interpreter to solve it (the student may request hints at this point, but he cannot

Table 12

Summary of Student Performance on Selected "Use AID" Exercises

Lesson no.	Problem no.	No. students who used AID	No. correct solutions: first use of AID	% correct: first use of AID	No. correct solutions: all AID uses	% correct all AID uses
5	30	13	2	15.4	12	92.3
5	31	12	3	25.0	11	91.7
8	9	20	11	55.0	15	75.0
8	27	16	4	25.0	6	37.5
8	28	12	4	33.3	12	100.0
9	3	18	3	16.7	7	38.9
12	4	15	6	40.0	9	60.0
15	15	11	2	18.2	8	72.7
Totals	8	117	35	29.9	80	68.4

give an answer until he has called the AID interpreter). Next, the student calls the AID interpreter and attempts to solve the problem. This step is called the "first use of AID" in Table 12. The student may at this time write a program, try it out, replace one or more commands, correct typographical errors, etc. Finally, when he is satisfied with his solution, the student switches back to the instructional program and is asked for a report on the results he obtained. If his reported results are incorrect, the instructional program gives him additional instruction and asks him to switch back to AID and try again.

Table 12 shows the number of students who correctly solved each problem on the first try and the number of students who eventually arrived at a correct solution. The percent correct for first AID use is quite low (29.9%). Since these problems are among the most difficult in the entire course, this percent is expected to be lower than the 66% average for other exercises. Because of the additional help given by the instructional program after the first use of AID, the percent correct is expected to rise when subsequent uses are considered, and, in fact, it does rise to 68.4%. The difference between proportion correct for first try and for subsequent tries is most pronounced for Lesson 5, Exercise 30. Only two out of 13 students solved the problem on the first try, but 12 of them solved it eventually.

During the hand-grading of these exercises, notes were made on the most common errors for each problem. Following are the problem statements (including subproblems) for each of the eight exercises, including comments on the errors observed while the exercises were being graded.

Problem 5-30:

1 CENTIMETER = .3937 INCHES. START AID AND CONVERT THE FOLLOWING LENGTHS TO CENTIMETERS:

6.9 INCHES
7.445 INCHES
23.9753 INCHES

Problem 5-30.1:

FROM THE ABOVE CALCULATIONS,

6.9 INCHES = ??? CENTIMETERS

Problem 5-30.2:

7.445 INCHES = ??? CENTIMETERS

Problem 5-30.3:

23.9753 INCHES = ??? CENTIMETERS

The students must solve this problem with direct TYPE and SET commands since indirect (stored) commands have not yet been introduced. One solution would be

```
SET X = .3937
TYPE 6.9/X, 7.445/X, 23.9753/X
```

On this problem only two of 13 students were correct on the first try. Seven of the students made algebraic errors; six used multiplication instead of division, and one used .3937/6.9 instead of 6.9/.3937. Two students made algebraic errors on the second try, one repeated the error of using multiplication rather than division, and one came up with a unique use for π (6.9/3.1416).

Although a number of syntax errors were made, they were almost all typographical errors and were corrected by the student without any intervening instruction. Only one student was unable to correct his errors in syntax. It is likely that had the students known the correct algebraic formulation, all but one would have produced a correct solution on the first try.

Problem 5-31:

> TO FIND THE NEW AMOUNT IN A SAVINGS ACCOUNT, CALCULATE THE INTEREST AND ADD IT TO THE LAST BALANCE. START AID AND CALCULATE THE INTEREST AND THE NEW BALANCE AFTER ONE YEAR FOR AN ACCOUNT WITH AN INTEREST RATE OF 4.5 PERCENT PER YEAR AND A PREVIOUS BALANCE OF $3274.86. (ASK FOR A HINT IF YOU NEED ONE.)

Problem 5-31.1:

> WHAT IS THE INTEREST ON THE ABOVE ACCOUNT TO THE NEAREST PENNY?

Problem 5-31.2:

> WHAT IS THE NEW BALANCE IN THE ACCOUNT?

This problem can be done in several ways. One solution, which was not used by any of the students, would be

```
SET P = 3274.86
SET I = .045 * P
TYPE I, P+I
```

Only three out of 12 students were correct on their first try, but 11 eventually succeeded. Two students made syntax errors of the same kind; they both used the symbols $ and % in an arithmetic expression. Two students calculated the new balance directly, without calculating the interest independently, which left them unable to answer the question in the first subproblem. Obviously, the problem should have been explicit in its request for two independent calculations. Four students used 4.5 as the interest rate rather than .045, and two students used the wrong arithmetic operation. Of the 11 students who eventually gave correct answers, only four used variables. Here again it would seem that the major difficulty is with algebra, rather than with the use of AID.

Problem 8-9:

```
START AID AND USE A "LET" COMMAND TO DEFINE A FUNCTION
THAT GIVES THE RECIPROCAL OF X.  USE YOUR FUNCTION TO
FIND THE RECIPROCAL OF

     119.4
     67.3↑3.
     6+4
```

Problem 8-9.1:

```
WHAT IS THE RECIPROCAL OF 119.4?
```

A correct solution of this exercise would be

```
LET F(X) = 1/X
TYPE F(119.4), F(67.3↑3), F(6+4)
```

Over half of the students solved this on the first try, and three-quarters of them were eventually correct. The LET command was introduced in Lesson 8, and this was the second exercise in which the students used AID in this lesson. Five out of 20 students made syntax lerrors that they could not correct, all in the LET command. Three students made algebraic errors, using as the formula for reciprocal .1 x, $x^2/2$, and $1/x^2$. Two students who defined the function correctly redefined it needlessly before each function call.

In this exercise, the syntax of the newly introduced LET command was the source of most errors.

Problem 8-27:

USE AID TO DO THIS PROBLEM. DEFINE A "VOLUME" FUNCTION THAT WILL GIVE THE VOLUME OF A CYLINDRICAL TANK OF RADIUS R AND HEIGHT H.

(VOLUME = 3.1416 TIMES THE RADIUS SQUARED TIMES THE HEIGHT.)

FIND THE VOLUMES OF 2 TANKS:

TANK A IS 57.5 FEET HIGH AND HAS A RADIUS OF 18.6 FEET.

TANK B IS 65.4 FEET HIGH AND THE RADIUS IS 19.3 FEET.

Problem 8-27.1:

WHAT IS THE VOLUME OF TANK B?

Problem 8-27.2:

WHICH TANK HOLDS MORE, TANK A OR TANK B?

A correct solution for this exercise is

```
LET V(R,H) = 3.1416 * R↑2 * H
TYPE V(18.6, 57.5), V(19.3, 65.4)
```

Only four of 16 students succeeded on this exercise on the first try, and only six eventually succeeded. Seven students did not define a function at all but solved the problem arithmetically. Most of the students who tried to use a function were confused about the syntax of the LET command, the use of functions of more than one variable, or the use of dummy variables. Two students used the wrong algebraic formula (πrh instead of $\pi r^2 h$) even though the problem gave the formula.

Here it seems the course moved too fast in introducing functions of more than one variable so soon after simple functions were introduced. Some of the preceding exercises which required substitution of expressions containing real variables for the dummy variables used in function definitions may have exacerbated an existing confusion about variables. The next exercise which required a function of only one variable proved much less difficult.

Problem 8-28:

> USE AID TO DO THIS PROBLEM. DEFINE A FUNCTION TO CONVERT DEGREES FAHRENHEIT TO DEGREES CENTIGRADE. THEN CONVERT THESE TEMPERATURES TO CENTIGRADE:
>
> 0, 10, 32, 72, 212

Problem 8-28.1:

> 72 DEGREES FAHRENHEIT IS EQUAL TO HOW MANY DEGREES CENTIGRADE?

A correct solution is

```
LET C(F) = (F-32) * 5/9
TYPE C(0), C(10), C(32), C(72), C(212)
```

and one-third of the students gave this solution, or an equivalent one, on their first try. All of the students arrived at a correct solution on a later try.

Seven out of 12 students made algebraic errors, five of them identical: x - 32 * 5/9 instead of (x-32) * 5/9. Omission of necessary parentheses in an algebraic expression with only two operations indicates a basic confusion about the hierarchy of operations.

Problem 9-3:

WRITE A FUNCTION, H(A,B), THAT WILL FIND THE HYPOTENUSE OF A RIGHT TRIANGLE IF THE LENGTHS OF THE OTHER TWO SIDES ARE GIVEN BY A AND B. START THE AID INTERPRETER AND TEST YOUR FUNCTION ON THESE TRIANGLES;

1. A=3, B=4
2. A=12, B=12
3. A=1/2, B=3/4
4. A=9, B=13.2

Problem 9-3.1:

WHAT IS THE HYPOTENUSE OF THE FIRST TRIANGLE ABOVE?

(WHERE A=3, B=4)

Problem 9-3.2:

WHAT IS THE HYPOTENUSE OF THE TRIANGLE WITH SIDES A=1/2 AND B=3/4?

This is the first exercise using AID after the standard AID function SQRT is introduced, and the correct solution is

```
LET H(A,B) = SQRT(A↑2 + B↑2)
TYPE H(3,4), H(12,12), H(1/2,3/4), H(9,13.2)
```

The students did very poorly on this problem. Three of the 18 were correct on their first try, but only seven eventually achieved a correct solution. Most of the errors were syntax errors either in the LET command or in the function call. Like Problem 8-27, there were a number of errors in the use of parentheses:

1/2↑2 + 3/4↑2 was used for (1/2)↑2 + (3/4)↑2

SQRT(A↑2) + (B↑2) was used for SQRT(A↑2 + B↑2)

SQRT A was used for SQRT(A)

SQRT(A)↑2 + (B)↑2 was used for SQRT(A↑2 + B↑2)

916.25↑1/2 was used for 916.25↑(1/2)

Several students did not know the correct formula and used:

$(\sqrt{A} + \sqrt{B})^2$

$(A^2) \div + (B^2)$

Three students did not define a function but did produce correct specific solutions for all four triangles.

Problem 12-4:

```
START AID AND WRITE A PROGRAM (PART) THAT WILL ASK YOU
FOR 3 NUMBERS A, B, AND C, AND THEN GIVE YOU THE AVERAGE
OF THE 3 NUMBERS.  AFTER YOU HAVE TESTED YOUR PROGRAM,
USE IT TO FIND THE AVERAGE OF

     A = 179.053
     B = 23.7
     C = 271.0015
```

Problem 12-4.1:

```
WHAT ANSWER DID YOU GET?
```

This is the first exercise in the eight hand-graded exercises that requires the use of a stored program. One correct solution is

```
4.1  DEMAND A
4.2  DEMAND B
4.3  DEMAND C
4.4  TYPE (A+B+C)/3
```

This program would be executed by giving the command

```
DO PART 4
```

Six of 15 students solved this problem on the first try, and another three succeeded later. The majority of errors were algebraic; seven students used incorrect formulas:

172

```
A + B + B/3        (used by four students)
ABC/3              (used by two students)
3/A + B + C        (one student)
```

Of the nine correct solutions, four were not debugged, and the others were tested for only one set of poorly chosen values. Here again, as in Problem 9-3, some students produced specific but not general solutions.

Problem 15-15:

> USING AID, WRITE A PROGRAM THAT WILL FIND THE SMALLER OF TWO NUMBERS X AND Y. TRY SEVERAL DIFFERENT VALUES OF X AND Y.

Problem 15-15.1:

> DID YOUR PROGRAM WORK?

An economical solution to this problem is

```
9.1  DEMAND X
9.2  DEMAND Y
9.3  TYPE X IF X < Y
9.4  TYPE Y IF Y <= X
```

For this exercise, two out of 11 students were correct on their first try, and only three did not solve the problem at all.

The greatest difficulty was in inputting values for the two variables. Eight students had trouble with this. Several of them tried rather ingenious (but incorrect) variants of the FOR clause:

```
DO PART 9 FOR X = 6, Y = -2.
DO PART 9 FOR (X,Y) = (4,7), (8,4), (4321,6493).
```

Four of the eight correct solutions were well debugged, three were tested for only one set of values, and one was not tested at all.

The above discussion of the performance of students on eight selected exercises is not intended as a definitive survey or as a final evaluation of the curriculum but as an indication of the ways students solve problems and of guidelines for future analyses of student interaction with the AID interpreter. It is clear that some analysis of errors is desirable and leads to more meaningful interpretation than correct-incorrect grading alone.

In summary, the most common difficulties of the students involved algebra more than programming and centered around the following aspects:

(1) Use of parentheses. Whether or not students would have correctly formulated the algebraic expressions if they were using ordinary algebraic notation rather than AID notation is not known, but one suspects that in many cases they could not have done so.

(2) Use of standard algebraic formulas. The number of errors made in standard formulas is a strong indication that the course was delinquent in not providing these formulas wherever needed.

(3) Use of functions of more than one variable.

9. Conclusions

In this preliminary report, we discussed the interaction between students and a tutorial computer-assisted course in programming. Rates of progress as measured by the Daily Report program were discussed using one class of students as an example. A classification scheme for the exercises was devised, and the first responses to exercises in the first 21 lessons were examined by problem type and by lesson. Two classes of exercises, one set of 49 multiple-choice exercises and one set of 104 constructed-response exercises, were analyzed in detail to determine the structural variables that affected problem difficulty. Multiple stepwise linear regression was used to model exercise characteristics as determiners of exercise difficulty. Eight exercises that required the students to use the AID interpreter were analyzed by hand.

Some of the results of these investigations were unexpected, some confirmed predictions, and some were anomalous. First, the proportion of correct initial responses (65.7%) was lower than hoped for. There are no firm guidelines giving the optimal proportion correct for CAI. Some curriculum designers aim for nearly 100% correct, others have claimed 85% to be ideal. We assumed that there is little learning if the success rate is very high. On the other hand, with a very low success rate, students are likely to become discouraged. Until further research suggests an optimum success rate, each curriculum designer is left to his own best judgment. In our case, we believed that the proportion of correct first responses should be about 75%, which is nearly 10% higher than the proportion correct attained by the students studied in this investigation. It should be noted, however, that proportion correct summed over students is a simplistic measure of effective instruction, particularly for computer-assisted instruction where complex branching strategies are used, and better measures are obtained by examining the distributions of correct answers achieved by individual students over all exercises and the success with which students meet the goals of the course.

Second, proportion correct among all the lessons was quite stable, with only four lessons in the first 21 deviating from the total 65.7%

correct by more than 10%. The very low 11.9% for Lesson 17 was unexpected and remains unexplained.

Third, examination of proportion correct for different problem types indicated that the multiple-choice exercises were more difficult than the constructed-response exercises (54.9% correct compared to 66.6%).

Fourth, the multiple-choice exercises that were examined closely were those with only one correct choice, excluding the exercises for which the correct answer is "none of the above." The total proportion correct for these exercises was 60.7%, but the variation between different subclasses is striking. The easiest multiple-choice exercises concerned the mechanics of operating the instructional program and the AID interpreter; the most difficult were those in which the student had to choose which of two given AID programs would produce a specified result.

Fifth, we found no simple, unambiguous relationship between difficulty and the number of correct answers in multiple-choice exercises. The category labeled "multiple-choice, 1 correct choice" was easiest in terms of proportion of correct first responses (60.7%), but the other single choice category, "multiple-choice, correct choice: none," was the most difficult (41.8%). Further, the category with three correct choices was easier (49.2%) than the category with two correct choices (37.5%).

Sixth, a more sophisticated approach was taken in the analysis of performance on a selected class of constructed-response exercises. The exercises used were those in which the student predicts the result of using given AID commands. A dozen structural variables were defined and a stepwise linear regression performed to determine the parameters that could estimate the difficulty of these and similar exercises. We succeeded in obtaining a model that accounted for 21% of the obtained variance in problem difficulty, a reasonable but not spectacular fit for a first model. Three structural variables (the number of operations, the number of clauses, and the number of function calls) accounted for 18% of the variance. All these variables correlated negatively with proportion correct.

The investigation suggested a few general conclusions about student behavior. The mechanics of using the instructional system did not cause undue difficulty, as shown by the performance of the students on Lesson 1 and the subsequent test of that lesson, as well as by their performance on the Multiple-choice Exercise Class. It remains unclear whether or not the students learned to use the various control features as effectively as they could (the number of requests for hints was much lower than expected), and further investigation is needed of this aspect of the instructional system.

The use of direct commands, and the concept of stored programs and their execution were readily mastered. The syntax of AID commands, except for the LET command, caused little difficulty. Although many students made syntactical errors in using the AID interpreter, most of these errors were immediately corrected by the students.

The three areas in which the students seemed least adept were:

(1) Hierarchy of arithmetic operations.

(2) Use of functions, especially functions of more than one variable.

(3) Sequence of execution of AID commands.

This last observation is substantiated by the study of structural variables in constructed-response exercises, and also by the examination of the student performance using the AID interpreter. Because of the difficulty students had with the hierarchy of operations and the use of functions, it is reasonable to assume that they lacked necessary algebraic experience and that the course did not provide that experience for them. It is harder to draw conclusions from the difficulty students had with the sequence of execution. If this result holds true for later lessons and for other groups of students, it will pinpoint a major weakness in the course, since an understanding of the sequence of execution is essential in programming. We hope that in later reports more conclusive evidence can be supplied on these and other unanswered questions.

REFERENCES

Davis, M. Computability and unsolvability. New York: McGraw-Hill, 1958.

Friend, J., & Atkinson, R. C. Computer-assisted instruction in programming: AID. Technical Report No. 164, Institute for Mathematical Studies in the Social Sciences, Stanford University, January 25, 1971.

Friend, J. INSTRUCT coders' manual. Technical Report No. 172, Institute for Mathematical Studies in the Social Sciences, Stanford University, May 1, 1971.

Jerman, M. Instruction in problem solving and an analysis of structural variables that contribute to problem-solving difficulty. Technical Report No. 180, Institute for Mathematical Studies in the Social Sciences, Stanford University, November 12, 1971.

Loftus, E. J. F. An analysis of the structural variables that determine problem-solving difficulty on a computer-based teletype. Technical Report No. 162, Institute for Mathematical Studies in the Social Sciences, Stanford University, December 18, 1970.

Mark, S. L., & Armerding, G. W. The JOSS primer. The RAND Corporation, Santa Monica, California, August, 1967.

Shaw, J. C. JOSS: Experience with an experimental computing service for users at remote typewriter consoles. The RAND Corporation, Santa Monica, California, May, 1965.

N74-26707

PROVIDING SOFTWARE SUPPORT FOR COMPUTER-ASSISTED INSTRUCTION[1]

J. D. Fletcher and R. W. Schulz

The burgeoning use of computer-assisted instruction (CAI) has left many system managers with the problem of appropriately modifying their software for CAI. In maintaining system support for CAI at Stanford University and in consulting with managers of commercial time-sharing systems, we have observed a set of common issues that arises whenever the question of CAI comes up. These issues are not necessarily peculiar to CAI, but they receive greater emphasis in CAI time sharing. Our intent is to document these issues and to make specific suggestions for adopting extant time-sharing systems for CAI. Some controversy may be occasioned by our suggestions. We hope that this paper will initiate such an interchange; it is time that implicit assumptions about CAI time sharing were explicated and examined. Our assumptions should benefit from such an interchange as much as anyone's.

SYSTEM DESIGN AND ACCESS

Reentry

Reentry usually receives fairly low priority in commercial time sharing. The value of reentry is often far outweighed by the programming effort required to provide for it and by the rarity of occasions when it might be of significant advantage. Reentry is considerably more important in CAI time sharing in which many users access the same curriculum program at one time. It is doubtful that many commercial time-sharing systems can be economically modified to support reentrant code. However, the desirability of reentry for CAI systems is an issue that always arises, and if reentry is economically available, it should be used.

System Access

In CAI time sharing, system access that allows a student to issue monitor commands should be carefully controlled. The simplest assurance of this control is the provision of two modes of operation: student mode and programmer mode. A user can then be assigned to one of these modes when he logs in or whenever the system establishes his job status.

Programmer mode should allow full access to all normally provided system privileges. These privileges should include execution of curriculum programs so that the system's full debugging power is available to a programmer simulating a student.

Student mode should allow no access to the system monitor. In providing for student mode, it is usually necessary to write a special logout routine that simply signs a user off rather than returns him to monitor control. This routine is then used to halt execution of student curriculum programs either normally or because of an error condition.

This recommendation may sound more autocratic than its intent. There may be perfectly good reasons for some students to run in programmer mode. It is important, however, to provide sufficient protection so that, for instance, a random fourth-grade student is not inadvertently given the opportunity to delete all files from a directory.

Number of Jobs

One issue that frequently arises is whether to assign separate jobs to separate students or to run many students under one job. Generally, it is easier to write CAI programs assuming separate jobs for separate students. Even if reentry is available, this technique requires far more system overhead than does the assignment of many students to a single job. Nevertheless, because of the frequency and depth of CAI program changes, simplicity of programming may be at a greater premium than minimizing system overhead, and we recommend, in general, that separate students be assigned separate jobs.

I/O- and Compute-bound Jobs

We consider a job to be balanced if the time it spends waiting for I/O is roughly equal to the amount of central processor time it takes. Most CAI programs are I/O-bound in that they spend much more time in waiting for I/O than in processing. Many commercial time-sharing systems, however, appear written for compute-bound jobs, and managers of CAI time sharing often lament that they must run I/O-bound jobs on a system designed for compute-bound operations. Some systems incorporate parameters that can be adjusted to provide better support for I/O-bound jobs, and we recommend that these adjustments be made wherever possible.

This recommendation should not be made without a word of caution. Early versions of CAI curriculums are almost inevitably heavily I/O-bound. However, as techniques of CAI develop and become more sophisticated, CAI curriculum programs become concomitantly larger, more complex, and, eventually, compute-bound. Adjustments for I/O-bound jobs are desirable in CAI time sharing, but they should not be irreversible.

Passing Information Between Chained Programs

Most time-sharing systems permit program chaining, but its efficiency may depend on the frequency with which the system designers expected it to occur. In CAI time sharing, chaining is likely to occur frequently, and its efficiency is at a premium.

The need to pass information efficiently between chained programs is also characteristic of CAI. This latter need is seldom met by commercial time-sharing systems which may require such cumbersome procedures as the creation of temporary disk files to pass information between chained programs. For CAI time sharing, we recommend that user programs be permitted read and write access to a small block of information that is included as part of their job status information. This block can then be used to pass information between chained programs. Whether this recommendation is worth the investment in technical effort necessary for its implementation depends on the efficiency of the system involved.

TERMINAL I/O

Data Communication

CAI time sharing must inevitably support a great amount of character transmission between student terminals and the control system. Further, an educational technique that has emphasized rapid feedback and knowledge or results as much as has CAI, requires that character transmission be as efficient as possible. This topic is far too complex to be discussed satisfactorily in a paper of this scope; but, in general, we note that efficient character transmission is a perennial topic in CAI and that as much character processing as possible should be accomplished outside of the central processor. Special hardware and special codes for character transmission are particularly desirable in CAI time sharing.

Character Timing

Character timing and time outs are always desirable in CAI time sharing. Knowing when characters are transmitted and received is important in obtaining "latency" measures, which indicate how long students take to answer problems and which are especially important for research and development in CAI. The maximum error than can be tolerated in these measures is about 250 msec;

certainly, errors of 2-3 sec are too large. Character timing, therefore, should be accomplished by the system rather than by user programs which may be descheduled for 2-3 sec under heavy usage.

Time-outs send a special "wake-up" character to curriculum programs if no response is sent by the student after a specifiable interval of time. Time-outs in CAI time sharing do not require the precision of latency measures; but they should, at least, be available in the languages used for CAI curriculum programming. Further, they should be available in both character mode and line mode. Character mode is distinguished from line mode in that every character sent to user programs acts as a "wake-up" character; in line mode only certain break characters such as (line feed), (return), or (end of transmission) cause programs to be scheduled. If time-outs are available only in line mode, a student may be interrupted in the middle of a correct response or he may have to wait an inordinately long time for his time-out to occur. If time-outs are available only in character mode, the system scheduler will occasionally be encumbered with much unnecessary overhead.

Character Editing

Character editing of terminal input buffers at the system level is a desirable feature of CAI time sharing. This feature allows a student to correct typing errors, for instance, without requiring the system to schedule his curriculum program. Ideally, a CAI curriculum program could enable line mode or character mode input as they become pedagogically appropriate. System editing of the terminal input buffer would be, then, an automatic concomitant of enabling line-mode input.

Break Characters

Break characters for terminating line-mode input should be programmable. The language used for CAI programming should allow break characters to be set and changed.

Direct Terminal Output

It should be possible to send any character to a student terminal without interference from the system. Many commercial systems automatically send a carriage return to a terminal before outputting characters from a program, and many systems send a prompt character to terminals when a program is ready to accept input. These features are desirable in CAI time sharing as default options, but they should be under the control of CAI curriculum programs.

DISK FILE STRUCTURE

Special Structures

CAI curriculum programs frequently interrogate student history and curriculum files. Therefore, efficient and truly random disk access is essential in CAI time sharing.

Further, CAI typically requires a large number of small files. For instance, a system that supports 1,000 students using three curriculums might require 6,000 small, perhaps single-record, files. Some characteristics of these files can be considered in reducing the system overhead necessary for maintaining them. They are relatively permanent, are of small maximum length, and are more likely to be read than written. Both flexibility and protection can be sacrificed to minimize system overhead and maximize speed in accessing these files. On the other hand, there remains a need for high-level, powerful disk file structure in CAI time sharing. When it is practicable, therefore, we recommend that CAI time-sharing systems support two disk file structures: a low-level structure that is fast and minimizes system overhead used for student history, curriculum, and performance data files, and a powerful, high-level structure used for maintaining other, more typical files.

Performance Data Files

CAI time-sharing systems that support curriculum research and development are typically required to record and organize large amounts of student performance data. We recommend that performance data recording be accomplished by the system. The form and content of the data records should be under control of the CAI curriculum programs, because different curriculums may require different data and because saving all performance data for every CAI program inevitably results in extensive inefficiencies in both recording and organizing the data. The best procedure is to permit curriculum programs to call a system routine that will record a fixed length block of information when and where it is appropriate. Data analysis programs can then access a common set of organizing and retrieving routines that are maintained and updated in the system.

LANGUAGES FOR CAI

Interpretive Compilers versus Code Generating Compilers

A system that supports CAI on a daily, full-operational basis usually must defend its cost by providing as many student contact hours as possible. The tendency of CAI curriculum programs to grow larger and more complex as they evolve is likely to overwhelm the capacity of systems to support sufficient number of students executing interpreter-based code.

The superiority of interpretive compilers in providing helpful diagnostics during debugging may become an important consideration if the CAI curriculum programmers are relatively unsophisticated. This consideration is particularly important if teachers are expected to produce their own CAI. Generally, an explicit administrative choice must be made between teacher-written CAI and large-scale CAI usage. A strong case can be made for interpretive compilers in the former instance but not in the latter.

Reentry (Again)

Some systems support reentry but only for assembler language programs. This is an obvious, unnecessary hardship, and we strongly recommend that reentry be available in higher-order languages used for programming CAI curriculums.

Linkages to System Library Functions

Higher-order languages on some systems permit access to an unnecessarily small subset of system functions. CAI curriculum programs often require special system functions and unusual "front-end" equipment such as light pens, audio tape controllers, and graphics displays. Therefore, linkages to system library functions should be as simple as possible. Provision for inserting assembler language statements, or even binary code, directly into the set of higher-order language statements is desirable and may permit the most facile access to system routines.

Optimization of Code

Sooner or later, CAI time-sharing systems are pushed to maintain as many users as possible for as long as possible, and it is appropriate to end these comments with a strong recommendation for as optimal and efficient a code as possible. Code optimization for string and character manipulation is especially important. CAI curriculum programs shall, for example, be able to avoid repeated concatenation of the same string and be permitted to specify upper limits on the character lengths of strings. In a choice between speed of compilation and code optimization, systems for CAI should generally opt for the latter. An ideal CAI system would provide an interpreter for debugging curriculum programs and a compiler for generating optimized code for daily operations.

FOOTNOTE

[1] This paper was presented at the Computer Sciences Conference held in Columbus, Ohio, February 20-22, 1973. Funding support was provided by NASA Research Grant NGR-05-020-547.

John D. Fletcher has served Stanford University as Research Associate for approximately nine years. He received his baccalaureate degree from the University of Arizona, and both his master's degree in Computer Science and his Ph.D. degree from Stanford University. Dr. Fletcher has authored and co-authored a large number of articles, reports, books, and other publications around the country.

Rainer W. Schulz is Chief System Designer on the ILLIAC IV project at NASA/Ames. In prior years he has served as system manager for IBM and XDS, software manager for the Berkeley Computing Corporation, and system manager at the Institute for Mathematical Studies in the Social Sciences, Stanford University. Mr. Schulz' baccalaureate degree was obtained from California State University at San Jose.

N74-26708

COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING: A CURRICULUM DESCRIPTION

by

Jamesine Friend

July 31, 1973

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING: A CURRICULUM DESCRIPTION[1]

Jamesine Friend
Stanford University

In 1967 the Institute for Mathematical Studies in the Social Sciences began the development of a computer controlled course in computer programming. This course is an introduction to computer programming for community college students who have had high school algebra, and it is equivalent to a three unit, one quarter community college course. All instruction is presented by computer, so that the course can be used where there are no qualified instructors of programming; a supplementary student manual used as a reference book is provided. In the first stage of development about one-third of the course was implemented on the PDP-1 computer and tested on a small group of volunteers, mostly Stanford University students. The course was then revised, completed, and implemented on the larger PDP-10 computer. Since then several hundred students in several schools and colleges have been enrolled for the course. In 1970 data collection routines were added to the instructional system so that the course could be used for research.

Students taking the course communicate with the PDP-10 computer at Stanford University using KSR Model 33 teletypewriters which are connected to the computer by ordinary telephone lines. A teletypewriter may be located anywhere that electricity and telephone lines are available--in an office, a classroom, or even a private home; in practice, these student terminals are usually put into small classrooms in clusters of four to eight machines. Each terminal communicates independently with the computer, and each student works independently of other students. A student may work at any hour (provided the computer is in operation at that time) and at any terminal. The communication between the computer and the student takes the form of a "conversation" in which the computer and student take turns typing on the terminal, the computer presenting instruction and problems and the student replying by typing his answers; the computer then analyzes the student's answers, tells the student whether he is right or wrong, and supplies further instruction.

A standard Model 33 KSR teletypewriter is ordinarily used although several similar models could also be employed. These machines have limitations that affect both the style and content of the material being taught. For example, the Model 33 teletypewriter is a slow output device, delivering only ten characters per second. Since a college student can read much faster than ten characters per second, large quantities of text are printed by teletypewriter only at the risk of boring the student. Partly for this reason, instruction in the course is given in small, succinct paragraphs, never more than several hundred words, and usually less than one hundred words. Another limitation of the Model 33 teletypewriter is its inability to produce graphic characters. Depending upon the subject to be taught this limitation is more or less severe; elementary geometry, for example, is essentially impossible, but social studies are less difficult. For a course in computer programming the limitation is not severe, although flow charting, which ordinarily forms a part of an introductory course in programming, is best omitted. It is possible, using the limited set of characters available, to produce fairly acceptable flow charts, but this is so time-consuming, both in the original programming of the display and in the printing itself, that the inclusion of flow charts is of questionable value.

Other than the omission of flow charts, the content of the course is quite similar to other introductory courses in programming. The course teaches the concept of stored programs, the use of variables, and introduction to input and output, the syntax of algebraic expressions, definitions of functions, conditional clauses and the syntax of logical statements, conditional and unconditional branching, core and disk storage, and an introduction to subroutines. Some of these subjects, such as disk storage and subroutines, are discussed only briefly, while others, such as the syntax of algebraic expressions and the use of conditional branches, are covered more extensively.

The language that is being taught in this course is AID (Algebraic Interpretive Dialogue), an algebraic language in the same class as the programming languages ALGOL, FORTRAN, and BASIC. AID is in some respects superior to these other languages as a first programming language. Since it is interpreted rather than compiled, students can use direct commands that will be executed immediately, giving them an opportunity for early hands-on experience; in a language that is compiled the user cannot execute a command until his program, written in the proper format, is stored, and compiled, and the run command is given. Another advantage of AID is that it is a subset of English extended by the language of elementary mathematics, making AID commands and programs easy to read. For example, these typical commands can be read and understood with no instruction:[2]

```
SET X = 5
SET Y = 1/X
TYPE Y IF X + Y < X/2
```

One disadvantage of AID is that it is less well-known and not as available as BASIC, ALGOL, FORTRAN, and a number of other languages. This means that a student who learns AID is less likely to be able to put his knowledge to immediate use. In partial compensation for this, AID is sufficiently similar to the more widely used algebraic languages that a student who knows AID well can readily learn one of these other languages with very little instruction. Also, several variants of AID bearing different names are implemented on a variety of computers. One of these, the original on which AID was modeled, is JOSS, a language designed at RAND Corporation for use by engineers and scientists. Another nearly identical language is FOCAL, which is implemented for several Digital Equipment Corporation computers.

When AID was chosen as the language to teach, the choice had rapidly narrowed down to AID and BASIC. BASIC, an algebraic language designed at Dartmouth College as a beginner's language, is widely known and used, and has several very useful advanced commands for matrix manipulation that are lacking in AID. BASIC is an excellent beginner's language, and perhaps would have been the choice for this course except for two factors: (1) AID as an interpreted language was felt to be more responsive, and (2) (a more pragmatic consideration), BASIC was not at that time implemented on the IMSSS system. Soon after development of the AID course began, a BASIC compiler became available for the PDP-10, and a BASIC course was subsequently written at Stanford. The BASIC course was patterned after the AID course and uses the same instructional system.

This instructional system was designed to give tutorial instruction under computer control and implements several teaching strategies that might be used by a human tutor. Tutorial instruction, as contrasted with the lecture method of teaching, requires an ability to tailor the sequence and content of the instruction to the individual student. In order to do this the student's desires and abilities must be taken into account. The analysis of an individual student's ability depends upon prior accumulation and analyses of his responses to exercises and problems, and the bulk of the instructional system consists of routines for analyzing such responses. Some of these routines, such as the routine for analyzing a response to a true-false exercise, are quite simple; others are relatively complex, allowing for such possibilities as "correct but incomplete" and "partially correct," and for a wide range of equivalent answers. All of the analysis routines used in the system depend upon the student's response being in an expected form. Thus, a student confronted with an arithmetic problem like 50^2 is expected to reply with a numeral like 2500 or 2500.0 or 2.5×10^3; responses like "twenty-five hundred" or "The answer is 2500" are not recognized as correct answers and the instructional program will reply "No. Type a number." Because the student's response must be phrased in a restricted form, each exercise is written so that the expected form of the response is clear to the student. Despite the limitation of restricted forms for student responses, a great variety of problem types can be used, and the correctness of responses determined with precision, permitting considerable individualization of instruction.

Individualization of instruction is also achieved in the course by varying both the amount of instruction given and the sequence of instruction. Both of these are largely under student control. Students control the amount of instruction in each exercise by a device called the "hint" option. A student may request additional information for any exercise by asking for a "hint." For very difficult problems, several hints may be provided while for simpler exercises, such as true-false exercises, no hints are provided. If a student requests a hint for one of these easier exercises, he is simply told that there are no hints available. For most exercises students are allowed an unlimited number of opportunities to respond. A student may attempt to answer, then ask for a hint, try again, ask for another hint, etc. If a student tries one problem several times and feels that he is making no headway, he may ask for the answer and continue with the lesson; in this way each student is allowed to judge for himself how much time he should profitably spend on each problem.

The sequence of instruction is also controlled by the student. There is an automatic sequence of exercises that each student takes, unless he initiates a change of sequence. This sequence may differ for different students depending upon their ability. A student may interrupt the instructional program at any time to request a different exercise. He will then proceed in the automatic sequence from the exercise he requested until he again interrupts the program. Students use this feature either to skip over lessons or portions of lessons, or to review some topic they have previously studied.

Although the students may override the automatic sequencing of exercises provided by the program, most prefer to follow the prescribed sequence more or less as given. Even if students choose to take exercises and lessons exactly as given, they will not all receive an identical set of exercises, since the program provides alternative branches based on current assessments of student ability. Such automatic branching is used in this course primarily to provide short remedial sequences of additional instruction for students whose performance indicates an inadequate grasp of the principles being presented. These remedial sequences are imbedded at various places in the lessons where appropriate and are frequently used to provide additional practice in algebraic skills that have been inadequately learned. Because of this automatic remediation, different students may receive different exercises in a given lesson.

A student who makes an incorrect response to a single exercise may not need an entire sequence of remedial exercises. He may profit from a single specific corrective message, pointing out the error in his response and allowing him another try at the same problem. This kind of specific correction is used for most exercises in the course. Messages are provided, not for all possible incorrect responses, but for those incorrect responses most likely to occur and most indicative of a student's misconception.

Besides the immediate remediation provided in the specific correction messages and in the imbedded sequences of remedial exercises, opportunity is also provided for review and practice of previously covered topics. This review is given in the form of lesson reviews and block reviews, as described in the following section.

Organization of the Course

The AID course contains a main strand of 50 lessons organized into "lesson blocks." Each lesson block contains five teaching lessons, a test, and a block review. The fiftieth, and last, lesson is a set of programming exercises with no new instruction and is not included in a lesson block. The structure of the lesson strand is illustrated in Figure 1. Lessons numbered 6, 13, 20, 27, 34, 41, and 48 are tests; and lessons numbered 7, 14, 21, 28, 35, 42, and 49 are block reviews. The teaching lessons present concepts in a tutorial style and supply numerous practice exercises; the exercises in the teaching lessons provide hints and allow the student an unlimited number of trials per exercise. The tests are intended to be optional self-tests. They provide no hints and allow the student only one chance at each exercise. The block reviews are also optional and recommended only for those students who do poorly in the preceding tests. These reviews are essentially no more than quick reference sources, providing brief summaries of the topics covered in the lesson block and supplying the student with lesson and exercise numbers for the topics he wants to review. The block reviews for the first two blocks are somewhat more complicated than later block reviews in that appropriate review exercises are automatically selected for the student who wants them, whereas in later blocks students are simply given the reference and expected to use the available student controls to access those exercises.

Each teaching lesson covers a set of related concepts and the length of the lessons varies depending upon the complexity of the ideas covered.

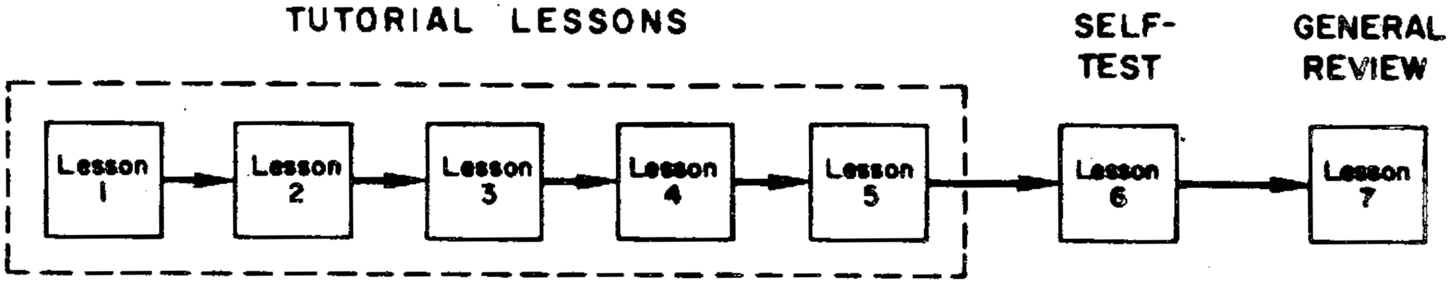


Figure 1. Structure of main lesson strand.

The lessons average about 30 exercises in length with a range from 14 to 63 exercises. The lessons take about an hour apiece although there is considerable variance because of the differences in length and difficulty. Students need not complete a lesson each day; they may work on the same lesson for several days, or they may take several lessons in one day. Ordinarily, the students take the lessons in numeric order but they are not required to do so; if they wish to skip around in the course, they may.

In the first two lesson blocks each teaching lesson is followed by an optional lesson review. These lessons are for students who are having difficulty getting started in the course and cover the same concepts that were presented in the lessons. The instruction in the lesson reviews is given more slowly and carefully, and additional practice problems are provided. A student need not review an entire lesson since the lesson reviews are structured to allow the student to review only those topics he chooses. At the beginning of each review the topics covered in the lesson are listed with essential topics starred and the student chooses which he wants to review and in which order. Since students are allowed to skip about in the course as they please, a student may skip a lesson review and return to it at a later time.

After a student finishes a lesson, he is asked if he wants a summary. Each summary is printed on an 8-1/2" by 11" form that can be torn off and saved by the student as a permanent record. For many lessons there are also a few optional "extra-credit" problems. These problems are substantially more difficult than the exercises given in the lessons and are intended for more capable students. Wherever the AID course is used as a graded course, we recommend that correct solutions to these problems entitle the student to extra credit. Not every lesson has such extra-credit problems since they are not always appropriate to the subject matter.

The relationship and sequence of lessons, lesson reviews, summaries, and extra-credit problems are illustrated in Figure 2. The number of exercises in each lesson and review, and the number of extra-credit problems for each lesson are shown in Table 1.

The teaching lessons comprise the first five lessons in each lesson block and form the substantive part of the course. As can be seen in Table 1, there are a total of 1943 exercises in the course; of these 1180 are in the teaching lessons. All lessons except the teaching lessons are optional and may be bypassed by a student who wishes to complete the course as quickly as possible. A student who could work at the rate of one problem every two minutes, a reasonable expectation for a good student, could complete the course in 40 hours. Several of the teaching lessons could also be omitted, namely, those on recursive functions, trigonometric functions, and exponential functions. If a student skipped these lessons, he might well finish in 30 hours or less. A slower student, on the other hand, might take all of the optional lessons and problems, and put over 100 hours into the course. In colleges where the AID course is given for credit, it is the usual practice to assign a fixed number of hours of computer time to students regardless of their ability. An assignment of five hours per week for 10 weeks is enough to allow the better students to complete the course while slower students finish only the first 20 or so lessons. The course content is organized

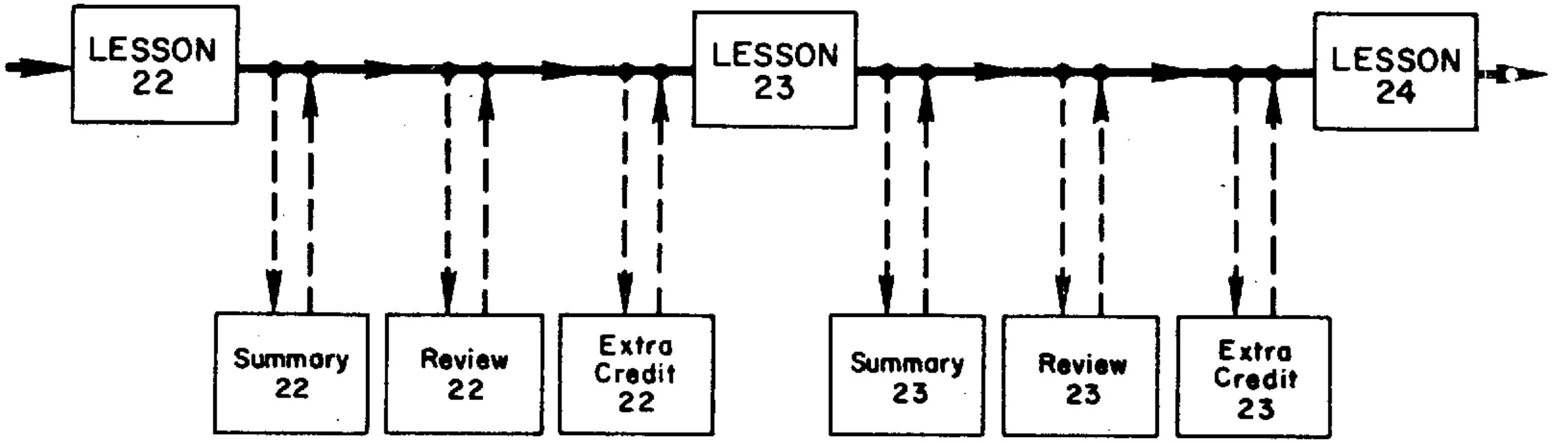


Figure 2. Relationship and sequence of tutorial lessons, summaries, reviews and extra-credit problems.

Table 1

Number of Exercises in Lessons, Lesson Reviews, Tests, Block Reviews, and Number of Extra-credit Problems

Lesson block	Lesson number	Number of exercises in lesson	Number of exercises in review	Number of extra-credit problems
1	1	18	18	0
	2	28	33	0
	3	49	44	0
	4	61	87	0
	5	63	37	0
	6	43	*	*
	7	13	*	*
Subtotals		219	219	0
2	8	62	23	4
	9	31	28	9
	10	33	40	0
	11	30	25	3
	12	14	14	2
	13	33	*	*
	14	12	*	*
Subtotals		170	130	18

Table 1 (cont'd)

Lesson block	Lesson number	Number of exercises in lesson	Number of exercises in review	Number of extra-credit problems
3	15	62	55	2
	16	27	0	0
	17	27	0	0
	18	35	0	0
	19	32	0	0
	20	27	*	*
	21	9	*	*
Subtotals		183	55	2
4	22	29	0	1
	23	36	0	1
	24	35	0	4
	25	20	0	1
	26	19	0	3
	27	30	*	*
	28	19	*	*
Subtotals		139	0	10
5	29	33	0	2
	30	17	0	2
	31	24	0	0
	32	50	0	2
	33	23	0	2
	34	27	*	*
	35	13	*	*
Subtotals		147	0	8

Table 1 (cont'd)

Lesson block	Lesson number	Number of exercises in lesson	Number of exercises in review	Number of extra-credit problems
6	36	58	0	3
	37	43	0	0
	38	34	0	3
	39	37	0	2
	40	31	0	0
	41	31	*	*
	42	8	*	*
Subtotals		203	0	8
7	43	24	0	0
	44	28	0	0
	45	23	0	2
	46	23	0	1
	47	21	0	1
	48	32	*	*
	49	12	*	*
Subtotals		119	0	4
Totals		1489	404	50

Number of exercises in teaching lessons (first five lessons in each lesson block): 1180

Number of exercises in tests (sixth lesson in each lesson block): 223

Number of exercises in block reviews (seventh lesson in each lesson block): 86

Total number of exercises: 1943

*There are no review or extra-credit problems for tests or block reviews (the last two lessons in each lesson block).

so that the most essential programming concepts are presented in the first half of the course, leaving more advanced concepts such as the conditional definition of functions and the use of matrices to the later lessons. In this way the slower students are given a good introduction to the basic principles of computer science. A brief outline of the lessons is given in Table 2; notice that the first 25 lessons cover algebraic expressions, some standard functions, stored programs, conditional clauses, branching, and loops.

Types of Exercises Used in the AID Course

Although the aim of the AID course is to teach students how to program, and in particular how to program using the AID language, not all of the exercises in the course are programming problems. Most exercises are simpler, ranging in difficulty from trivial true-false exercises to exercises that require the student to construct a complete syntactically and semantically correct AID command that could be used as part of an AID program. The exercises used in the AID course fall into five major groups:

Multiple-choice, and similar, exercises

Short constructed-response exercises

Programming problems

Questions that ask the student to express an opinion or preference

Ungraded exercises

Each of these problem classes is described below, and the first two classes are further subdivided. The exact number of exercises in each of these classes is shown in Table 3 for each teaching lesson and test.

Table 2

Outline of Course

Lesson 1	Using the Instructional Program
Lesson 2	Using AID for Arithmetic
Lesson 3	Order of Arithmetic Operations
Lesson 4	Exponents and Scientific Notation
Lesson 5	The SET and DELETE Commands
Lesson 6	Test of Lessons 1 to 5
Lesson 7	Block Review
Lesson 8	The LET Command
Lesson 9	Some Standard AID Functions
Lesson 10	Indirect Steps, the DO Command, the FOR Clause
Lesson 11	Parts
Lesson 12	The DEMAND Command
Lesson 13	Test of Lessons 8 to 12
Lesson 14	Block Review
Lesson 15	Relations and the Use of the IF Clause
Lesson 16	The TO Command
Lesson 17	Debugging Techniques
Lesson 18	The Indirect Use of the DO Command
Lesson 19	Debugging, Permanent Storage
Lesson 20	Test of Lessons 15 to 19
Lesson 21	Block Review
Lesson 22	The FORM Statement
Lesson 23	Loops
Lesson 24	Loops with Variables in the Exit Condition
Lesson 25	Loops and the FOR Clause
Lesson 26	Loops with a DEMAND Command
Lesson 27	Test of Lessons 22 to 26
Lesson 28	Block Review
Lesson 29	Absolute Value

207

Table 2 (cont'd)

Lesson 30	SIN and COS
Lesson 31	Exponential and Logarithmic Functions
Lesson 32	Lists
Lesson 33	Using Loops with Lists of Numbers
Lesson 34	Test of Lessons 29 to 33
Lesson 35	Block Review
Lesson 36	Nested Loops and Decrementing Counters
Lesson 37	SUM, PROD, MAX and MIN
Lesson 38	Arrays
Lesson 39	More about Arrays and Lists
Lesson 40	Conditional Definition of Functions
Lesson 41	Test of Lessons 36 to 40
Lesson 42	Block Review
Lesson 43	Recursive Functions
Lesson 44	AND, OR and NOT; Truth Tables
Lesson 45	TV(X) and the FIRST Function
Lesson 46	LET and Boolean Expressions; Debugging Tools
Lesson 47	More Standard AID Functions
Lesson 48	Test of Lessons 43 to 47
Lesson 49	Block Review
Lesson 50	Programming Problems

Class 1: Multiple-choice, and similar, exercises. These exercises pose a set of alternate possibilities that are listed or clearly implied by the text of the exercise. There are four distinct subclasses within this class.

(a) Multiple-choice exercises. This subclass includes only those exercises that are in the traditional multiple-choice format, with the choices explicitly listed and each choice labeled with a letter identifier. A number of multiple-choice exercises in the course have more than one correct choice, and the student is expected to find all of the correct choices. Multiple-choice exercises can be further subdivided according to how many correct choices there are. Also, in some exercises none of the choices listed is correct, and the list of choices includes the entry

N. NONE OF THE ABOVE.

Exercises for which there is no correct true choice could also be put into a separate subclass. The following, finer, division of the class of multiple-choice exercises will not be found in Table 3, which shows only the broader classification.

(i) Multiple-choice exercises with one correct answer (other than exercises in which the correct answer is "None of the above."). Of the 174 multiple-choice exercises in the teaching lessons and tests, 107 are in this class.

(ii) Multiple-choice exercises with two correct choices. There are 39 exercises of this kind.

Table 3

Number of Problems in Each Class of Problems, by Lesson

	Problem Class											
	1a	1b	1c	1d	2a	2b	2c	2d	3	4	5	
Lesson number	Multiple choice	Yes-no	True-false	Other implied choice	Predicted result of AID command	AID command	Reported result of AID command	Other constructed responses	Programming problems	Opinion questions	Ungraded exercises	Total
1	5	2						3		5	3	18
2	11			1	4	2	3		3	4		28
3	11				19	7	3	1	3	5		49
4	7			1	25	1	3	15	3	6		61
5	6				5	9	13	17	9	4		63
6	6		7		12	5	2	6	2	3		43
8		1			32	8	4	8	4	5		62
9	1				17		6		3	4		31
10	4				1	2	4	13	5	4		33
11	1	1		1		6	2	5	6	8		30
12						1	3	2	5	3		14
13	3				13	8		4	2	3		33
15	5		24	6	6		1	6	6	8		62
16	5	1				1		10	2	8		27
17	1							18		4	4	27
18	6							22	1	6		35
19	1	7		3		7		2	4	8		32
20	6	1	1		1	6		10		2		27
22	12	1				1	1	6	4	4		29
23	1					1		28	2	4		36
24	4			1				18	1	7	4	35
25	1							13	1	4	1	20
26	1				1	1	2	6	2	3	3	19
27	7					2		20		1		30

Table 3 (cont'd)

Problem Class

	1a	1b	1c	1d	2a	2b	2c	2d	3	4	5	
Lesson number	Multiple choice	Yes-no	True-false	Other implied choice	Predicted result of AID command	AID command	Reported result of AID command	Other constructed responses	Programming problems	Opinion questions	Ungraded exercises	Total
29					6		3	15	2	5	2	33
30					7	1	2	1	2	4		17
31	5			6	4		3	1	3	2		24
32	4				9	1	2	25	3	3	3	50
33					2	1	2	6	4	5	3	23
34	4		1		8	2		11		1		27
36	3					2	6	31	5	7	4	58
37	1				14	3	4	15	4	2		43
38					11		1	14	2	3	3	34
39	5					1	3	15	6	4	3	37
40	10							13	2	2	4	31
41	11			1	2	6		10		1		31
43	2						9	5	4	3	1	24
44	4		8		1			5	3	2	5	28
45	2				2		3	6	4	3	3	23
46	4				3	1	3	2	4	3	3	23
47	1							15	1	3	1	21
48	13		7					10		2		32
Total, Teaching Lessons:												
	124	13	32	19	169	57	86	362	113	155	50	1180
Total, Tests:												
	50	1	16	1	36	29	2	71	4	13	0	223
Totals:	174	14	48	20	205	86	88	433	117	168	50	1403

(iii) Multiple-choice exercises with three or more correct choices. There are 21 exercises in this class.

(iv) Multiple-choice exercises to which the correct response is "None of the above." There are seven exercises like this (although there are many exercises that list "None of the above" as a possibility).

(b) Yes-no questions. The exercises in this class are those in which the two possibilities, yes and no, are implied by the grammatical form of the question rather than being explicitly listed as are the choices for multiple-choice exercises. All of the exercises in this class deal with matters of fact, rather than opinion. A large number of yes-no questions are also found in the class of exercises that ask for students' opinions (Class 4, described below), but those exercises are also not included here.

(c) True-false exercises.

(d) Other "implied-choice" exercises. There are a number of exercises in the course that appear to be constructed-response exercises in that the text does not list the possible answers from which to choose. These exercises are revealed by closer inspection to be more related to multiple-choice exercises than to constructed-response exercises since a limited number of choices are clearly implied in the statement of the problem. As an example, the following question gives two alternatives, one of which is the correct answer:

IF YOU USED THIS COMMAND

TYPE 1/100

WOULD AID GIVE THE ANSWER IN DECIMAL FORM OR IN SCIENTIFIC NOTATION?

Class 2: Short-constructed-response exercises. These exercises require short-constructed responses that will be checked for correctness by the instructional program. Class 3 exercises and Class 5 exercises, described below, could also be considered constructed-response exercises and are distinguished from exercises in this class in that they are not checked for correctness by the instructional program, for reasons given below. The short-constructed-response exercises, like the multiple-choice exercises, can be further divided into several subclasses, which are described below and are tallied in Table 3.

(a) Exercises that ask the student to predict the result of using a given set of AID commands. In these exercises the students are shown an AID command or sequence of commands and are asked to predict what result would be given if such commands were used.

(b) Exercises that require the student to construct an AID command. In these exercises the student's response is analyzed by the instructional program, as are all exercises in Class 2; students are also expected to construct AID commands as they work the programming problems in Class 3, but those commands are not analyzed by the instructional program, and are not included in this class. On occasion students are asked to construct a part of an AID command or to complete a given partial command; those exercises are in Class 2d, below.

(c) Exercises that ask the student to report results obtained from student-constructed AID programs. These exercises always follow an exercise from Class 3. They are used by the instructional program to judge the correctness of the program written by the students for the preceding Class 3 exercise. A more complete explanation of the sequence of problems will be given below in the description of the Class 3 exercises.

(d) Other constructed-response exercises. This class incorporates the miscellaneous constructed-response exercises. Exercises similar to those in Class 2b but requiring the student to construct only a part of an AID command, rather than an entire command, are included here. Also included are exercises similar to those in Class 2c but requiring non-numeric responses. The class is large and heterogenous; the only common characteristics of the exercises in this class are that they all require constructed responses less than one line long that are checked for correctness by the instructional program.

Class 3: Programming problems. Characterization of the exercises in this class as "programming problems" is imprecise; many of these exercises require the student to construct only a single AID command that is not part of a program and may simply be copied from the text given by the instructional program. However, the class is well-defined in that all its exercises require the student to use the computer as a programmer, not as a student. To clarify this issue a few remarks about the instructional system are called for.

The various types of commands that can be written in the AID language and used in AID programs are introduced to the student one at a time, using a sequence of instruction like the one described below.

First, an example of an AID command or program is shown and its use explained.

Second, the student is shown several examples and asked to predict the result of using such a command or program.

Third, the student is required to construct part or all of a similar command or program.

Fourth, one or more programming problems illustrating the use of the new principle are given.

In the first three steps of the sequence described above the instruction is given to the students by means of an instructional program named INST; in the fourth step the students use the AID interpreter which interprets and executes AID commands. After using the AID interpreter to solve the problem, students return to the instructional program for further instruction. Programming problems are posed by the instructional program, but once a student starts using the AID interpreter he is given no further instruction until he branches back to INST. This branching is completely under student control; the details of accessing both programs are taught in the course. Thus, while the student is using AID he is completely on his own, communicating with the computer by means of the AID interpreter, just as a working programmer would do. These two programs, INST and AID, together control all of the student's interaction with the computer as he takes the AID course, and together form the interactive part of the instructional system. In essence, the difference between these two programs is that INST talks <u>about</u> the language AID, while the AID interpreter <u>uses</u> the language AID. The AID interpreter is a commercial program that was written for the use of programmers, not students, and contains no routines for comparing a student's program to a correct solution for the same problem. For instructional purposes, however, it is desirable to know if a student solved the given problem, and this is accomplished in the course by the instructional program which keeps track of which programming problem the student is working on and asks him about the results obtained by his program after he returns to the instructional program. The problems that the student is working on while he is using the AID interpreter are exactly the problems in Class 3, and the exercises that ask the students the results of his work are largely in Class 2c, although a few are in Class 2d.

As mentioned before, not all of the exercises in Class 3 should be labeled programming problems since some require no more than a single direct command. Thus, the difficulty of exercises in this class varies from the easiest to hardest to be found in the course. Some exercises, in fact, are no more than trivial copying tasks (with the added complication of starting and stopping the AID interpreter), while others are programming problems of a complexity that might challenge experienced programmers. The exercises in Class 3 could be further subdivided into at least three subclasses on the basis of difficulty. The easiest subclass, which asks the student to start the AID interpreter, copy a given set of commands, and observe the result, contains 37 of the 117 programming problems in the teaching lessons. Another seven programming problems are almost as simple, requiring only minor modifications of programs given as examples in the lessons. There are 73 problems that are difficult enough to require some original thinking or problem-solving skills on the part of the student; these 73 problems themselves vary considerably in difficulty. Note that we are talking here about the 117 programming problems in the teaching lessons and tests. There are also numerous programming problems in the review lessons, and all of the extra-credit problems are programming problems.

Class 4: Opinion questions. These exercises ask the students to express an opinion or preference. Typical exercises of this class are "Would you like to review any of the topics from Lesson 8?", "Do you want the summary for this lesson?", and "Did your program give the results you expected?" The response to these exercises are analyzed, and acted upon, by the instructional program, but are not classified as right or wrong. Most of these exercises are in the form of yes-no questions.

Class 5: Ungraded exercises. This class of exercises is very small and contains only those exercises that require responses that cannot for one reason or another be graded by the instructional program. These exercises are separate from the programming problems in Class 3 which cannot be graded for an entirely different reason. Some of the ungraded exercises are ungraded, not because of any theoretical difficulty, but simply because the requisite mechanism does not exist in the instructional program at this time. Others are ungraded because they ask for freely constructed responses that cannot be analyzed because of the complexity of the English language; in these cases, students are given a sample correct answer and asked to judge for themselves whether or not they responded correctly (no record of the student response is kept by the system).

The above scheme for classifying the exercises in the course is necessarily a hybrid scheme, using both response format and exercise content as bases for classification. This scheme was chosen because it gives a good picture of the curriculum. Other schemes could also have been used. One possible scheme of classification would be to divide the exercises strictly according to the form of the expected response. This classification would yield classes such as

multiple-choice exercises
yes-no questions
true-false exercises
constructed-response exercises: numeric
constructed-response exercises: single letter or character
constructed-response exercises: word or phrase
constructed-response exercises: AID command
constructed-response exercises: AID program

This classification has certain virtues, one of which is that there is no ambiguity, but distinctions such as the one between "opinion" questions and other yes-no questions are lost.

Another method of classification that would yield a different profile of the course would depend strictly upon content and not upon the form of the response. Since the division of the course into lessons is a division based upon content, a closer look at the content of each lesson is warranted.

Description of Content of Lessons

A complete table of contents for the teaching lessons is given in the appendix which lists all the topics discussed in each lesson together with a list of the exercises on that topic (and the number of exercises for each topic). Below is a more informal discussion of the teaching lessons, tests, and block reviews, with numerous examples.

Lesson 1: Using the instructional program. Lesson 1 is a set of 18 exercises explaining how to use the instructional program. The mechanics of typing and erasing responses are explained, and instructions are given for starting and stopping the program. The student is also taught how to get additional instruction (hints), how to get the answer for any exercise from the program, and how to control the sequence of instruction.

The style of instruction in Lesson 1, as in succeeding lessons, is informal.

Lesson 1, Exercise 3:

> IF MULTIPLE CHOICE PROBLEMS HAVE MORE THAN ONE CORRECT ANSWER YOU CAN LIST THE CORRECT CHOICES IN ANY ORDER. SUPPOSE B, C, AND D ARE THE CORRECT CHOICES FOR A PROBLEM. WHICH OF THESE WOULD BE CORRECT WAYS TO ANSWER?
>
> A. D, B, C, A
>
> B. B, D, C
>
> C. B, C, D
>
> D. D, B, C

Lesson 1, Exercise 14:

> FROM LESSON 1, YOU SHOULD HAVE LEARNED HOW TO SIGN ON AND OFF, HOW TO START AND STOP THE TEACHING PROGRAM, HOW TO GET A HINT, AND HOW TO USE CTRL-G. DO YOU WANT TO REVIEW ANY OF THESE TOPICS?

In Lesson 1, five of the 18 exercises are multiple choice, similar in form to Exercise 3 shown above. This proportion of multiple-choice exercises is fairly typical of the course.

Exercise 14 illustrates an instructional strategy that is used in the lessons in the first two lesson blocks. At the end of each lesson, its content is briefly summarized and the student is asked if he wants to take the lesson review. In this way the student is forced to judge whether he is competent to proceed with the course or whether he needs additional instruction and practice.

A number of exercises are designed more to elicit opinion than information, and have no "correct" answer. Exercise 14 above exemplifies these; other examples are questions like "Do you want to go on to Lesson 2 now?" and "Do you want to practice signing on and off?" In Lesson 1 there are five exercises of this type (Class 4).

The exact number of each type of exercise in Lesson 1 (and other teaching lessons and tests) is shown in Table 3.

Lesson 2: Using AID for arithmetic. In Lesson 2 the student gets his first experience with the AID interpreter. The 28 exercises in this lesson teach the student how to start and stop the AID interpreter and how to use the AID interpreter for doing simple arithmetic by giving direct "TYPE" commands. The AID symbols for the four simple arithmetic operations (+, -, *, and/) are taught, and the use of optional parentheses in arithmetic expressions is introduced. By the end of the lesson the student is able to start the AID interpreter and give simple, direct commands like

```
TYPE 5/25
TYPE 3.25 + 17.4 + 3.12
TYPE 15*17 + 25*19
```

One of the most persistent errors made by students learning an algebraic programming language is the omission of the multiplication operator in certain kinds of algebraic expressions. The root of this difficulty is the convention used in ordinary algebra of implying multiplication by juxtaposition. For example, we ordinarily write

$$24x + 56(y - z)$$

without explicit multiplication operators. AID, like other algebraic programming languages, requires the use of operation symbols:

```
24*X + 56*( Y - Z)
```

In Lessons 2 through 5 there are a number of exercises aimed specifically at preventing the error of omitted operation symbols.

Lesson 2, Exercise 11:

```
WHICH ARE VALID AID COMMANDS?

    A.  TYPE (17.01)/32.765
    B.  TYPE 1/2 + .1785 - (12/16)
    C.  TYPE 2(10) + 3(10) + 4(10)
    D.  TYPE 1/2 + (7* 3/2)
    N.  NONE OF THE ABOVE
```

2221

Lesson 2, Exercise 19:

USE AID TO DO THESE PROBLEMS:

1. FIND THE AREA OF A RECTANGLE WITH WIDTH 1.72375 AND LENGTH 12.001325.
2. SUPPOSE A SQUARE OF WIDTH .63725 IS CUT FROM THE ABOVE RECTANGLE. FIND THE AREA OF THE SQUARE.
3. FIND THE AREA OF THE REMAINING PART OF THE RECTANGLE.

Of the 28 exercises in Lesson 2, 12 are constructed-response exercises. These exercises vary in difficulty with the most difficult being Exercise 19, shown above. None of these exercises approaches in difficulty the programming problems given later in the course.

Lesson 3: Order of arithmetic operations. The arithmetic used in Lesson 2 was relatively simple, but in Lesson 3 the complexity increases with the addition of the concept of hierarchy of operations and the use of parentheses. Because of the necessarily linear nature of computer input, an algebraic formula cannot be written on more than one line. The following expression, for example, is typically written on two lines.

$$\frac{x + y}{x - y}$$

In AID this expression would be transformed into

$$(X + Y)/(X - Y)$$

The horizontal bar is replaced with a slash and one of the functions of the bar, that of implied grouping, is lost, so that parentheses must be added. The latter expression is not only more difficult to read, but also more difficult to construct correctly since it requires the student to make a conscious decision about the desired order of evaluation. A large part of Lesson 3 is devoted to teaching the student how to force the order of evaluation by the use of parentheses, and how to determine the order of evaluation if parentheses are not used, by using an explicit set of rules for the hierarchy of the four basic arithmetic operations.

Lesson 3, Exercise 10:

```
LOOK AT THESE THREE COMMANDS. AID WILL GIVE THE SAME
ANSWER TO TWO OF THEM. WHICH TWO?

     TYPE 3 + (2*4)

     TYPE (3+2) * 4

     TYPE 3 + 2 * 4

START AID AND TRY THE THREE COMMANDS.
```

Lesson 3, Exercise 22:

```
     TYPE 100/10/10/2

COULD BE WRITTEN AS

     A.  TYPE (100/10)/(10/2)

     B.  TYPE (100/(10/10))/2

     C.  TYPE (100/(10/10/2))

     D.  NONE OF THE ABOVE
```

Since a large part of Lesson 3 reviews algebraic notions that may be better understood by some students than by others, several remedial sequences are imbedded in the lessons. Since these remedial sequences are bypassed by students who are responding correctly, a good student can complete Lesson 3 in 27 exercises whereas a student who performs poorly may receive up to 49 exercises.

<u>Lesson 4: Exponents and scientific notation</u>. Lesson 4 is much longer than average (61 exercises), and extends the work on arithmetic

expressions to include expressions with exponentiation. First, the concept of exponentiation is reviewed, and the AID symbol (↑) is introduced. The rules for the hierarchy of operations are extended to include exponentiation, and the AID form of scientific notation is introduced. Negative exponents, fractional exponents, and the zero exponent are also covered. Lesson 4, like Lesson 3, is largely review of algebraic principles that may have been forgotten. The exercises also provide practice in reading and constructing expressions in the linear form required by the AID interpreter.

As noted above the horizontal division bar so frequently used in algebra has two functions, that of signifying division and that of implying grouping. The usual notation for exponentiation also has two functions: signifying exponentiation and implying grouping. As an example, the following expression raises number 5 to the power 1/2

$$5^{1/2}$$

However, when this expression is translated into an AID expression with the symbol ↑ used to denote exponentiation, the grouping function is lost and parentheses must be added:

$$5\uparrow(1/2)$$

Since students are accustomed to this grouping function of ordinary notation, they may erroneously translate the expression above into

$$5\uparrow 1/2$$

which is equivalent to

$$5^{1}/2$$

Several exercises in Lesson 4 are designed to prevent errors of this kind.

Lesson 4, Exercise 2:

WHAT WOULD AID ANSWER TO THIS COMMAND?

TYPE 2↑3

Lesson 4, Exercise 12:

USE AID TO EVALUATE EACH OF THE FOLLOWING.

1. 4 SQUARED TIMES 3.1416
2. THE SUM OF 4 CUBED AND 6
3. THE SUM OF THE SQUARES OF 1, 2, 3, 4, 5, 6, 7, AND 8

In Lesson 4, 47 of the 61 exercises require constructed responses. Most of these are numeric results of arithmetic calculations to answer questions like that in Exercise 2. Three exercises, with several parts, require the student to use the AID interpreter, and he is encouraged to use AID throughout the lesson whenever he wishes.

<u>Lesson 5: The SET and DELETE commands</u>. After the sizeable dose of arithmetic given in Lessons 3 and 4, Lesson 5 returns to the mainstream of instruction with the introduction of two new AID commands: the SET command and the DELETE command. SET is used in AID to assign numeric values to variables, and DELETE is used to delete a previous assignment or definition. In AID, variables are single letters, and the forms of the SET and DELETE commands are straightforward and easily learned (SET X = 5 + 2, DELETE X). A few word problems are given to illustrate the use of the new commands.

Lesson 5 also introduces the multiple-argument form of the TYPE command, in which several TYPE commands can be combined by separating the arguments with commas (TYPE X, Y, X + Y).

Lesson 5, Exercise 3:

```
WHAT WILL AID ANSWER AFTER THESE COMMANDS?

     SET B = 1.5

     TYPE 3*B
```

Lesson 5, Exercise 11:

```
WHAT COMMAND WILL CAUSE AID TO SET M EQUAL TO S PLUS 9?
```

Lesson 6: Test of Lessons 1 to 5. Lesson 5 is the last teaching lesson in the first lesson block, and is followed by a test in Lesson 6. Like other tests, Lesson 6 supplies no hints, and students are allowed only one try on each exercise. However, correct answers are programmed for each exercise, and the student may request these at any time. Whenever a student misses an exercise, he is given a review reference and advised to review that topic before proceeding with the course.

The exercises in Lesson 6 are classified by type in Table 3. Of the 43 exercises in Lesson 6, three ask for students' opinions and two are programming problems that require the use of AID and are not directly checked by the instructional program. The remaining 38 exercises are test exercises that are checked by the instructional program; these 38 exercises are classified in Table 4 according to which of the teaching lessons in the lesson block are being tested. Many of the exercises test more than one lesson since the material being taught builds progressively from one lesson to the next; such exercises are listed only once in Table 4 however.

Lesson 7: Block review (general review of Lessons 1 to 5). At the beginning of Lesson 7 the student is informed that the block review is optional, but recommended for students who missed more than five problems

Table 4

List of Exercises that Test Each Teaching Lesson

Number of teaching lesson	Test Exercises	
	Test number	Exercise numbers
1	6	2, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15, 16
2	6	8, 11, 17, 18, 19, 20, 38.1, 39.1
3	6	21, 22, 23
4	6	24, 25, 26, 27
5	6	28, 29, 30, 31, 32, 33, 34, 35, 36, 37
8	13	1.1, 2, 3, 4, 5, 6, 7, 8
9	13	9, 10, 11, 12, 13, 14, 16
10	13	17, 18, 19, 20, 21, 22, 23, 24
11	13	25, 26, 27
12	13	28, 29.1
15	20	1.1, 2, 3, 4, 5, 6
16	20	7, 16
17	20	8, 9, 10
18	20	11, 12, 13, 14, 15
19	20	17, 18, 19, 20, 21, 22, 23, 24, 25
22	27	1, 2, 2.1, 2.2, 2.3, 3, 4
23	27	5, 5.1, 5.2, 5.3, 5.4, 10
24	27	6, 6.1, 6.2, 6.3, 7, 7.1, 8
25	27	5.5, 9, 11, 11.1, 12, 13, 14
26	27	15, 15.1
27	34	1, 2, 3, 4, 5, 6, 7
30	34	8, 9, 10, 11

Table 4 (cont'd)

Number of teaching lesson	Test Exercises	
	Test number	Exercise numbers
31	34	12, 13, 14, 15, 15.1, 16, 17
32	34	18, 18.1, 18.2, 19, 20, 21, 22, 23
33	34	None
36	41	1, 2, 3, 4, 5, 6
37	41	7, 8, 9, 10
38	41	11, 12, 13, 14, 15, 16
39	41	17, 18, 19, 20, 21, 22, 23
40	41	24, 25, 26, 27, 28, 29, 30
43	48	1, 2, 3, 4, 5
44	48	6, 7, 8, 9, 10, 11, 12
45	48	13, 14, 15, 16, 17, 18
46	48	19, 20, 21, 22, 23, 24, 25, 26
47	48	27, 28, 29, 30

in the preceding test (Lesson 6). In order to allow students to review only selected portions of the lessons in the lesson block, the branching structure used in Lesson 7 is more complex than that used in the teaching lessons and tests. The individual lesson reviews for Lessons 1 to 5 are called as subroutines by Lesson 7, and students may select not only the lesson they want to review but a particular part of the lesson. The following example from Lesson 7, Exercise 5 illustrates how this selection procedure operates.

```
LESSON 4 WAS ABOUT EXPONENTS AND SCIENTIFIC NOTATION.
FRACTIONAL EXPONENTS AND NEGATIVE EXPONENTS WERE DISCUSSED,
AND ALSO THE USE OF 0 AND 1 AS EXPONENTS. THE ORDER OF
ARITHMETIC OPERATIONS + - * / AND ↑ WAS COVERED.
DO YOU WANT TO REVIEW ANY OF THESE THINGS?
```

If a student answers "yes" to the above question, he is sent to the review lesson for Lesson 4, where he is allowed to review any of the topics in Lesson 4 in any order he wants.

This first block review also reminds students that they can control the sequence of instruction by using the CTRL-G key and that they can use the student manual as a reference book for the course.

Lesson 8: The LET command. Lesson 8, the first lesson of the second lesson block, introduces the LET command, used in AID to define functions. The syntax of the LET command for functions of one, two, and three variables is taught, as is the syntax of function calls. The difference between LET and SET commands is explained and the use of the DELETE command for deleting function definitions is described. A substantial part of Lesson 8 is on substitution of arithmetic expressions for variables in function calls and other arithmetic expressions.

Lesson 8, Exercise 14:

```
WHAT WILL AID ANSWER?

     LET Q(A, B, C) = C*(A + B)/2

     TYPE Q(3, 5, 7)
```

Lesson 8, Exercise 28:

```
USE AID TO DO THIS PROBLEM. DEFINE A FUNCTION TO CONVERT
DEGREES FAHRENHEIT TO DEGREES CENTIGRADE. THEN CONVERT
THESE TEMPERATURES TO CENTIGRADE:

     0, 10, 32, 72, 212
```

Lesson 9: Some standard AID functions. In Lesson 9, the student is introduced to four of the standard AID functions. These are:

SQRT(X) - the square root function;

IP(X) - the "integer part" function;

FP(X) - the "fraction part" function;

SGN(X) - the sign function.

These functions, together with functions defined by the student, are used in several programming problems.

Lesson 9, Exercise 14:

```
YOU CAN USE THE AID FUNCTION FP(X) TO FIND OUT IF ONE
NUMBER CAN BE DIVIDED BY ANOTHER WITHOUT A REMAINDER...
IS 2976 EVENLY DIVISIBLE BY 3?
```

Lesson 9, Exercise 16:

```
THE SIGN FUNCTION

     SGN(X)
```

```
GIVES 1 IF X IS A POSITIVE NUMBER

AND 0 IF X IS 0

AND -1 IF X IS A NEGATIVE NUMBER

WHAT WILL AID ANSWER?

     TYPE SGN(25)
```

Lesson 10: Indirect steps, the DO command, the FOR clause. In Lesson 10 the concept of a stored program is introduced. Up to this point, students have been using AID as a desk calculator, doing all exercises with direct commands, i.e., commands that are executed immediately. In this lesson students are taught that TYPE commands can be stored for later execution by prefacing the command with a "step number," as in the following examples:

```
2.1 TYPE F(16)

4.7 TYPE X↑2, X↑3
```

They are also taught how to execute these stored commands by using a DO command.

Two variants of the FOR clause are used to modify DO commands. In the first variant, values for the iteration variable are given by a simple listing:

```
DO STEP 17.3 FOR Y = 1, 2, 7, 14.3
```

In the second variant of the FOR clause the values for the variable are given in a range specification, which gives an initial value for the variable, a step size, and a final value:

```
DO STEP 5.6 FOR X = 3(2)9
```

This command specifies that X will assume the value 3, then be incremented by 2 after each iteration of step 5.6 until X $\geq$ 9. This is equivalent to the FORTRAN form:

```
      DO 56 X = 3, 9, 2
   56 < statement >
```

and the ALGOL form:

```
      FOR X ← 4 STEP 2 UNTIL 9 DO < statement >
```

Lesson 10, Exercise 6:

```
      START AID AND GIVE THESE COMMANDS:

           47.3 TYPE X, X↑2, X↑3, X↑4, X↑5

           SET X = 3

           DO STEP 47.3

           SET X = 4

           DO STEP 47.3

           SET X = 5

           DO STEP 47.3

      WHAT IS THE VALUE OF X↑5 IF X = 4?
```

Lesson 10, Exercise 17:

```
      WHAT VALUES OF A WILL BE USED IF THIS COMMAND IS GIVEN?

           DO STEP 73.7 FOR A = 5(10)35

      A.  5, 10, 15, 20, 25, 30, 35

      B.  10, 15, 20, 25, 30, 35

      C.  5, 15, 25, 35
```

Lesson 11: Parts. Lesson 11 explains how indirect (stored) steps are grouped into parts. Steps 12.1, 12.7, and 12.8, for example, are grouped as "Part 12," and can be executed by a single command:

```
      DO PART 12
```

The sequence of execution depends only upon the numerical order of the step numbers, and not upon the sequence in which they were written. Thus, steps 3.7, 3.2, and 3.8 will be executed in the order 3.2, 3.7, 3.8. Students have little difficulty with this concept except when step numbers end in zeros; some students cannot readily sort into numerical order a sequence like 3.5, 3.8, 3.10 (the correct order is 3.10, 3.5, 3.8).

Lesson 11, Exercise 5:

```
YOU CAN TYPE THE STEPS IN ANY ORDER, BUT AID WILL ALWAYS
DO THEM IN NUMERICAL ORDER.  WHICH STEP WILL BE DONE FIRST?

     17.4  TYPE X*Y
     17.5  SET N = 5
     17.2  SET X = 10
     17.3  SET Y = 2
```

Lesson 11, Exercise 11:

```
A PART (SET OF INDIRECT STEPS) IS ALSO CALLED A PROGRAM.
USE AID TO WRITE A PROGRAM THAT WILL LIST THE RADIUS,
DIAMETER, CIRCUMFERENCE, AND AREA OF A CIRCLE OF RADIUS R.
THEN USE THE PROGRAM FOR R = 10, 20, 30, 40, AND 50.
```

Lesson 12: The DEMAND command. In Lesson 12 the DEMAND command is introduced. The DEMAND command is used in AID programs for keyboard input. The DEMAND command can be used only indirectly, unlike previously introduced commands which can be used both directly and indirectly. DEMAND is used for numerical input only and the form is simple:

```
DEMAND X
```

where X is the variable name to which the input number is assigned.

Lesson 12 also introduces the TIMES clause which can be used to modify a DO command, in this way:

```
DO PART 7, 5 TIMES
```

Lesson 12, Exercise 4:

```
START AID AND WRITE A PROGRAM THAT WILL ASK YOU FOR 3
NUMBERS, A, B, AND C, AND THEN GIVE YOU THE AVERAGE OF
THE 3 NUMBERS. AFTER YOU HAVE TESTED YOUR PROGRAM, USE
IT TO FIND THE AVERAGE OF

     A = 179.053

     B = 23.7

     C = 271.0015
```

Lesson 12, Exercise 5:

```
WHAT COMMAND WOULD YOU USE IF YOU WANTED PART 2 DONE
7 TIMES?
```

Lesson 13: Test of Lessons 8 to 12. This lesson is the test for the second lesson block and is structured like other tests: the student is given only one try for each exercise, and no hints are provided although a student who cannot do an exercise can request the correct answer.

As for the first test (Lesson 6), the exercises are classified by type in Table 3, and are classified according to which of Lessons 8 through 12 are being tested in Table 4. An exercise that might be considered a test of more than one lesson is listed only once in Table 4.

Of the 33 exercises in Lesson 13, only 28 are properly test exercises, since three ask for opinions (Class 4 exercises) and two ask the student to use the AID interpreter but do not check the work done by the student while he is using AID.

Lesson 14: Block review. Lesson 14, like other block reviews, is optional; the review is recommended for students who missed more than three problems in the preceding test.

Lesson 14, Exercise 3:

> LESSON 8 WAS ABOUT THE "LET" COMMAND AND HOW TO USE IT TO DEFINE A FUNCTION. FUNCTIONS OF 2 AND 3 VARIABLES WERE DISCUSSED. INSTRUCTIONS FOR PRINTING AND DELETING A FUNCTION WERE GIVEN.
>
> DO YOU WANT TO REVIEW ANY OF LESSON 8?

The student who answers "yes" will be branched to the lesson review for Lesson 8 and then given his choice of which topics to review in what order.

Lesson 15: Relations and the use of the IF clause. Lesson 15 begins a new lesson block with the introduction of the most powerful of programming tools, the conditional clause. The conditional (IF) clause consists of the word "if" followed by a Boolean statement, and may be appended to any of the commands so far introduced.

```
SET Z = 2 IF X < 10

TYPE X IF X > 0

DO PART 5 IF M = N
```

Most of Lesson 15 concerns the syntax and meaning of logical statements. The following AID symbols for arithmetic relations are introduced:

<	less than;
>	greater than;
<=	less than or equal;
>=	greater than or equal;
#	not equal

The terms "positive," "negative," and "non-negative" are reviewed. The Boolean connectives "and" and "or" are also introduced although their meanings and the hierarchy for the connectives are not discussed extensively at this point. The students are required to write several programs using conditional clauses.

Lesson 15, Exercise 14:

STUDY THIS PROGRAM.

```
49.5  TYPE X IF X > Y
49.6  TYPE Y IF X <= Y
DO PART 49
```

IF X = 12.1 AND Y = 6, WHAT WILL AID ANSWER?

Lesson 15, Exercise 15:

USING AID, WRITE A PROGRAM THAT WILL FIND THE SMALLER OF TWO NUMBERS X AND Y. TRY SEVERAL DIFFERENT VALUES OF S AND Y.

Lesson 16: The TO command. Lesson 16 introduces the idea of conditional branching and provides additional practice in the use of conditional clauses. Although Lesson 16 is quite short (27 exercises), some of the programming problems are very difficult. Several sample programs are analyzed in detail with special emphasis on the order of execution.

Lesson 16, Exercise 3:

HERE IS A PROGRAM THAT CALCULATES THE AREA OF A RECTANGLE OF LENGTH L AND WIDTH W. IF EITHER L OR W IS NEGATIVE, PART 15 IS USED TO GIVE AN "ERROR" MESSAGE.

```
14.1  DEMAND L
14.2  TO PART 15 IF L < 0
14.3  DEMAND W
14.4  TO PART 15 IF W < 0
14.5  TYPE L*W
15.1  TYPE "DO NOT USE NEGATIVE NUMBERS."
```

WHICH STEPS WILL BE DONE IF L = 5 AND W = -3?

Lesson 16, Exercise 6:

WRITE A PROGRAM THAT WILL TYPE 3 NUMBERS A, B, AND C IN NUMERIC ORDER (THAT IS, THE SMALLEST FIRST, ETC.)

<u>Lesson 17: Debugging techniques</u>. Lesson 17 concentrates on the debugging technique of tracing the step-by-step execution of a program listing the changes in values of the variables used in the program.

Lesson 17, Exercise 3:

FOR PRACTICE, LET'S MAKE A TRACE OF THIS PROGRAM, ASSUMING A = 3.

```
31.3  DEMAND A
31.2  SET B = A↑2 - 10
31.3  SET C = A IF A > B
31.4  SET C = B IF A <= B
31.5  TYPE B
31.6  TYPE C
```

FILL IN THE VALUES OF C IN THIS TRACE (STARTING AT STEP 31.3).

STEP	A	B	C
31.1	3	-	-
31.2	3	-1	-
31.3	3	-1	?
31.4	3	-1	?
31.5	3	-1	?
31.6	3	-1	?

Lesson 18: The indirect use of the DO command. In Lesson 18 the indirect use of the DO command is introduced. Up to this point the student has used DO commands directly to execute programs or single steps. The DO command can also be used indirectly, as part of a stored program, or to execute subroutines. Frequently, a conditional clause is appended to indirect DO commands.

Several exercises in Lesson 18 provide the student with practice in determining the order of execution of steps in a program with conditional subroutine calls.

Lesson 18, Exercise 2:

WHEN AID COMES TO AN INDIRECT "DO" COMMAND, IT WILL DO THE STEP OR PART INDICATED AND THEN RETURN TO THE STEP AFTER THE "DO" COMMAND.

```
16.1  DO STEP 2.1 IF Q < 0
16.2  TYPE Q
 2.1  SET Q = -Q
DO PART 16
```

IF Q = 3, THE STEPS WILL BE DONE IN WHICH ORDER?

A	B	C	D
16.1	16.1	16.1	16.1
2.1	16.2	2.1	16.2
16.2	2.1		

Lesson 19: Debugging, permanent storage. The first part of Lesson 19 is an optional section of tips for writing and debugging programs. This section is primarily for students who have difficulty with the programming problems in the preceding lessons and covers such topics as planning and editing a program, distinguishing between and correcting several kinds of syntactic and semantic errors, and single-stepping through a program with commands like.

DO STEP 34.2.

The second half of the lesson describes the difference between core memory and disk storage, and teaches students how to store their programs on disk by using the AID file commands: USE, FILE, RECALL, and DISCARD.

Lesson 19, Exercise 5:

> SUPPOSE YOU FORGOT TO TYPE ONE OF THE STEPS IN A PROGRAM.
> IF THERE IS A STEP MISSING BETWEEN STEPS 2.5 AND 2.6, YOU
> CAN INSERT THE STEP BY USING WHAT STEP NUMBER?

Lesson 19, Exercise 13:

> WHAT COMMAND WOULD YOU USE TO FILE PART 29 AS ITEM 3?

Lesson 20: Test of Lessons 15 to 19. This lesson is the test for the third lesson block. Lesson 20 contains 27 exercises of which two are requests for opinions. The other 25 are grouped in Table 4 according to which of the preceding lessons they test.

Lesson 21: Block review. Lesson 21 is a review of the lessons tested by Lesson 20. Like other block reviews it is optional, but is recommended for students who miss more than three problems in the test. Unlike preceding block reviews this one does not call on lesson reviews as subroutines, since there were no lesson reviews for Lessons 16 to 19; instead, the students are given references to pertinent exercises in the lessons themselves.

Lesson 21, Exercise 8:

```
LESSON 19 EXPLAINED HOW TO PLAN, WRITE, AND EDIT A PROGRAM;
WHAT KINDS OF ERRORS THERE ARE AND HOW TO CORRECT THEM; AND
HOW TO USE PERMANENT STORAGE.

DO YOU WANT TO REVIEW ANY OF LESSON 19?
```

A student who answers "yes" will be given a list of the topics in Lesson 19 and the exercises that treat each of those topics.

Lesson 22: The FORM statement. The FORM statement, used in AID to specify the form of teletype output, is introduced in Lesson 22. Up to this point students use the standard AID form for teletype output, but in this case they learn how to define new forms for themselves. A FORM statement allows the programmer to determine the spacing used in the output, to define positions for more than one number per line, and to insert text into the output. The number of digits to be printed can be specified in a FORM statement and any numbers printed will be rounded to fit the space specified. The FORM statement can also be used to type text, although a TYPE command can equally well be used for this purpose, as explained in the lesson.

Lesson 22, Exercise 16:

WRITE A PROGRAM (PART 7) THAT WILL GET A VALUE OF X FROM THE USER AND THEN TYPE X, X SQUARED, X CUBED, AND X TO THE FOURTH POWER ON ONE LINE, LIKE THIS:

```
    X = 3.0  X↑2 = 9.0  X↑3 = 27.0  X↑4 = 81.0
```

SAVE THIS PROGRAM BY TYPING "USE FILE 100" AND "FILE PART 7 AS ITEM 1" BECAUSE YOU WILL NEED IT FOR ANOTHER PROBLEM LATER.

Lesson 22, Exercise 19:

START AID AND CHANGE THE PROGRAM YOU WROTE FOR PROBLEM 16 SO THAT IT PRINTS THE TITLE "POWERS OF X". ALSO PUT 1 OR 2 BLANK LINES AFTER THE TITLE. TRY THE PROGRAM FOR X = 1.2, .7, 6.25.

Lesson 23: Loops. Lesson 23 introduces the vital subject of loops. Several later lessons will also deal with this topic, and the loops used in this lesson are of the simplest sort. These loops use an incremented variable to count the number of times the loop is used. The students are taught to set the counter to an initial value before the loop, to add a constant to the counter inside the loop, and to end the loop with a conditional branch in which the value of the counter is compared to a constant.

The lesson starts with a detailed study of several sample programs incorporating simple loops. Several exercises are intended to illustrate the most common errors made in writing loops, and how to detect and correct these errors.

Lesson 23, Exercise 4:

```
            54.1  SET A = 1

            54.2  TYPE "CONVERSION OF FEET TO INCHES"

            54.3  TYPE A, 12*A IN FORM 2

            54.4  SET A = A + 1

            54.5  TO STEP 54.3

            FORM 2:

            ← ← ← FEET = ← ← ← ← INCHES

        WHICH STEP IS WRONG?
```

Lesson 23, Exercise 5:

```
            56.1  LET F(X) = 3*X↑3 + 5

            56.2  TYPE X, F(X)

            56.3  SET X = X + .5

            56.4  TO STEP 56.2 IF X < 3.5

        IN PART 56 THERE IS A MISSING STEP.  IT SHOULD GO BEFORE

        WHAT STEP?
```

Lesson 24: Loops with variables in the exit condition. In Lesson 24 the study of loops is continued and extended to include loops in which the counter is compared not to a constant but to another variable. The use of variables for initial values of the counter is also discussed.

Lesson 24, Exercise 12:

```
        IN A LOOP BOTH THE INITIAL VALUE AND THE VALUE USED FOR

        COMPARISON IN THE EXIT CONDITION CAN BE VARIABLES.  THE

        VALUES OF THE VARIABLES WILL BE GIVEN BY THE USER BEFORE

        THE LOOP IS USED.
```

```
6.51  DEMAND L
6.52  DEMAND U
6.53  SET X = L
6.54  TYPE X*3/17
6.55  SET X = X + 1
6.56  TO STEP 6.54 IF X < U
```

WHAT VARIABLE IS USED FOR THE INITIAL VALUE?

Lesson 24, Exercise 13:

HERE IS A MORE COMPLICATED PROGRAM. TRY TO FIND OUT WHAT THE PROGRAM DOES.

```
3.1  SET N = 1
3.2  SET S = 0
3.3  TYPE N
3.4  SET S = S + N
3.5  SET N = N + 1
3.6  TO STEP 3.3 IF N < 4
3.7  TYPE S
```

Lesson 25: Loops and the FOR clause. In Lesson 25 loops are reviewed and practiced. The students are taught that many programs with loops are equivalent to simpler programs that are executed iteratively by using a DO command with a FOR clause; in several exercises the students are asked to rewrite looping programs so as to be able to use the FOR clause, and are advised to use this simpler approach because of the greater ease in writing and debugging such programs. Students are cautioned that not every looping program can be rewritten in this way, and examples are shown to illustrate the point.

Lesson 25, Exercise 7:

```
4.1  SET C = 1

4.2  TYPE 60/C

4.3  SET C = C + 1

4.4  TO STEP 4.2 IF C < 7
```

WHAT DOES THIS PROGRAM DO?

Lesson 25, Exercise 7.4:

HERE IS THE ABOVE PROGRAM REWRITTEN TO USE A "FOR" CLAUSE:

```
4.1  TYPE 60/C

DO STEP 4.1 FOR ...
```

COMPLETE THE "FOR" CLAUSE.

Lesson 26: Loops with a DEMAND command. In interactive programming it is a frequent practice to allow for input from the user during the execution of a loop; in AID this is done with a DEMAND command. When a DEMAND command is executed and the response from the user is empty, i.e., a carriage return only, AID terminates the execution of the program at that point. Because of this feature, loops that incorporate DEMAND commands need no conditional clause to determine when to cease iteration.

Lesson 26, Exercise 2:

IF THE USER TYPED THE RETURN KEY ONLY, INSTEAD OF TYPING A VALUE FOR L, AID WOULD STOP THE PROGRAM (THIS IS LIKE AN EXIT CONDITION).

```
5.1  DEMAND R

5.2  SET A = 3.14159265*R↑2

5.3  TYPE A
```

```
5.4 TO STEP 5.1
```

WHAT DOES PART 5 DO?

Lesson 26, Exercise 5:

WRITE A PROGRAM TO CONVERT INCHES TO FEET AND INCHES.
USE A "DEMAND" COMMAND IN A LOOP. START AID AND TEST
THESE VALUES:

159 INCHES
17 INCHES
44 INCHES

Lesson 27: Test of Lessons 22 to 26. This lesson, the block test for the fourth lesson block, contains 30 exercises, of which 29 are test exercises. The test exercises are classified by type in Table 3, and are classified in Table 4 by the lesson being tested.

Lesson 28: Block review. Lesson 28 is the block review for the fourth lesson block, and like the block review in Lesson 21 refers students to particular parts of the teaching lessons for review of specific subjects. In addition, the contents of the three previous lesson blocks are summarized and the students are referred to the appropriate block reviews (Lessons 7, 14, and 21) for review and practice on the topics covered in preceding blocks.

Lesson 29: Absolute values. Lesson 29 reviews the concept of absolute value of real numbers and introduces the AID notation for absolute value: !x! . The hierarchy of arithmetic operations is reviewed and the place of absolute value in the hierarchy is specified. The use of absolute values for finding Euclidean distances is discussed.

Lesson 29, Exercise 17:

```
!17.1 - 7.9! IS THE DISTANCE BETWEEN 7.1 AND WHAT?
```

Lesson 29, Exercise 19:

```
WRITE A PROGRAM TO FIND WHICH OF THE THREE NUMBERS A, B, AND
C IS CLOSEST TO 13/17.  HAVE YOUR PROGRAM PRINT ONE OF THESE
MESSAGES:

     A IS CLOSEST TO 13/17.

     B IS CLOSEST TO 13/17.

     C IS CLOSEST TO 13/17.
```

Lesson 30: SIN and COS. Lesson 30 is an optional lesson introducing the basic trigonometric functions for students who are interested and have some background in the subject. Some basic trigonometric notions are reviewed, the two standard AID functions are introduced, and there are a few practice problems involving the definitions of the tangent and secant functions. None of the problems are of great difficulty and the lesson is very short (17 exercises).

Lesson 30, Exercise 9:

```
DEFINE A TANGENT FUNCTION, T(X), WHICH WILL FIND THE
TANGENT WHEN X IS GIVEN IN RADIANS.  START THE AID
INTERPRETER AND USE YOUR FUNCTION TO FIND THE TANGENT
WHEN X = 0, 2.4, 3.1, -6.
```

Lesson 31: Exponential and logarithmic functions. Lesson 31 is also optional and provides a brief introduction to the exponential and logarithmic functions that can be used in AID programming; this lesson is intended only for students with the appropriate background and interest. The concepts of exponential and logarithmic functions are reviewed and the AID notation for e^x and $\ell n(x)$ is introduced. There are a few problems on the conversion of bases, although the treatment of this topic is adequate only for students who have already studied it.

Lesson 31, Exercise 10:

> USING AID, FIND THE VALUES OF EXP(X) FOR X = .5, 1.0, 1.5,...,10.0

Lesson 31, Exercise 12:

> DEFINE A FUNCTION T(X) WHICH WILL COMPUTE THE LOGARITHM TO THE BASE 2 OF X. TEST THE FUNCTION FOR THESE VALUES OF X:
>
> .6, 5, 8.7, 100

Lesson 32: Lists. The first three lessons in this lesson block were relatively short and easy, and were devoted more to mathematical concepts than programming concepts. Lesson 32, in contrast, is quite long and very difficult, introducing one of the most complex topics in programming: the storage and use of indexed arrays.

Lesson 32 begins with a discussion of indices and the AID notation for them. A simple program for inputting a list of numbers is shown and several difficult programming problems whose solution depends upon the retrieval of data from stored lists follow.

Lesson 32, Exercise 8:

> WRITE A PROGRAM TO FIND THE AVERAGE OF THE NUMBERS IN A LIST OF TEN NUMBERS.

Lesson 32, Exercise 19:

> WRITE A PROGRAM TO FIND AND PRINT ALL THE NUMBERS LESS THAN 30 IN A LIST OF 10 NUMBERS.

Lesson 33: Using loops with lists of numbers. Lesson 33 continues the treatment of stored arrays introduced in Lesson 32. The first topic covered in Lesson 33 is a method for counting and simultaneously storing the elements in a list so that later computations, such as finding the average, can be done more easily. Students are shown several examples of looping programs that use data stored in lists, and are asked to write five programs of this kind.

Lesson 33, Exercise 3:

```
WHAT IS THE PURPOSE OF THIS PROGRAM?

     2.1  SET I = 1
     2.2  TYPE L(I) IF L(I) # 0
     2.3  SET I = I + 1
     2.4  TO STEP 2.2 IF I <= N
```

Lesson 33, Exercise 6:

```
SUPPOSE L IS THE LIST 12, 0, -7, 0, 0, 8.  (N IS THE
LENGTH OF L)  HERE IS A PROGRAM THAT BUILDS A NEW LIST A.
WHAT IS THE VALUE OF A(1) AFTER THIS PROGRAM IS RUN?

     7.1  SET I = 1
     7.2  SET A(I) = 1 IF L(I) # 0
     7.3  SET A(I) = 0 IF L(I) = 0
     7.4  SET I = I + 1
     7.5  TO STEP 7.2 IF I <= N
```

Lesson 33 is not very long (23 exercises) but there is a rather high proportion of programming problems (5) of medium difficulty.

Lesson 34: Test of Lessons 29 to 33. Lesson 34 is the test for the fifth lesson block. There are 26 test exercises and one request for student opinions in the lessons. The 26 test exercises are classified in Table 4 by the lesson they are testing.

Since Lessons 30 and 31 were optional, the corresponding exercises in the test (Exercises 8 through 17) are also optional.

Lesson 35: Block review. Lesson 35 is the block review for the fifth lesson block. About half of the lesson contains review exercises and the other half gives students references to exercises that cover specific topics.

Lesson 36: Nested loops and decrementing counters. This lesson continues work on loops, and starts by showing several examples of programs in which the conditional clause that determines the number of iterations is formed by comparing the value of the counter to the value of an algebraic expression (rather than to a simple variable or constant). Nested loops are introduced, and decremented counters are used in several problems.

Lesson 36, Exercise 10:

> WRITE A PROGRAM USING NESTED LOOPS WHICH WILL MAKE ONE TABLE OF INTEREST CORRESPONDING TO A RATE OF 6% AND THE FOLLOWING GROUPS OF PRINCIPLES:
>
> 50, 100, 150, 200; 250, 500, 750, 1000; 1250, 2500, 3250, 5000.

249

Lesson 37: SUM, PROD, MAX, and MIN. Four AID functions that are very useful for finding the sum, product, maximum or minimum of a sequence of numbers are introduced in this lesson. The sequence of numbers may be expressed by simply listing them as the argument of the function:

```
SUM(4.53, 3.72, 6.29, 7.81)
```

or by a formula:

```
SUM(J = 1, 2, 3, 4 : J*3)
```

or by giving the variable name of a stored list:

```
SUM(J = 1, 2, 3, 4 : L(J))
```

When the sequence is defined by formula or is to be found in a stored list, the indices to be used must be specified either by a simple listing of the indices or by a "range specification."

The lesson starts with a brief review of numeric sequences and formulas for sequences before the syntax of the new AID functions is introduced.

Lesson 37, Exercise 18:

```
USE AID TO FIND THE PRODUCT OF

          1, 4, 9, ..., 100
```

Lesson 37, Exercise 26:

```
WHAT WILL AID ANSWER?

          SET X = 24

          SET Y = X/3

          TYPE MIN(SUM(X,Y), PROD(X,Y))
```

Lesson 38: Arrays. In Lesson 38 two dimensional arrays are introduced. After some discussion of ordered indices, the lesson gives a simple program for storing data in two-dimensional arrays and then gives the student several programming problems using data stored in this form.

Lesson 38, Exercise 8:

STORE THIS TABLE AS A 5 BY 3 ARRAY A.

1	5	25
2	10	100
3	15	225
4	20	400
5	25	625

USE "TYPE A" TO GET A LISTING OF YOUR ARRAY.

Lesson 38, Exercise 9:

WRITE A PROGRAM THAT WILL ADD THE COLUMNS OF A 5 BY 3 ARRAY. THE PROGRAM SHOULD TYPE

```
THE SUM OF COLUMN 1 IS ...
THE SUM OF COLUMN 2 IS ...
THE SUM OF COLUMN 3 IS ...
```

Lesson 39: More about arrays and lists. Lesson 39 discusses the limitations on the dimension of AID arrays and the permissible range of the indices. The use of algebraic expressions as indices is also mentioned. Several quite difficult programming problems are given, as illustrated in the example below.

Lesson 39, Exercise 8:

STORE THE FOLLOWING ARRAYS:

A: 3 BY 4   WHERE A(I,J) = 3*I - J

B: 3 BY 4   WHERE B(I,J) = -2*I + J

THEN FORM A NEW 3 BY 4 ARRAY, E, WHOSE ELEMENT IN THE ITH ROW AND JTH COLUMN IS THE MAXIMUM OF THE ELEMENTS IN THE SAME POSITION IN THE ARRAYS A AND B.

Lesson 40: Conditional definition of functions. One important feature of AID is the simplicity of its conditional definition of functions. Since the conditional definition of functions depends upon the use of Boolean expressions, the lesson begins with a brief survey of these expressions. The syntax of conditional definitions is then given and a number of examples are presented.

In mathematics, we often encounter functions that cannot be defined by a single formula but may, instead, be defined by several formulas each applying to a particular part of the domain of the function; the definitions of such functions usually are given in a form such as

If CONDITION 1 then f(x) = EXPRESSION 1

If CONDITION 2 then f(x) = EXPRESSION 2
etc.

In AID this definition is expressed as follows:

LET F(X) = (CONDITION 1: EXPRESSION 1; CONDITION 2:
EXPRESSION 2; ...)

In some cases a function may be defined like this:

If CONDITION then f(x) = EXPRESSION 1

Otherwise f(x) = EXPRESSION 2.

This kind of definition is written in AID by simply omitting the final condition and using the final expression as the definition of the function for all cases where one of the preceding conditions does not hold:

```
LET F(X) = (CONDITION: EXPRESSION 1;  EXPRESSION 2)
```

Lesson 40, Exercise 13:

```
WRITE THE CONDITIONAL DEFINITION OF A FUNCTION F(X) SUCH THAT
IF X < 0 THEN F(X) = X↑2
IF X >= 0 THEN F(X) = X↑3
```

Lesson 40, Exercise 21:

```
LOCAL FIRST CLASS POSTAL RATES UP TO 32 OUNCES ARE DEFINED
AS FOLLOWS, WHERE W IS IN OUNCES:
     IF W <= 13, THE COST IS $.06 PER WHOLE OUNCE, PLUS
          $.06 FOR ANY FRACTION OF AN OUNCE.
     IF 13 < W <= 16, THE COST IS $.80
     IF 16 < W <= 24, THE COST IS $.98
     IF 24 < W <= 32, THE COST IS $1.16.
WRITE AND RUN A PROGRAM TO COMPUTE POSTAGE COSTS.
```

<u>Lesson 41: Test of Lessons 36 to 40</u>. Lesson 41, like other tests, allows the students only one trial per exercise. If a student cannot answer a question, he can request the correct answer and go to the next exercise. There are no hints provided by the program. The number of exercises of each kind are listed in Table 3, and are also classified in Table 4 according to which teaching lesson is being tested.

As before, test exercises may test more than one lesson but each exercise is listed only once in Table 4.

Lesson 42: Block review. Lesson 42 is the block review for the sixth lesson block, covering the same lessons as the test in the preceding Lesson 41. Here again students who want to review a particular topic are referred to appropriate exercises in the teaching lessons.

Lesson 43: Recursive functions. Recursive functions are defined in the same form as other conditional functions and are presented only for those students with appropriate background and interests. The most commonly used examples of recursive functions, such as the factorial and the Fibonacci numbers, are used as examples. Since this lesson is intended only for those students who are somewhat more sophisticated mathematically, the exercises are correspondingly more difficult than other lessons in the course.

Lesson 43, Exercise 6:

```
WRITE A PROGRAM CONTAINING A RECURSIVE FUNCTION N(A,X,E)
THAT USES NEWTON'S ALGORITHM FOR OBTAINING THE APPROXIMATE
SQUARE ROOT OF THE NUMBER A.
     A = POSITIVE NUMBER WHOSE SQUARE ROOT WILL BE APPROXIMATED
     X = FIRST APPROXIMATION TO SQRT(A)
     E = ALLOWABLE DIFFERENCE BETWEEN X↑2 AND A
```

Lesson 44: AND, OR, and NOT; truth tables. The Boolean connectives AND, OR, and NOT are discussed in detail in this lesson and the hierarchy is given explicitly. Truth tables are introduced and the truth tables for various compound statements are used in the exercises. This lesson is primarily an introduction to sentential logic for students who have not previously studied the subject, and the exercises also serve as vehicles for further practice using AID.

Lesson 44, Exercise 10:

WHICH OF THE FOLLOWING STATEMENTS ARE TRUE?

A. -1 < 2 OR 3 > 4

B. -1 < 2 AND 3 > 4

C. 6 = 7 OR 9 < 1

D. .5 # 0 OR 0 < .5

E. 9 >= 11 OR (6 > 1 AND 1 < 2)

Lesson 45: TV(X) and the FIRST function. TV and FIRST are two advanced AID functions that are useful in special circumstances. TV is used only for Boolean expressions and takes on the values 0 (false) and 1 (true). The function FIRST is used to find the first number in a sequence that satisfies a given condition; for example, FIRST(K = 6(2)14 : (K/2)↑2 > 24) will give the first number K in the sequence 6, 8, 10, 12, 14 such that (K/2)↑2 is greater than 24.

Lesson 45, Exercise 5:

HOW WOULD YOU DEFINE A FUNCTION F SUCH THAT

F(X) = 1 IF X IS FALSE

F(X) = 0 IF X IS TRUE

Lesson 45, Exercise 16:

GIVEN THE SEQUENCE

0, .2, .4, .6, ..., Z*.2, ...

WRITE A PROGRAM THAT WILL FIND THE FIRST MEMBER OF THIS SEQUENCE SUCH THAT SIN(Z*.2) < 0 AND THE FIRST MEMBER AFTER THIS ONE.

Lesson 46: LET and Boolean expressions; debugging tools. The first topic covered in Lesson 46 is the use of LET to assign a variable name to a Boolean expression. For example, to assign to the variable S the sentence "P or not Q or not R", the following command is used:

```
        LET S = P OR NOT Q OR NOT R
```

P, Q, and R must, of course, be given Boolean values before S is called.

The remainder of Lesson 46 is devoted to several useful debugging tools. The first of these is the GO command which requests AID to continue executing an interrupted program. The GO command is used when the execution of a program is interrupted because of an error; after correcting the error, the programmer may type "GO" to continue. Another command that helps to debug complex programs is DONE. DONE is used indirectly, i.e., as part of the stored program, and its effect is to cause the execution of the program to stop at that point. DONE is most often used conditionally to stop execution under certain conditions, for example, when a variable assumes a value that is out of bounds:

```
        6.58  DONE IF X > 10↑8
```

Lesson 46, Exercise 3:

```
        WHAT VALUE WILL AID TYPE FOR A?

             LET A = B AND C

             SET B = TRUE

             SET C = FALSE

             TYPE A
```

Lesson 46, Exercise 14:

THE "DONE" COMMAND HAS A PARTICULAR APPLICATION FOR LOOPS. SOMETIMES YOU MIGHT WANT TO TEST A LOOP FOR ONLY ONE OR TWO LOOPINGS. THE CONDITIONAL DONE COMMAND CAN BE USED FOR THIS.

```
5.1  SET N = 1
5.2  SET K = 1/N
5.3  TYPE N,K
5.4  SET N = N + 1
5.5  TO STEP 5.2 IF N <= 100
```

WHICH OF THE FOLLOWING COULD BE USED TO STOP THE LOOPING AFTER TWO LOOPS?

A. 5.42 DONE IF K = 1/4

B. 5.21 DONE IF N = 5

C. 5.36 DONE IF N = 2

D. 5.6 DONE IF N = 2

N. NONE OF THE ABOVE

Lesson 47: More standard AID functions. The two AID functions that are covered in this lesson are DP (digit part) and XP (exponent part). The lesson starts with a review of scientific notation, explaining that two parts of a number in scientific notation are the digit part and the exponent part. The digit part of a number can be found by using the digit part function, DP(X), and the exponent part by using the exponent part function, XP(X).

Lesson 47, Exercise 4:

WHAT IS THE EXPONENT PART OF 8325.6?

Lesson 47, Exercise 12:

> WRITE A PROGRAM THAT WILL TAKE ANY NUMBER X AND ROUND IT
> TO THREE SIGNIFICANT DIGITS.

Lesson 48: Test of Lessons 43 to 47. Lesson 48 is the test for the seventh, and last, lesson block in the course. Of the 32 exercises in the lesson, two are requests for students' opinions. The other 30 exercises test one of Lessons 43 to 47, as shown in Table 4.

Lesson 48 has a rather high proportion of true-false exercises (7) and no programming problems.

Lesson 49: Block review. Lesson 49 is the review of the seventh lesson block and covers Lessons 43 to 47. The contents of each of the teaching lessons in the block are summarized briefly, and students are asked if they want to review any part of the lessons. If a student wants to review a particular topic, he is given a reference to pertinent exercises.

Lesson 50: Programming problems. Lesson 50 is not part of a lesson block, nor is it a teaching lesson, but rather a collection of lengthy and difficult programming problems for students who want to practice the programming skills they have acquired in the course. Some of the exercises require considerable mathematical knowledge and sophistication; these exercises are intended only for the student with an appropriate background. Other problems, though difficult, are accessible to students with less mathematical background.

APPENDIX

Table of Contents: Teaching Lessons

	Number of Exercises	List of Exercises
Lesson 1: Using the Instructional Program		
How to answer	4	1, 2, 3, 4
How to erase	3	5, 6, 7
How to get hints and answers	2	8, 9
How to use Ctrl-G	4	10, 11, 11.1, 12
How to sign on and off	3	13, 13.1, 13.2
Prompted decisions*	2	14, 15
Lesson 2: Using AID for Arithmetic		
The AID interpreter	4	1, 2, 17, 18
Symbols for arithmetic operations	9	8, 9, 10, 11, 12, 13, 14, 14.1, 14.2
The TYPE command	12	3, 4, 5, 6, 7, 15, 16, 16.1, 16.2, 19, 19.1, 19.2
Prompted decisions	3	20, 21, 21.1
Lesson 3: Order of Arithmetic Operations		
Use of parentheses	22	1, 2, 2.1, 3, 4, 4.1 to 4.6, 5, 5.1, 6, 7, 8, 8.1, 8.2, 9, 20, 20.1, 20.2
Hierarchy of operations	21	10, 10.1, 10.2, 11, 11.1, 12, 12.1, 12.2, 13, 13.1, 13.2, 14, 15, 16, 21, 22, 23, 24, 24.1 to 24.3
Negative numbers	3	17, 18, 19
Prompted decisions	3	25, 26, 27
Lesson 4: Exponents and Scientific Notation		
Exponents	12	1, 1.1, 2, 2.1, 2.2, 2.3, 5, 5.1, 22, 22.1 to 22.3
Using zero and one as exponents	2	3, 4

*Exercises that ask the student to state a preference for the sequence of instruction.

	Number of Exercises	List of Exercises
Order of operations	20	6, 6.1 to 6.8, 7, 8, 9, 9.1, 10, 10.1, 11, 12, 12.1 to 12.3
Using fractional exponents	3	13, 14, 15
Negative exponents	9	16, 17, 17.1, 17.2, 18, 19, 20, 20.1, 21
Reading scientific notation	7	23, 23.1, 23.2, 25, 25.1, 27, 27.1
Writing scientific notation	5	24, 24.1, 26, 26.1, 26.2
Prompted decisions	3	28, 29, 30
Lesson 5: The SET and DELETE Commands		
The SET command	47	1, 2, 3, 4, 5, 5.1, 6, 7, 7.1, 7.2, 8, 9, 10, 11, 11.1, 11.2, 12, 12.1, 13, 13.1, 13.2, 14, 14.1, 15, 15.1, 16, 16.1, 17, 17.1, 18, 18.1, 18.2, 19, 19.1, 19.2, 20, 20.1, 21, 21.1, 21.2, 30, 30.1, 30.2, 30.3, 31, 31.1, 31.2
The DELETE command	6	22, 23, 23.1, 24, 25, 26
The multiple TYPE command	7	27, 27.1, 28, 29, 29.1 to 29.3
Prompted decisions	3	32, 33, 34
Lesson 8: The LET Command		
Functions of one variable	30	1, 1.1 to 1.3, 2, 2.1, 2.2, 3, 4, 5, 6, 7, 8, 8.1, 9, 9.1, 10, 10.1 to 10.3, 22, 23, 23.1, 28, 28.1, 29, 30, 30.1, 30.2, 31
Functions of two or more variables	15	11, 11.1 to 11.5, 12, 13, 14, 15, 16, 17, 27, 27.1, 27.2
Substituting algebraic expressions for variables	11	18, 19, 20, 20.1 to 20.4, 21, 24, 24.1, 24.2
Printing and deleting definitions of functions	2	25, 26
Prompted decisions	4	32, 33, 34, 35

260

	Number of Exercises	List of Exercises
Lesson 9: Some Standard AID Functions		
The SQRT function	5	1, 2, 3, 3.1, 3.2
The IP function	7	4, 5, 6, 7, 8, 8.1, 8.2
The FP function	9	9, 10, 11, 12, 13, 14, 14.1, 14.2, 15
The SGN function	6	16, 17, 18, 19, 20, 21
Prompted decisions	4	22, 23, 24, 25
Lesson 10: Indirect Steps, the DO Command, the FOR Clause		
Step numbers and the DO command	12	1, 1.1, 2, 3, 4, 5, 5.1, 6, 6.1, 7, 12, 12.1
Deleting, replacing, and printing indirect steps	3	8, 9, 10
The FOR clause	2	11, 11.1
Range specifications	13	13, 13.1, 14, 15, 15.1, 15.2, 16, 16.1, 16.2, 17, 18, 19, 19.1
Prompted decisions	3	20, 21, 22
Lesson 11: Parts		
Parts	2	1, 2
DO PART ...	3	3, 4, 4.1
DO PART ... FOR ...	5	8, 9, 10, 11, 11.1
Printing and deleting parts	6	12, 12.1, 13, 13.1, 14, 14.1
Sequence of execution	10	5, 5.1, 6, 7, 7.1, 15, 15.1 to 15.4
Prompted decisions	4	16, 17, 18, 19
Lesson 12: The DEMAND Command		
The DEMAND command	7	1, 1.1, 2, 3, 3.1, 4, 4.1
Answering a DEMAND with a "return"	2	7, 7.1
DO PART ..., ... TIMES	2	5, 6
Prompted decisions	3	8, 9, 10
Lesson 15: Relations and the Use of the IF Clause		
Relation symbols	23	1, 1.1, 1.2, 3, 3.1 to 3.4, 4, 4.1 to 4.3, 5, 5.1 to 5.4, 6, 6.1 to 6.5

261

	Number of Exercises	List of Exercises
The number line	8	2, 2.1 to 2.7
Positive and negative	6	7, 8, 9, 9.1, 9.2, 10
The IF clause	14	11, 12, 12.1, 13, 13.1, 14, 14.1, 14.2, 15, 15.1, 16, 17, 17.1, 18
Using AND and OR in IF clauses	7	19, 19.1, 20, 20.1, 21, 21.1, 21.2
Prompted decisions	4	22, 23, 24, 25
Lesson 16: The TO Command		
The TO command	7	4, 4.1, 4.2, 5, 6, 6.1, 6.2
Endless loops	7	2, 2.1 to 2.6
Sequence of execution	10	1, 1.1 to 1.4, 3, 3.1 to 3.4
Prompted decisions	3	7, 8, 9
Lesson 17: Debugging Techniques		
Tracing values of variables	12	1, 1.1, 1.2, 2, 2.1 to 2.4, 3, 3.1 to 3.3
Sequence of execution	1	7
Tracing expected output	7	4, 5, 5.1 to 5.5
Writing a complete trace	4	6, 7.1, 7.2, 7.3
Prompted decisions	3	8, 9, 10
Lesson 18: The Indirect Use of the DO Command		
The indirect use of DO	25	1.1 to 1.20, 4, 4.1, 5, 6, 7
Sequence of execution	6	2, 2.1, 2.2, 3, 3.1, 3.2
Prompted decisions	3	8, 9, 10
Lesson 19: Debugging, Permanent Storage		
Planning a program	6	2, 2.1 to 2.4, 3
Editing the program	5	4, 4.1, 5, 5.1, 8
Testing the program	2	6, 11
Syntax and semantic errors	3	7, 7.1, 7.2
Executing the program step-by-step	3	9, 9.1, 10
Disk storage	9	12, 13, 14, 15, 15.1, 16, 17, 18, 19
Prompted decisions	4	1, 1.1, 20, 21

	Number of Exercises	List of Exercises
Lesson 22: The FORM Statement		
The FORM statement	13	1, 2, 3, 4, 5, 6, 7, 8, 10, 10.1, 10.2, 11, 12
Rounding	3	9, 9.1, 12.1
TYPE "..."	2	17, 17.2
TYPE ←	3	18, 19, 19.1
Using a FORM statement for more than one number	5	13, 14, 15, 16, 16.1
Prompted decisions	3	20, 21, 22
Lesson 23: Loops		
Loops	33	1, 1.1, 1.2, 2, 2.1 to 2.14, 3, 3.1, 3.2, 4, 4.1, 4.2, 5, 5.1, 6, 6.1, 6.2, 7, 7.1, 8, 8.1
Prompted decisions	3	9, 10, 11
Lesson 24: Loops with Variables in the Exit Condition		
Using a variable in the exit condition	16	1, 1.1 to 1.3, 2, 2.1 to 2.4, 3, 4, 4.1, 4.2, 5, 10, 11
Using a variable for the initial value	3	12, 12.1, 12.2
Common errors in loops	4	6, 7, 8, 9
Other ways to use loops	9	13, 13.1 to 13.8
Prompted decisions	3	14, 15, 16
Lesson 25: Loops and the FOR Clause		
Replacing a loop with DO PART ... FOR ...	17	1, 2, 3, 4, 5, 6, 6.1 to 6.4, 7, 7.1 to 7.4, 8, 8.1
Prompted decisions	3	9, 10, 11
Lesson 26: Loops with a DEMAND Command		
Loops with a DEMAND command	16	1, 2, 3, 3.1, 4, 4.1 to 4.3, 5, 5.1, 6, 6.1, 6.2, 7, 8, 8.1
Prompted decisions	3	9, 10, 11

	Number of Exercises	List of Exercises
Lesson 29: Absolute Value		
Absolute value	14	1, 1.1, 2, 2.1 to 2.3, 3, 3.1, 4, 5, 6, 15, 15.1, 15.2
Hierarchy of operations	8	7, 8, 9, 10, 11, 12, 13, 14
Distance	8	16, 17, 18, 18.1 to 18.3, 19, 19.1
Prompted decisions	3	20, 20.1, 21
Lesson 30: SIN and COS		
SIN and COS	10	1, 2, 3, 4, 5, 5.1, 5.2, 6, 6.1, 10
Radians and degrees	2	7, 7.1
Other trigonometric functions	3	8, 9, 9.1
Prompted decisions	2	11, 12
Lesson 31: Exponential and Logarithmic Functions		
Exponents and bases	8	1, 1.1, 2, 2.1, 3, 3.1, 4, 4.1
Exponential functions	8	5, 6, 6.1 to 6.3, 7, 8, 8.1
EXP(X)	3	9, 10, 10.1
LOG(X)	3	11, 12, 12.1
Prompted decisions	2	13, 13.1
Lesson 32: Lists		
Lists and indices	16	1, 1.1 to 1.4, 2, 2.1, 2.2, 3, 4, 4.1 to 4.3, 16, 17, 18
Programs that use lists	25	5, 5.1, 6, 6.1 to 6.8, 7, 7.1, 7.2, 8, 8.1, 9, 10, 11, 11.1, 11.2, 19, 19.1 to 19.3
LET S BE SPARSE	4	12, 13, 13.1, 13.2
Printing a list	2	14, 15
Prompted decisions	3	20, 21, 22
Lesson 33: Using Loops with Lists of Numbers		
Using loops with lists	14	1, 2, 2.1, 3, 3.1, 4, 4.1, 5, 5.1, 5.2, 7, 7.1, 8, 8.1
Using loops to make new lists	6	6, 6.1 to 6.3, 9, 9.1
Prompted decisions	3	10, 11, 12

	Number of Exercises	List of Exercises
Lesson 36: Nested Loops and Decrementing Counters		
Using algebraic expressions in the exit condition	9	1, 1.1, 2, 2.1, 3, 3.1, 3.2, 4, 4.1
Nested loops	29	5, 5.1 to 5.4, 6, 6.1 to 6.6, 7, 7.1 to 7.6, 8, 8.1 to 8.4, 9, 9.1, 9.2, 10, 10.1
Decrementing counters	17	11, 11.1, 11.2, 12, 12.1 to 12.3, 13, 13.1, 13.2, 14, 14.1 to 14.3, 15.1, 16, 16.1
Prompted decisions	3	17, 18, 19
Lesson 37: SUM, PROD, MAX, and MIN		
Sequences and formulas	4	1, 2, 3, 4
SUM used with formulas	9	5, 6, 7, 8, 9, 10, 11, 12, 12.1
SUM used with lists	10	13, 14, 15, 20, 20.1 to 20.4, 27, 27.1
PROD	5	16, 17, 18, 18.1, 19
MAX	4	21, 23, 25, 25.1
MIN	5	22, 24, 25.2, 28, 28.1
Using SUM, PROD, MAX and MIN with listed arguments	4	26, 26.1 to 26.3
Prompted decisions	2	29, 30
Lesson 38: Arrays		
Arrays	4	1, 1.1, 2, 3
Storing arrays	14	4, 4.1, 5, 6, 6.1 to 6.4, 7, 7.1, 7.2, 8, 10, 10.1
Using arrays	7	6.5, 6.6, 6.7, 9, 9.1, 10.2, 10.3
LET A BE SPARSE	6	11, 11.1, 12, 12.1 to 12.3
Prompted decisions	3	13, 14, 15
Lesson 39: More about Arrays and Lists		
Subscripts for arrays	12	1, 1.1, 2, 2.1, 2.2, 9, 10, 10.1 to 10.3, 11, 11.1
Nested DO commands used to store arrays	13	3, 3.1, 4, 5, 5.1, 5.2, 6, 6.1, 6.2, 7, 7.1, 8, 8.1

	Number of Exercises	List of Exercises
Using arrays	9	12, 12.1 to 12.3, 13, 14, 14.1 to 14.3
Prompted decisions	3	15, 16, 17
Lesson 40: Conditional Definition of Functions		
Boolean expressions	6	1, 2, 3, 4, 5, 6
Conditional expressions	6	7, 8, 9, 10, 11, 11.1
Conditional definition of functions	13	12, 13, 13.1 to 13.3, 16, 17, 18, 18.1, 21, 22, 23, 24
Terminating clause in conditional definition of functions	2	14, 15
Ordering of clauses	2	19, 20
Prompted decisions	2	25, 26
Lesson 43: Recursive Functions		
Recursive functions	22	1, 2, 2.1, 3, 3.1 to 3.4, 4, 4.1, 5, 6, 6.1, 6.2, 7, 7.1 to 7.3, 8, 8.1 to 8.3
Prompted decisions	2	9, 10
Lesson 44: AND, OR, and NOT; Truth Tables		
AND, OR, and NOT	2	1, 2
Truth value of Boolean expressions	5	3, 4, 5, 6, 7
Conjunctions	4	8, 8.1, 8.2, 9
Disjunctions	2	10, 11
Order of evaluation	6	12, 13, 14, 15, 15.1, 15.2
Truth tables	4	16, 17, 18, 19
AND chains	3	20, 21, 22
Prompted decisions	2	23, 24
Lesson 45: TV(X) and the FIRST Function		
TV(X)	9	1, 1.1, 2, 3, 4, 4.1, 5, 6, 7
The function FIRST	11	8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 17.1
Prompted decisions	3	18, 19, 20

	Number of Exercises	List of Exercises
Lesson 46: LET and Boolean Expressions; Debugging Tools		
Using LET with Boolean expressions	6	1, 2, 3, 4, 5, 6
GO	5	7, 7.1, 8, 9, 9.1
DONE	9	10, 11, 11.1, 12, 13, 14, 15, 16, 16.1
Prompted decisions	3	17, 18, 19
Lesson 47: More Standard AID Functions		
DP(X)	11	1, 2, 3, 4, 5, 5.1, 7, 7.2, 10, 11, 12
XP(X)	7	4.1, 6, 6.1, 7.1, 7.3, 8, 9
Prompted decisions	3	13, 14, 15

References

Friend, J. Student manual, introduction to programming: AID. Institute for Mathematical Studies in the Social Sciences, Stanford University, 1969.

Friend, J., & Atkinson, R. C. Computer-assisted instruction in programming: AID. Technical Report No. 164, Institute for Mathematical Studies in the Social Sciences, Stanford University, January 25, 1971.

Friend, J. INSTRUCT coders' manual. Technical Report No. 172, Institute for Mathematical Studies in the Social Sciences, Stanford University, May 1, 1971.

Suppes, P., Goldberg, A., Kanz, G., Searle, B., & Stauffer, C. Teacher's handbook for CAI courses. Technical Report No. 178, Institute for Mathematical Studies in the Social Sciences, Stanford University, September 1, 1971.

Friend, J. E., Fletcher, J. D., & Atkinson, R. C. Student performance in computer-assisted instruction in programming. Technical Report No. 184, Institute for Mathematical Studies in the Social Sciences, Stanford University, May 10, 1972.

N74-26709

100 PROGRAMMING PROBLEMS

(With a Description of the Programming Language AID)

by

Jamesine E. Friend

September, 1973

Institute for Mathematical Studies in the Social Sciences
Stanford University
Stanford, California

269

Table of Contents

270

Introduction

The 100 problems in this booklet were prepared for students taking the computer-assisted course "Introduction to AID Programming," an introductory course in algebraic programming. Also included is a brief reference manual for the programming language AID.*

The problems include applications of elementary programming techniques to a variety of fields such as elementary arithmetic and algebra, geometry, linear algebra, probability and statistics, consumer and business problems, and calculus. Most of the problems require no mathematical background beyond high school algebra but some of the problems in linear algebra, statistics, and calculus will not be readily solved by students who do not have some acquaintance with fundamental concepts such as the sigma notation, solution of linear equations by determinants, limits of sequences and series, statistical correlation, and standard deviation. In all cases the necessary formulas are given, either within the problem or in the list of formulas on page 35.

These problems are not specific to the programming language AID but can also be used in introductory courses in BASIC, FORTRAN, ALGOL, APL, etc.

If any reader can suggest additional, useful problems to be included in a second edition of this booklet, the author would be most grateful.

*The subset of AID that is described herein includes that part of the language that is taught in the course "Introduction to AID Programming."

AID Commands and Programs

Numbers and Algebraic Expressions

Algebraic expressions in the programming language AID follow ordinary algebraic notation quite closely. The letters A, B, C,..., Z are used as variables, and the following symbols are used for arithmetic operations and grouping:

+	addition
-	subtraction
*	multiplication
/	division
↑	exponentiation
! !	absolute value
( )	parentheses

In forming algebraic expressions, juxtaposition cannot be used to indicate multiplication; the expressions 2x and xy must be written as 2*X and X*Y in AID notation. Algebraic expressions must be given as a linear string of symbols, which precludes the use of the horizontal bar as indicator of division; $\frac{a}{b}$ must be written as A/B, and $\frac{a+b}{a-b}$ must be written as (A+B)/(A-B). Neither can subscripts or superscripts be used; x_i is written as X(I) and y^2 is written as Y↑2.

Grouping is indicated with parentheses just as in ordinary algebraic notations, and parentheses may be imbedded as desired. If parentheses are not used, arithmetic operations are performed in this order:

↑	
* and /	from left to right
+ and -	from left to right

Thus, exponentiation is always done first (unless parentheses are used to indicate otherwise), then either * or /, and finally either + or -. If two operations with the same order of precedence appear, they are evaluated in left-to-right order; in the expression X/Y*Z/W, the first operation to be performed will be X/Y.

AID numbers may be written in integer form (275) or in decimal form (5.87, 0.01, .72). Numbers are limited to nine significant digits and must be less than 10^{100} in absolute value. Numbers may also be written in a form of scientific notation that is a direct translation of ordinary scientific notation. For example, 2.3076×10^5 is written as 2.3076 * 10↑5. Since the slash (/) is used to indicate division, an expression like 2/3 is read as "two divided by three" rather than "two thirds." Because of this, an expression like X↑2/3 means $x^2 \div 3$, not $x^{2/3}$; to write $x^{2/3}$ in AID notation, use X↑(2/3).

Negative numbers are indicated by a minus sign: -2.7. When negative numbers are used in certain combinations, such as 2 + (-3), the negative number must be enclosed in parentheses; to be on the safe side, always use parentheses around negative numbers.

The variables A, B, C,..., Z may be used for numbers, as indicated above. They may also be used as indexed (subscripted) variables to identify lists of numbers or arrays of numbers. The list $x_1, x_2, \ldots, x_n$ is written in AID notation as X(1), X(2),..., X(N), and the entire list is then referred to simply as X. A two-dimensional array (matrix) of numbers may be identified by a variable using two indices: a_{ij} is written in AID as A(I,J). Up to 10 indices may be used (for up to 10-dimensional arrays). Indices may be given as numbers, or variables, or algebraic expressions: X(12), X(N,J), and

X(2*I+3, J/4). Regardless of how the indices are indicated they must have integer values and are limited to -250 to 250, including zero. Thus, the longest list of numbers has 501 members: X(-250), X(-249),..., X(-1), X(0), X(1),..., X(249),X(250). A two-dimensional array could have 501 × 501 members, etc.

To summarize, here are some examples of algebraic expressions and their AID equivalents:

$5x^2 + 3y^4$	5*X↑2 + 3*Y↑4
$z^{1/2}$	Z↑(1/2) or Z↑0.5
$\|x-y\|$	!x-y!
$x_1 + x_2 - x_3$	X(1) + X(2) - X(3)
A_{ij}	A(I,J)
$\frac{a+b+c+d}{4}$	(A+B+C+D)/4

In general, spaces may be used whenever desired in algebraic expressions. The expression 5*X+4 may also be written 5*X + 4 or 5 * X + 4 or 5* X+4. The exceptions to this rule are in indexed variables and, as we shall see later, in function notation. Expressions like X(5) and A(1,2) must be written without a space between the identifier and the opening parenthesis; X (5) or A (1,2) will cause an error message.

274

The Form of AID Commands

AID commands are quite similar to English commands:

```
TYPE X

SET Y = 7

STOP
```

Each command begins with a verb (TYPE, SET, STOP) and the form of the rest of the command depends upon the verb that is used. The verb TYPE, for example, may be followed by any algebraic expression (and the result will be that the expression is evaluated and the value typed on the user's teletypewriter):

```
TYPE X - 2

TYPE 3/(4+2/7)

TYPE X↑2 + Y↑2
```

Some commands, like STOP, may consist of only one word, but most commands have either variables or algebraic expressions or equations or other kinds of arguments following the verb. Some commands also have optional modifiers, which are phrases that can be added to the command to modify its meaning. For example, the TYPE command may be modified by an IN FORM phrase:

```
TYPE X IN FORM 12
```

where Form 12 specifies the form in which X is to be typed. (This will be explained more fully below.)

With one exception (FORM), AID commands must be given in one line; a line is terminated by the user by typing the return key on the teletypewriter.

There are two kinds of AID commands: direct commands and indirect commands. Direct commands will be executed as soon as they are given, whereas indirect commands are stored and will not be executed until the user gives an order to do so. Many AID commands may be used as either direct or indirect commands. To indicate whether a command is to be a direct command or an indirect command, "step numbers" are used before indirect commands:

```
12.7  TYPE 15/16 + 1/32
```

This command will be stored rather than executed immediately, and the step number may be used in later references to the command. When the user wishes to have the command executed, he gives a DO command like the following:

```
DO STEP 12.7
```

Step numbers are decimal numbers between 1 and 10^9, and, like all numbers, are limited to 9 significant digits.

When indirect commands are stored, they are grouped into "parts" according to the integer portion of the step number. Commands numbered 23.2, 23.7, 23.84, and 23.001 are all grouped together into "Part 23." Indirect commands may be executed singly:

```
DO STEP 23.2
```

or they may be executed in groups:

```
DO PART 23
```

When the above command is given, all the steps in Part 23 will be executed, in numeric order. When Part 23 is exhausted, the execution will cease; even if there are steps numbered 24.1, 24.2, etc., execution will not automatically proceed to Part 24. A set of stored commands, to be executed as a group, is called a "program." A program may consist of a single part or, by the use of branching commands as explained below, several parts.

Although most AID commands can be used either as direct commands or indirect commands, there are a few that may be used only in one form. Table 1 lists the AID commands and shows which can be used directly and which indirectly.

Table 1

Direct and Indirect AID Commands

Command	May be used directly	May be used indirectly
DELETE	Yes	Yes*
DEMAND	No	Yes
DISCARD	Yes	Yes*
DO	Yes	Yes
FILE	Yes	Yes*
FORM	Yes	Yes
GO	Yes	No
LET	Yes	Yes
RECALL	Yes	Yes*
SET	Yes	Yes
SET (short version)	Yes	No
STOP	No	Yes
TO	No	Yes
TYPE	Yes	Yes
USE	Yes	Yes*

*Rarely used in the indirect form.

Basic Commands: SET, TYPE, DEMAND, TO, and DO

The five commands SET, TYPE, DEMAND, TO, and DO form the core of a basic AID vocabulary. Together with the algebraic expressions described above, a few standard AID functions, and the conditional clause described in the next section, these five commands are sufficient to solve any of the 100 problems given in this booklet.

The SET command is used to assign a value to a variable:

```
SET X = 12.7

SET K = 0.002305

SET M = K*X↑2
```

The algebraic expression used on the right of the equal sign may contain one or more other variables, but all of the variables used must have values so that the expression can be immediately evaluated. When a SET command is executed, the expression on the right of the equal sign is evaluated and that number is stored in temporary (core) storage with the specified identifier (the variable used on the left of the equal sign); that stored number may thereafter be referred to by its identifier. A SET command may be used to "define a variable in terms of itself." The result of the following sequence of commands would be that the number 7 is stored as N:

```
SET N = 13          sets N equal to 13.

SET N = N + 1       adds 1 to the current value of N.

SET N = N/2         divides the current value of N by 2.
```

SET may be used either indirectly (with a step number) or directly. If used as a direct command, the short form which omits the word SET may be used:

278

X = 7   equivalent to   SET X = 7

K = 0.07835   equivalent to   SET K = 0.078305

SET may also be used with indexed variables:

SET X(2,3) = 7   sets the element $X_{2,3}$ from the array X equal to 7

L(5) = 72.31   sets L_5 equal to 72.31

The TYPE command is used with an algebraic expression:

```
TYPE (X+K*Y)/3
```

Here again the algebraic expression must contain only variables that have values (or will be given values before the TYPE command is executed). When a TYPE command is executed, the value of the algebraic expression will be calculated and typed on the user's teletypewriter.

A TYPE command can be given with several arguments, separated by commas:

```
TYPE X,Y, (X+Y)/2
```

This command is equivalent to the three commands:

```
TYPE X

TYPE Y

TYPE (X+Y)/2
```

Caution: Only two commands, TYPE and DELETE, allow multiple arguments; other commands, like SET and DO, use only one argument.

The TYPE command can be used to type text by giving the text enclosed in quotation marks:

```
TYPE "TITLE: COMPOUND INTEREST CALCULATIONS"
```

Other uses of the TYPE command will be described later.

The DEMAND command can only be used indirectly (as a stored command):

```
20.4  DEMAND X
```

The DEMAND command uses a single variable as an argument, and the result of such a command is to cause the program to halt, type

```
X=
```

wait for the user to type a value for X, and then continue the execution of the program. By using DEMAND commands, a program can be written so as to ask for the data it needs. A useful variant of the DEMAND command is formed by appending the modifying phrase AS "text." The command

```
17.9  DEMAND R AS "INTEREST RATE"
```

will cause the program to stop at Step 17.9, type

```
INTEREST RATE=
```

and wait for the user to type a value which will be assigned the identifier R.

A feature of the DEMAND command that is frequently useful in iterated programs is that if the user refuses to give a value for the DEMANDed variable, and responds simply by typing the return key, the execution of the program will halt at that point; thus, seemingly endless loops can be used if they incorporate DEMANDs.

DEMAND is used solely for input, SET is used for both input and for internal computations, and TYPE is used for both computation and output. Here is an example of a complete program using all three of these commands:

```
4.1  TYPE "COMPUTATION OF INTEREST AT 4.5%"
4.2  SET R = 0.045
4.3  DEMAND P AS "PRINCIPAL"
4.4  SET I = R * P
4.5  SET T = P + I
4.6  TYPE I,T
```

This program would be executed by the command

```
        DO PART 4
```

and it would start by typing

```
        COMPUTATION OF INTEREST AT 4.5%

        PRINCIPAL =
```

As soon as the user typed a value for P, say 200, the program would reply

```
        I = 9

        T = 209
```

As mentioned, the steps within a part are ordinarily executed in numeric order. This order can be overridden by the use of the branching command, TO. TO, like DEMAND, can be used only as an indirect command. A TO command may be used to branch to either another step (within the same part or in some other part) or to another part:

6.3 TO STEP 7.29	will cause execution of Part 6 to cease and execution of Part 7 to commence at Step 7.29.
16.42 TO PART 8	will cause execution of Part 16 to cease and execution of Part 8 to commence at the lowest numbered step.

Although a TO command may be used unconditionally, as shown above, simply to alter the linear sequence of execution, it is more often used conditionally, that is, with an IF clause, as will be explained in the next section.

Several examples of direct DO commands have been given above. Used directly, DO causes the execution of a specified step or part:

```
        DO STEP 7.35

        DO PART 84
```

DO may also be used indirectly, as part of a program, to cause the execution of another part as a subroutine:

281

```
7.1  SET P = 3.14159

7.2  SET R = 15

7.3  DO PART 12

7.4  TYPE D, C, A
```

In this program Step 7.3 calls for the execution of Part 12. Part 12 is the "subroutine" and the DO command in Step 7.3 is the "subroutine call." When Part 7 is executed, the sequence of execution is:

Step 7.1

Step 7.2

Step 7.3

All of Part 12

Step 7.4

Thus, DO as well as TO can be used to override the automatic linear sequence of execution. The primary difference is that DO calls for another step or part to be inserted into the part being executed, whereas TO calls for a complete transfer of control to the part specified. Here are four sample commands, with comments, to summarize the difference between DO and TO.

3.6 DO PART 7 will cause all of Part 7 to be executed, followed by the execution of the remainder of Part 3.

3.6 TO PART 7 will cause all of Part 7 to be executed. Execution will halt at the end of Part 7. The remainder of Part 3 will not be executed automatically.

3.6 DO STEP 7.5 will cause Step 7.5 to be inserted as a one-step subroutine. After Step 7.5 is done, the remainder of Part 3 will be executed. No other steps in Part 7 will be done.

3.6 TO STEP 7.5 will cause execution of Part 7 to start at Step 7.5. Execution will halt at the end of Part 7, and the remainder of Part 3 will not be executed automatically.

There are two modifiers that may be used with DO commands: TIMES and FOR. The TIMES modifier is used to specify the number of times the required step or part will be executed:

```
DO STEP 3.5, 6 TIMES

13.2 DO PART 12, N TIMES
```

The number of times a step or part is to be iterated may be specified by a number or a variable, or even an algebraic expression, with the stipulation that the value is a positive integer.

The second modifier, the FOR clause, specifies values for some variable:

```
DO PART 4 FOR X = 7
```

This command is equivalent to the two commands

```
SET X = 7

DO PART 4
```

A list of values may be given in the FOR clause if desired:

```
DO PART 4 FOR X = 7, 23.8, 19
```

This command will cause Part 4 to be done three times, once for each of the listed values for X, and is thus equivalent to the six commands

```
SET X = 7

DO PART 4

SET X = 23.8

DO PART 4

SET X = 19

DO PART 4
```

The values for the variable may be given in the form of a "range specification," as in this example:

```
DO PART 21 FOR A = 5(2)13
```

The range specification 5(2)13 indicates that the initial value of A is to be 5 and that A is to be incremented by 2 with each successive iteration until the value of 13 is reached. That is, A will take on the values 5, 7, 9, 11, and 13. Any or all of the initial value, the size of the increment, and the final value may be given as algebraic expressions, and they need not be integral. The command

```
DO STEP 7.3 FOR Y = 3.2(.2)4
```

is equivalent to

```
DO STEP 7.3 FOR Y = 3.2, 3.4, 3.6, 3.8, 4
```

When values of a variable are given in a range specification, the final value is always used. Hence, the command

```
DO PART 2 FOR X = 0(2)7
```

will cause these values of X to be used: 0, 2, 4, 6, 7.

DO commands with either TIMES or FOR modifiers may, of course, be used as indirect steps to cause iterated execution of a subroutine.

The IF Clause

Certain modifiers, such as the AS or TIMES phrases, may be used to modify specific commands. There is one modifier that may be used with any AID command, and that is the IF clause. The addition of an IF clause changes any command from an "unconditional command" to a "conditional command." Here are a few examples:

```
TYPE X/Y IF Y > 0
3.2 DEMAND R IF T = A + X
7.3 DO PART 8, 3 TIMES IF X <= Y + 3
SET Z = X/(Q + S) IF Q + S > X
```

An IF clause contains the word IF followed by a Boolean expression. Boolean expressions (also called logical predicates) express relationships between numbers. The following relational symbols are used:

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
=	equal
#	not equal

As in ordinary usage, any algebraic expressions may be used in Boolean expressions:

```
X < 0
X + Y ↑ 2 # Z
2 >= Z
```

The Boolean operators AND, OR, and NOT may also be used:

```
NOT X < 0
X < 7  AND  Y > 8
X > 0  OR  X < Y - 2
X # 0  OR  Y # 0  OR  Z # 0
(A + B > 0  OR  A < 7)  AND  B > = 12
```

In evaluating Boolean expressions, the Boolean operators are evaluated in this order (unless there are parentheses to indicate otherwise):

```
NOT
AND
OR
```

When a conditional command is executed, the execution proceeds in two phases. First, the Boolean expression used in the IF clause is evaluated to determine whether it is true or false. Second, if the Boolean expression is true, the main clause will be executed.

Any command may be modified by an IF clause. One of the most important uses of the IF clause is in TO commands; a conditional TO command is called a "conditional branch" and is the principal mechanism used in writing non-linear programs, including those with loops. As an example, here is a simple program with a loop (this program simply counts from 0 to 30 by twos):

```
5.1  SET C = 0
5.2  TYPE C
5.3  SET C = C + 2
5.4  TO STEP 5.2 IF C < = 30
5.5  TYPE "THAT'S ALL."
```

Auxiliary Commands: FORM, LET, and DELETE

Besides the five commands (SET, TYPE, DEMAND, TO, and DO) that are used in writing simple programs, there are a number of auxiliary commands that are ordinarily used as direct commands. Two of these, FORM and LET, are to define forms and functions that will be used by TYPE and SET commands in programs, and are thus closely associated with the programs themselves. The other auxiliary commands are used more for bookkeeping or debugging purposes; these are DELETE, the file commands to be discussed in the following section, and the debugging commands to be discussed in the section after that.

FORM and LET are used in conjunction with stored programs. FORM is used to specify the format to be used for output. Ordinarily, when a TYPE command is used, the output is printed in a standard form. For example, when the command

```
          TYPE (X + 2)/Y
```

is given, the value will be typed in this form:

```
          (X + 2)/Y = 28.7
```

If a number is 10^6 or greater or if it is less than .001, it will be typed in scientific notation rather than decimal form:

```
          (X + 2)/Y = 2.87 * 10↑(-4)
          (X + 2)/Y = 2.87 * 10↑8
```

If the user prefers another form for output, he may specify it in a FORM statement. The FORM statement, unlike other AID commands, requires two lines; the first line specifies the form number (an integer between 1 and 10^9 to be used in later references) and the second line specifies the form itself:

```
FORM 12:

THE INTEREST IS ←←←.←←
```

The location of digits is indicated by the character ← and the position of the decimal point is shown by a period. When the form specified above is to be used, the TYPE command is modified by an IN FORM phrase:

```
TYPE P * R IN FORM 12
```

Numbers will be rounded to fit the specified form (which is the easiest way of rounding numbers to a fixed number of decimal places) and if no decimal point is specified, the number will be rounded to the nearest integer. When specifying a form, care must be taken to allow for as many digits before the decimal point as will be necessary; if an attempt is made to type a number in a form that is not large enough, an error message will result. If the number to be typed in a given form is negative, one of the digit locations will be taken up by the negative sign.

Any symbols, including punctuation marks, may be used in the text of a form:

```
FORM 42:

PRINCIPAL + INTEREST = $ ←←←.←←
```

No text is necessary if the user wishes merely to print a number in a given form and location.

More than one number may be provided for, and this is the only way in which more than one number can be printed on the same line:

```
FORM 6:

$ ←←←←.←← WILL EARN $ ←←←.←← INTEREST
```

To use a form with several numbers, the multiple-argument form of the TYPE command is required:

```
TYPE P, P * R IN FORM 6
```

The LET command is also used in conjunction with stored programs, but may be used independently for direct computations. The primary use of LET is in the definition of functions. The function $f(x) = 3x^2 + 2x$ is defined in AID as follows:

```
LET F(X) = 3*X↑2 + 2*X
```

When the function is used, in a SET or TYPE command, a value is substituted for the dummy variable X in the expression F(X):

```
SET Y = F(3)

TYPE F(5) - F(3.7)
```

The value that is substituted may be in the form of an algebraic expression, provided such an expression can be immediately evaluated:

```
SET N = 2

TYPE F(N/6)
```

Any of the variables A, B, C,..., Z may be used as function names. Take care, however, not to use the same identifier for both a real variable and a function since the first definition will be replaced by the second.

Functions of up to ten variables may be defined; here is an example of a function of three variables:

```
LET F(X, Y, Z) = (X*Y + Y*Z)/X*Y*Z
```

Caution: Do not use a space between the function name and the opening parentheses; an expression like F (3) will cause an error message.

A useful variant of the LET command is the conditional form of LET used to define functions conditionally. In ordinary notation, a function may sometimes be defined in this fashion:

$$f(x) = \begin{cases} -2x \text{ if } x < 0 \\ 5x \text{ if } x \geq 0 \end{cases}$$

In AID, this definition is given in a single line:

LET F(X) = (X < 0: -2*X; X > = 0: 5*X)

which is read "If $x < 0$, $f(x) = -2x$; if $x \geq 0$, $f(x) = 5x$." In the AID definition, the entire expression is enclosed in parentheses, the clauses within the definition are separated by semicolons, and each clause is divided into a condition and an algebraic expression separated from one another by a colon. Any number of clauses may be used; in the above example, there are two clauses.

If the definition of a function is given in ordinary terms with an "otherwise" clause,

$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ 2x \text{ if } x > = 0 \text{ and } x < 7 \\ 5x \text{ otherwise} \end{cases}$$

the AID definition does not require a condition in the final clause:

LET F(X) = (X < 0: 0; X > = 0 AND X < 7: 2*X; 5*X)

In this example, the final clause consists only of the algebraic expression 5*X, which will be used whenever all of the conditions in preceding clauses fail.

When a function definition is used, it is scanned from left to right until a condition that holds is found. Because of this, it is frequently possible to simplify AID definitions. For example, the condition in the second clause of the above example could be simplified from X > = 0 AND X < 7 to X < 7:

LET F(X) = (X < 0: 0; X < 7: 2*X; 5*X)

A function may call itself; hence, a variant of the conditional definition is definition by recursion. Here, for example, is the AID recursive definition of the factorial function X!

```
LET F(X) = (X = 1: 1; X*F(X-1))
```

Both LET and FORM serve to store information in core storage. In the one case a function definition is stored and in the other the definition of an output form. SET and DEMAND also use core storage; both of these cause a number and its identifier to be stored. Stored commands (indirect steps) are also put into core storage, as clued by the step number preceding the command. In programming it is often necessary to inspect the information that is being held in core or to delete some items.The contents of core can be displayed by using TYPE commands and deleted by means of DELETE commands. Some example of such TYPE and DELETE commands are given here, with comments:

TYPE X   will print the value of X if X is a number or a list or array, or the definition of X if X is a function.

DELETE X   will delete either a number X or a function X.

TYPE X(3)   will print the value of X_3.

DELETE X(3)   will delete the single value X_3 from the list X.

TYPE FORM 3   will type the definition of Form 3.

DELETE FORM 3   will delete the definition of Form 3.

TYPE STEP 7.1   will print the stored command identified as Step 7.1.

DELETE STEP 7.1   will delete Step 7.1.

TYPE PART 29   will print all of the steps in Part 29 in numeric order.

DELETE PART 29   will delete all of the steps in Part 29.

```
TYPE ALL           will print the entire contents of core.

DELETE ALL         will delete everything in core storage.

TYPE ALL VALUES    will print all numbers, lists, and arrays.

TYPE ALL FORMULAS will print all function definitions.

TYPE ALL STEPS

TYPE ALL PARTS

TYPE ALL FORMS

DELETE ALL VALUES

DELETE ALL FORMULAS

DELETE ALL STEPS

DELETE ALL PARTS

DELETE ALL FORMS
```

Both TYPE and DELETE may be used with several arguments, separated by commas:

```
TYPE X, STEP 3.7, F

DELETE STEP 3.7, PART 9, K, F
```

These are the only two AID commands that have multiple-argument forms.

File Commands: USE, FILE, RECALL, and DISCARD

Anything that is stored in core will be automatically deleted whenever the user signs off. Any or all of this information can be copied to more permanent storage space on the disk. To do this, the file commands USE, FILE, RECALL, and DISCARD are used. AID files are variable length disk files, identified by integers from 1 to 2750. The files need not be used in numeric order and the user specifies which file he wants to use by giving a command like

```
USE FILE 100
```

The file number is held in core until another USE command is given (or until the user signs off) and all subsequent FILE, RECALL, and DISCARD commands will refer to this file.

Each file is divided into "items," numbered from 1 to 25, and the user must specify the item when storing or retrieving information. Items need not be used in numeric order. To file an item, a command like

```
FILE PART 7 AS ITEM 3
```

is given. The user may file a form, a step, a part, a value, a function definition, or all of these, using commands similar to the TYPE and DELETE command shown just above. The entire contents of core may be stored as a single item by giving a command like

```
FILE ALL AS ITEM 17
```

When information is filed on the disk, the contents of core are not disturbed; a copy is made for transfer to the disk.

When the user wishes to retrieve information from the file, he uses a command like

```
RECALL ITEM 17
```

and when he wishes to discard an item from the file, he uses a command like

```
DISCARD ITEM 17
```

Debugging Commands

The commands STOP and GO are used primarily for debugging purposes. STOP is inserted as a temporary command, to be removed when debugging is complete, and may be used either conditionally or unconditionally to halt the execution of the program at the point where the STOP command is encountered:

```
47.3 STOP
47.352 STOP IF N > 100
```

While the program is STOPped, the user may inspect or alter the contents of core, checking current values of variables used by the program; replacing, inserting, or deleting steps in the program, etc. To resume execution the user gives the direct command

```
GO
```

During the time the program is STOPped, the user may not execute another step or part (that is, he cannot give another direct DO command), at least not if he wishes to resume the execution of the STOPped program at a later time.

GO may also be used to restart the execution of a program that was halted because of a syntax error. After the program stops, and the error message is printed, the user may correct the error and then resume execution from that point by giving a direct GO command.

Temporary TYPE commands may also be used for debugging purposes. These are commands like

```
32.105 TYPE X, Y, K, N
```

that are inserted temporarily so that the values of variables will be typed for inspection. When debugging is complete, these commands, and temporary STOP commands, are removed by giving DELETE commands:

```
        DELETE STEP 47.3, STEP 32.105
```

Summary of AID Commands

The following summary of AID commands is given in the form of examples, with comments. Commands that are ordinarily used directly are shown without step numbers and those that are ordinarily used indirectly are shown with step numbers; to find out which commands must be used directly (or indirectly) refer to Table 1.

Most of the examples are shown as unconditional commands; however, any command may be used conditionally (modified by an IF clause) if desired.

DELETE X	deletes the identifier X and its value.
DELETE F	deletes the definition of the function F.
DELETE A(2,3)	deletes the element $A_{2,3}$ from the array A.
DELETE STEP 7.1	deletes Step 7.1.
DELETE PART 7	deletes all steps in Part 7.
DELETE FORM 22	deletes the definition of Form 22.
DELETE K, STEP 4.3, STEP 4.4	deletes the three specified items.
DELETE ALL VALUES	deletes all real variables and their values.
DELETE ALL STEPS	etc.
DELETE ALL PARTS	
DELETE ALL FORMS	
DELETE ALL	

7.1 DEMAND M	requests a value for the real variable M.
2.05 DEMAND A(2,3)	requests a value for the element $A_{2,3}$ of the array A.
3.7 DEMAND X(I,J,K)	requests a value for the element $X_{i,j,k}$ of the three-dimensional array X.
16.4 DEMAND X AS "RADIUS"	requests a value for X by typing RADIUS =

DISCARD ITEM 20	discardsItem 20 from the previously designated disk file (see USE).

DO STEP 6.2	executes Step 6.2.
DO PART 9	executes the steps in Part 9 in numeric order.
DO PART 12, 7 TIMES	executes Part 12, 7 times.
DO PART 4 FOR X = 2, 7, 4.3	executes Part 4, 3 times, once with X = 2, once with X = 7, and once with X = 4.3.
7.2 DO PART 6, N TIMES	executes Part 6 (as a subroutine), N times.
62.15 DO STEP 32.3 FOR A = 5(2)12	executes Step 32.3 once for each of these values of A: 5, 7, 9, 11, 12.

FILE X AS ITEM 2	files the identifier X and its value as Item 2 of the previously designated disk file (see USE).
FILE A(7,3) AS ITEM 6	
FILE FORM 3 AS ITEM 12	
FILE STEP 6.25 AS ITEM 4	
FILE PART 9 AS ITEM 1	
FILE ALL STEPS AS ITEM 5	
FILE ALL PARTS AS ITEM 21	
FILE ALL FORMS AS ITEM 7	
FILE ALL VALUES AS ITEM 14	
FILE ALL AS ITEM 3	

(Note: The item number must be an integer from 1 to 25.)

FORM 7:

THE LENGTH IS ←←← INCHES MORE THAN THE WIDTH.

defines an output form with allowance for one value (see TYPE...IN FORM...).

FORM 13:

←←.← ←←.← ←←.←

defines an output form with allowance for three values, but no text.

FORM 2:

THE COST OF ←← ITEMS IS $ ←←←.←←

defines an output form with allowance for two values. The first value will be rounded to the nearest integer, and the second value will be rounded to two decimal places.

(Note: The form number must be a positive integer less than 10^9.)

GO

continues the execution of a program halted by a STOP command or by a syntax error.

LET F(X) = 3*X↑5 - 7

defines the function $f(x) = 3x^5 - 7$.

LET V(R,H) = 3.14159265*R↑2*H

defines the function $V(r,h) = \pi r^2 h$ (functions of up to 10 variables may be defined).

LET F(X) = (X < 0: X↑2 + 5; X > = 0: X + 5)

defines the function

$$f(x) = \begin{cases} x^2 + 5 \text{ if } x < 0 \\ x \ \ + 5 \text{ if } x \geq 0 \end{cases}$$

LET F(X) = (X = 1: 1; X + F(X-1))

defines the recursive function

$$f(x) = \begin{cases} 1 \text{ if } x = 1 \\ x + f(x-1) \text{ if } x > 1 \end{cases}$$

RECALL ITEM 7	recalls Item 7 from the previous designated disk file (see USE).

SET P = 3.14159265	assigns the value 3.14159265 to the identifier P.
6.35 SET A(5, 7) = 12.31	assigns the value 12.31 to the element $A_{5,7}$ in the array A.
7.3 SET N = N + 1	increases the current value of N by 1.
X = 4.3	short form of the SET command, equivalent to SET X = 4.3
L(7) = 2769	short form of the SET command, equivalent to SET L(7) = 2769

7.3 STOP	causes the program to stop execution of Step 7.3 (see GO).
26.64 STOP IF N > M + 1	causes the execution of the program to stop at Step 26.64 if N > M + 1.

31.3 TO STEP 31.1 IF N < 100	causes a branch to Step 31.1 if N < 100
8.25 TO PART 9	causes an unconditional branch to Part 9.

TYPE X↑Y	evaluates x^y and types the result.
7.3 TYPE X, F(X)	types the values of X and F(X).
12.9 TYPE "TAX COMPUTATIONS"	types an exact copy of the text enclosed in quotation marks.
TYPE FORM 2	types the definition of Form 2.
TYPE STEP 3.7	types the command stored as Step 3.7.
TYPE PART 5	types all of the commands in Part 5.

TYPE ALL STEPS

TYPE ALL PARTS

TYPE ALL FORMS

TYPE ALL VALUES

TYPE ALL

3.8 TYPE 5*X IN FORM 2 — evaluates 5x and types the result in the specified output form (see FORM).

USE FILE 100 — designates the disk file to be used by subsequent FILE, RECALL, and DISCARD commands.

(Note: The file number must be a positive integer from 1 to 2750.)

300

AID Functions

In addition to the functions that may be defined by the user by means of LET commands, there are a number of useful standard AID functions. There are two trigonometric functions, SIN(X) and COS(X); X is in radians and must have an absolute value less than 100. The natural logarithm function LOG(X) yields the logarithm to the base e of x, where x is any positive real number. The inverse of the LOG function is the exponential function EXP(X), equivalent to e^x.

Several functions depend upon features of the decimal representation or scientific notation of the argument:

IP(X), the "integer part" function, yields the integer portion of the decimal representation of the number x. For example, IP(7304.56) = 7304.

FP(X), the "fraction part" function, yields the fractional portion of the decimal representation of the number x. FP(7304.56) = .56.

DP(X), the "digit part" function, yields the digital part of the scientific notation of x. For example, DP(3789.54) = 3.78954 since the scientific notation for x is 3.78954×10^3.

XP(X), the "exponent part" function, yields the exponent part of the scientific notation. For example, XP(3789.54) = 3 since 3 is used as the exponent of 10 in the representation 3.78954×10^3.

Two other real functions that are occasionally used are SGN(X), the "sign" function, and SQRT(X), the "square root" function. These are defined as follows:

$$\text{SGN(X)} = \begin{cases} 1 \text{ if x is positive} \\ 0 \text{ if x is zero} \\ -1 \text{ if x is negative} \end{cases}$$

SQRT(X) = $\sqrt{X}$

There are four functions on lists of real numbers: MAX, MIN, SUM, and PROD. The forms of these are similar, and the resulting values are, respectively, the maximum of the specified list, the minimum, the sum of the numbers in the list, and the product. Each of these four functions may be used by simply listing the members of the argument:

MIN(.69, 2/3, .63) has a value of .63

SUM(2, 15, 0, 4) has a value of 21

The list of numbers to be used as an argument may be given by specifying a formula and the values of the dummy variable used in the formula:

SUM(I = 2, 10, 3: I * 5) is equivalent to

SUM(2 * 5, 10 * 5, 3 * 5).

The values of the variable may be given in a range specification:

SUM(I = 5(1)10: 3/I-7)

This expression is equivalent to

$$\sum_{i=5}^{10} \left(\frac{3}{i} - 7\right)$$

Similarly, the expression

PROD(J = 0(2)6: J↑2)

is equivalent to

$$\prod_{j=0,2,4,6} (j^2)$$

The function FIRST is a function on an indexed list of Boolean expressions. For a specified list of Boolean expressions, the FIRST function will yield the index of the first true expression. That is, it will find the location of the first true predicate. The form of the FIRST function is shown in this example:

FIRST(I = 1(1)50: I > 6↑2 + 3)

The value of this expression will be the first value of i in the set $\{1, 2, 3, \ldots, 50\}$ such that $i > 6^2 + 3$ (that value is 40).

Another simpler function on Boolean expressions is the function TV(X) which yields either 1 or 0 depending upon whether the Boolean expression X is true or false. For example, the value of TV(2 < 1 OR 5 > 4) is 1.

For all of the standard AID functions, the values are real numbers; hence, these functions can be used anywhere in algebraic expressions just as in ordinary algebraic notation. They may also be combined and composed in the usual ways. Here are a few examples of algebraic expressions in ordinary notation and in AID notation:

$\frac{\sin x}{\cos x}$	SIN(X)/COS(X)
$\sin^2 x$	(SIN(X))↑2
$\ln x$	LOG(X)
e^{2x}	EXP(2*X)
$\sqrt{x^2 + y^2}$	SQRT(X↑2 + Y↑2)

Formulas

1 kilometer = 0.621 miles

1 pound = 16 ounces

Diameter of a circle of radius r: 2r

Circumference of a circle of radius r: $2\pi r$

Area of a circle of radius r: πr^2

Area of a triangle of height h and base b: $\frac{hb}{2}$

Volume of a cylinder of height h and radius r: $\pi r^2 h$

Volume of a cone of height h and radius r: $\frac{\pi r^2 h}{3}$

Volume of a rectangular pyramid of height h and base w by ℓ: $\frac{hw\ell}{3}$

Volume of a rectangular prism of height h and base w by ℓ: $hw\ell$

Pythagorean Theorem: A given triangle is a right triangle if and only if the square of the length of the hypotenuse is equal to the sum of the squares of the lengths of the other two sides.

Distance between points in a plane: The distance from (x_1, x_2) to (y_1, y_2) is $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$

Distance between points in space: The distance from (x_1, x_2, x_3) to (y_1, y_2, y_3) is $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}$

Compound interest: The amount of money that will accumulate in n years if the amount P is invested at an interest rate r, compounded s times per year is

$$A = P\left(1 + \frac{r}{s}\right)^{ns}$$

Mean: The mean of a set $\{x_i\}$ of n numbers is

$$M = \frac{\sum_{i=1}^{n} x_i}{n}$$

Standard deviation: The standard deviation of a set $\{x_i\}$ of n numbers is

$$S_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - M)^2}{n}}$$

where M is the mean of $\{x_i\}$.

Standard score: For a score x_i from a set of scores $\{x_i\}$ the standard score is $z_{x_i} = \frac{x_i - M}{S_x}$ where M is the mean of $\{x_i\}$ and S_x is the standard deviation.

Correlation coefficient: The correlation coefficient for a set $\{(x_i, y_i)\}$ is

$$r_{xy} = \frac{\sum_{i=1}^{n} z_{x_i} z_{y_i}}{n}$$

where z_{x_i} is the standard score for x_i, z_{y_i} is the standard score for y_i, and n is the number of points in the set $\{(x_i, y_i)\}$.

Sample standard error of estimate for predicting standard scores:

$S = \sqrt{1 - r_{xy}^2}$ where r_{xy} is the correlation coefficient.

Regression equation for prediction of y from x:

Predicted value of $y = r_{xy} \frac{S_y}{S_x} (x - M_x) + M_y$

where M_x is the mean of $\{x_i\}$, M_y is the mean of $\{y_i\}$, S_y and S_x are the corresponding standard deviations, and r_{xy} is the correlation coefficient.

Programming Problems

Applications in Algebra

1. Convert miles to kilometers.

2. Convert miles per hour to feet per second.

3. Convert inches to feet and inches.

4. Compute

   (a) $\sqrt{x}$

   (b) $3x^2 - 2x + 5$

   (c) $x + xy + y$

   (d) $\frac{x + y}{x - y}$

   (e) $\sqrt{x^2 + y^2}$

   (f) $x^{1/3} + y^{1/5}$

5. Compute the average speed of a car that traveled d miles in h hours and m minutes.

6. Round a number x to the nth decimal place.

   Examples: 137.45702 rounded to the 2nd decimal place is 137.46.

   0.0273 rounded to the 2nd decimal place is 0.03.

7. Find the number of digits in a number x. Find the number of digits before the decimal point, and the number of digits after the decimal point.

8. Round a number x to the nth significant digit. n may be any integer between 1 and 9.

   Examples: 137.45702 rounded to 2 significant digits is 140.

   0.0273 rounded to 2 significant digits is 0.027.

9. Given 3 coins, two of which are equal in weight, determine which coin is different in weight and whether it is heavier or lighter. (Assume that when the program is used, it will be given the exact weight for each of the three coins.)

10. Find the age in years, months, and days of a person born on any given date. (If you can't give a precise solution, you may simplify the problem by assuming that every year has 365 days or that all months are the same length.)

11. The equation

$$x^5 - 48x^4 + 730x^3 - 3790x^2 - 2200x - 10969 = 0$$

has a solution somewhere between $x = 0$ and $x = 100$. Find one solution to the third decimal place.

Hint: The value of the polynomial is negative when $x = 0$ and positive when $x = 100$. Narrow down the interval where the polynomial changes sign by using a "binary search" pattern: First, find out if the polynomial changes sign between $x = 0$ and $x = 50$, or between $x = 50$ and $x = 100$. Then cut that interval in half and find out in which half the polynomial changes sign, etc.

Applications in Geometry

12. Given the radius r of a circle, compute the diameter, circumference, and area.

13. Compute the surface area of a box.

14. Write a general "volume calculating" program that will first ask which of these shapes is desired:

    1. Cylinder

    2. Cone

3. Rectangular pyramid

4. Rectangular prism

and will then ask for the appropriate dimensions and perform the calculation.

15. Suppose x and y are two points on a number line. Compute the distance between them.

16. Given two points, x and y, on a number line, determine which point is closer to the fixed point P = 2/3.

17. Given three numbers, a, b, and c, determine which, if any, is between the other two.

18. Find the length of a line segment whose end points are (x_1, x_2) and (y_1, y_2).

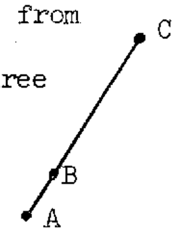
19. Suppose two points in space are given: (x_1, x_2, x_3) and (y_1, y_2, y_3). Find the distance between the two points.

20. Given two points in space, (x_1, x_2, x_3) and (y_1, y_2, y_3), find out which point is closest to the origin.

21. Given three points in the plane, (x_1, x_2), (y_1, y_2), and (z_1, z_2), find out which two of the three points are closest to one another.

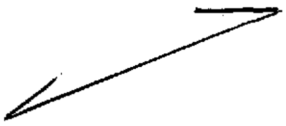
22. Determine whether or not three given points in a plane are collinear.

Hint: If the distance from A to B plus the distance from B to C is equal to the distance from A to C, the three points are on the same line (are collinear).



23. Given three numbers, find out if the numbers determine a triangle.

Hint: If there were three line segments of the given lengths, could they be placed so as to form a triangle? These can't:



24. Given three numbers, find out if the three numbers determine a right triangle.

Hint: Use the Pythagorean Theorem.

Data Storage

25. Write a program that will store a list of numbers, to be typed by the person using the program. The list may be of any length from 1 to 250. After the list is stored, the program should print it so it can be proofread.

26. Find the sum of the numbers in a list and put that sum into the list as an additional member.

Example: Old list: 2, 7, 3, 0, 1

New list: 2, 7, 3, 0, 1, 13

27. Find the smallest (largest) number in a list of numbers. Print both the number and its location in the list.

28. Given a number x and a list L, determine whether or not the number is in the list.

29. Find the locations of all numbers between x and y in a list L.

30. Find out what percentage of the numbers in a list L are greater than a given number x.

31. For a list L, form a new list S of the subtotals of the numbers in L, that is

$S_1 = L_1$

$S_2 = L_1 + L_2$

$S_3 = L_1 + L_2 + L_3$

etc.

32. Given a list L, make up a new list P of all of the positive numbers in L, in the same order in which they occur in L.

33. Write a program that will store an n × m array A. The elements in A are to be typed by the person using the program. The program should print the array so it can be proofread (if the array has 7 or fewer columns, it can be printed in table form for ease of reading).

34. Determine whether or not a number x occurs in the first column of an array A.

35. Given an n × m array A, determine which row and column, if any, contains the number x.

36. Given an array A and a number x, determine whether or not x occurs in the first column of A. If it does, print the entire row in which x occurs.

37. Find the sum of the numbers in each row of an array, and add those sums to the array as an additional column.

Example:

Old array		New array		
2	3	2	3	5
4	2	4	2	6
5	4	5	4	9

38. Determine which row in an array A has the largest (smallest) sum.

39. For a list L of non-negative numbers, find the smallest number that is greater than zero. Print both the number and its location.

40. Suppose L is a list of 10 positive numbers. Form a new list N by sorting the numbers from L into numeric order.

    Hint: Find the smallest number in L (other than zero), put that number into the first place in N, and replace it in L with a zero. Then, find the smallest number in L (other than zero), put that number into the next available place in N, and replace it in L with a zero. Repeat until L is exhausted.

41. Suppose L is a list of 10 numbers which may be negative, positive, or zero. Form a new list N by sorting the numbers from L into numeric order.

    Hint: Find the largest number in L and use that to replace each number that is moved from L to N.

42. Suppose you are given an ordered list L. (If the list is not already ordered, use the program for the last problem to sort it into numeric order.) Write a program that will insert a new element x into the list in the proper place.

    Hint: Start at the last number in the list and move each number one place farther out, working backward until you reach the place where x belongs.

43. Given an n × m array A, form a new array S that is the same as A with the first column sorted into numeric order.

311

44. Given an array, sort the rows into numeric order. Sort first by the number in the first column; then, if two rows have the same first element, sort according to the value in the second column, etc.

Example:

Old array			New array		
2	7	3	1	9	4
1	9	4	2	7	1
2	8	5	2	7	3
2	7	1	2	8	5

Business and Consumer Applications

45. A store owner buys items at a wholesale price and marks them up a certain percentage to the retail price. Calculate the total cost of i items at a wholesale price p, and print the retail price if the markup is 28%.

46. Small eggs weigh 18 ounces per dozen, medium weight 21 ounces, large 24, and extra large 27. If the price of eggs is s cents per dozen for small eggs, m for medium, ℓ for large, and e for extra large, determine which is the best buy and give the cost per pound of that size. Also, make up a table showing what the competitive price should be for medium, large, and extra large eggs if the price of small eggs is 20¢, 25¢, 30¢, ..., 90¢.

47. Laundry detergent of a certain brand comes in three size containers:

Regular	1 lb. 4 oz.
Giant	3 lbs. 1 oz.
King	5 lbs. 4 oz.

312

If the prices are r, g, and k, respectively, which size is the best buy? If a family uses 40 pounds of detergent per year, what would be the saving over the other two sizes per year? (In testing the program, reasonable values to use for r, g, and k are: r = 37¢, g = 85¢, k = $1.51.)

48. A sales tax of 3% is calculated as follows:

For a sale of 1¢ to 14¢, the tax is 0.

For a sale of 15¢ to 44¢, the tax is 1¢.

For a sale of 45¢ to 74¢, the tax is 2¢.

For a sale of 75¢ to $1.00, the tax is 3¢.

For a sale in even dollars, the tax is 3¢ per dollar.

Write a program that will calculate total price, including sales tax.

49. A wholesale lumber dealer sells lumber by the board foot. A board foot is a measure of volume, equivalent to a one-foot length of 1" × 12". Compute the total number of board feet in

n_1 pieces of 2" × 4" of length ℓ_1

n_2 pieces of 2" × 4" of length ℓ_2

n_3 pieces of 2" × 6" of length ℓ_3

n_4 pieces of 1" × 8" of length ℓ_4

50. First Federal Savings Company charges 8% interest on home loans. For a 30-year mortgage of $20,000, the monthly payment is $146.76. How much must be paid to the Savings Company over the 30-year period?

51. First class postage costs 8¢ per ounce or fraction of an ounce for 0 to 12 ounces. From 12 ounces to one pound, there is a flat fee of $1.00. (For over one pound, the price depends on distance.) Compute the postage for any piece of mail weighing up to one pound.

52. Write a program to balance your bank statement. The program should ask for the figures it needs (don't forget outstanding checks, bank charges, and deposits not shown on the statement).

53. First Federal Savings Company pays 5-1/2% per year interest, calculated daily, on its customers' savings accounts. If x dollars are deposited and d days later the account is closed, how much is withdrawn?

54. Assume data for a payroll department are given in this form:

Employee number	Hourly rate
25	2.43
73	7.15
36	3.50
42	2.43
etc.	

Store these data as an array. Write a program that will enter the hours worked in a given week as the third column. The program should first ask for the employee number and the hours worked and enter the number of hours in the third column in the appropriate row. If an employee did not work during the given week, a zero should be entered in the third column.

55. Using the data from the problem above, write a program that will compute the wages earned for the week for each employee and enter the computed wages as the fourth column of the array. The rate of pay for each employee is given in Column 2 and the computation is as follows:

The straight hourly rate is used for up to 40 hours.

For 40 to 48 hours, time-and-a-half is paid.

For over 48 hours, double-time is paid.

The program should print the total payroll (the sum of the fourth column).

56. If P dollars are deposited in a savings account paying 5-3/4% interest, compounded quarterly, what is the amount that accumulates in y years?

Hint: The formula for compound interest computations will be found on page 35.

57. On a $20,000 30-year mortgage with 9% interest, the monthly payments are $160.93. Some of this payment is for interest and some of it is for repayment of the loan. Make a table that shows how much of each monthly payment is for interest, how much is for repayment of the principle, and what the balance of the loan is. (This is called an amortization schedule.) This table would have 30 × 12 = 360 lines if printed. Do not print the entire table; make annual summaries instead.

Hint: The calculation must be done month by month with interest calculated on the unpaid balance of the loan. Thus, for the first month the interest paid would be on $20,000, but after that the interest would be decreasing since the unpaid balance would be smaller. Caution: Because of rounding, the final payment will be slightly different from $160.93.

Applications in Linear Algebra

58. The product of a matrix A and a scalar c is the matrix obtained from A by multiplying each of its members by c.

Example:

$$c\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix} = \begin{pmatrix} cx_{11} & cx_{12} & cx_{13} \\ cx_{21} & cx_{22} & cx_{23} \end{pmatrix}$$

Write a program that will find the product of any matrix by any scalar.

59. If A and B are both n × m matrices, the sum of A and B is another n × m matrix C such that

$$C_{ij} = A_{ij} + B_{ij}$$

Write a program that will add matrices.

60. If $X = (X_1, X_2, \ldots, X_n)$ and $Y = (Y_1, Y_2, \ldots, Y_n)$ are two vectors of the same length, the dot-product of the two vectors is a number defined as follows:

$$X \cdot Y = X_1Y_1 + X_2Y_2 + \cdots + X_nY_n$$

Write a program to evaluate dot-products.

61. If A is an m × n matrix and B is an n × k matrix, then the product of A and B is an m × k matrix C such that

$$C_{ij} = A_{i1}B_{ij} + A_{i2}B_{2j} + \cdots + A_{in}B_{nj}$$

(This is the same as the dot-product of the ith row of A and jth column of B.) Write a program that will perform matrix multiplication.

62. The value of a 2 × 2 determinant $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ is ad - bc. Write a program that evaluates 2 × 2 determinants.

63. The value of a 3 × 3 determinant $\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$ is defined to be the number $a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}$. Write a program that evaluates 3 × 3 determinants.

64. Two simultaneous linear equations

$$ax + by = c$$

$$dx + cy = f$$

316

where both c and f are not zero have a unique solution *if and only if* the determinant $\begin{vmatrix} a & b \\ d & e \end{vmatrix}$ is not zero. Write a program that determines whether or not a given pair of simultaneous linear equations has a solution.

65. Suppose two simultaneous linear equations are given:

$$ax + by = c$$

$$dx + ey = f$$

If there is a unique solution it will be

$$x = \frac{\begin{vmatrix} c & b \\ f & e \end{vmatrix}}{\begin{vmatrix} a & b \\ d & e \end{vmatrix}} \quad \text{and} \quad y = \frac{\begin{vmatrix} a & c \\ d & f \end{vmatrix}}{\begin{vmatrix} a & b \\ d & e \end{vmatrix}}$$

Write a program that solves two simultaneous linear equations.

66. The value of a 3 × 3 determinant

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

can be given in terms of "minors" (related 2 × 2 determinants):

$$c_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

Write a program that calculates the value of a 3 × 3 determinant by this method.

67. Suppose three simultaneous linear equations are given:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = k_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = k_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = k_3$$

If there is a unique solution it will be

$$x_1 = \frac{\begin{vmatrix} k_1 & a_{12} & a_{13} \\ k_2 & a_{22} & a_{23} \\ k_3 & a_{32} & a_{33} \end{vmatrix}}{A}$$

$$x_2 = \frac{\begin{vmatrix} a_{11} & k_1 & a_{13} \\ a_{21} & k_2 & a_{23} \\ a_{31} & k_3 & a_{33} \end{vmatrix}}{A}$$

$$x_3 = \frac{\begin{vmatrix} a_{11} & a_{12} & k_1 \\ a_{21} & a_{22} & k_2 \\ a_{31} & a_{32} & k_3 \end{vmatrix}}{A}$$

where

$$A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

318

If not all of k_1, k_2, and k_3 are zero, the solution exists and is unique if and only if A is not equal to 0. Write a program that solves three simultaneous linear equations.

68. The method of solution of simultaneous linear equations shown in the above problem can be extended to any number of equations. Write a program to solve four simultaneous linear equations.

69. If the product (see Problem 61) of the matrices $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $\begin{pmatrix} x & y \\ z & w \end{pmatrix}$ is the identity matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, then $\begin{pmatrix} x & y \\ z & w \end{pmatrix}$ is said to be the inverse of $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Find the inverse of a matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

Hint: The inverse exists if and only if the determinant $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ is not zero. This problem requires the solution of four simultaneous equations; use the program from the problem above.

Applications in Probability and Statistics

70. Find the average of four numbers A, B, C, and D. Then find which of the four numbers is closest to the average.

71. For a list of numbers L_1, L_2, ..., L_n, find the average.

72. Find the average of all the positive (negative, non-zero) numbers in a given list of numbers.

73. Suppose scores for a certain test are given in this form:

Number of students	Test score
n_1	t_1
n_2	t_2
n_3	t_3
etc.	

Find the mean score.

74. Find the mean and standard deviation for a given list of numbers.

Hint: The formula for standard deviation will be found on page 36.

75. Suppose L_1, L_2, ..., L_i is a list of test scores for a group of i people. Form a new list N_1, N_2, ..., N_i of the standardized test score for each person.

Hint: The formula for standard scores will be found on page 36.

76. Suppose IQ scores and programming aptitude scores for n students are given in this form:

IQ score	Programming aptitude score
x_1	y_1
x_2	y_2
.	.
.	.
.	.
x_n	y_n

Estimate the correlation between IQ and programming aptitude by calculating the correlation coefficient.

77. Suppose pre- and post-test scores for a course in calculus are given in this form:

Pre-test score	Post-test score
x_1	y_1
x_2	y_2
.	.
.	.
.	.
x_n	y_n

Assume these are given as standard scores. (If not, use the program from Problem 75 above to standardize them.) The post-test score for an individual can be predicted from his pre-test score by an equation of this form:

$$y = rx$$

where r is the correlation coefficient of $\{(x_i, y_i)\}$. Find the value of r. Also, find the sample standard error of estimate for predicting the standard post-test scores.

78. Suppose data on height and weight for n children are given in this form:

Height	Weight
x_1	y_1
x_2	y_2
.	.
.	.
.	.
x_n	y_n

The weight of an individual can be "predicted" from his height by using a regression equation of the form

$$y = a(x - c) + b$$

Calculate the values of a, c, and b for this equation.

Hint: See the formula for the regression equation for prediction of y from x on page 36.

79. A set of n objects can be arranged into linear order in n! ways, where $n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$. Find the value of n!.

80. If r objects are drawn from a set of n objects, the total number of possible arrangements (permutations) is

$$P(n, r) = n(n-1)(n-2)\cdots(n-r+1)$$

Calculate the value of P(n, r).

81. If r objects are drawn from a set of n objects, the total number of combinations (without regard to order) is

$$C(n, r) = \frac{P(n, r)}{r!}$$

where P(n, r) is as defined in the above problem. Use this formula to find the total number of bridge hands (13-card hands) that can be dealt from a deck of 52 cards. Also compute the probability of being dealt a bridge hand that contains cards in one suit only. (The probability is equal to the number of possible one-suit bridge hands divided by the total number of possible bridge hands.)

Applications in Arithmetic

82. Write a program that will count from n to m. If n is greater than m the program should count backwards.

83. Convert any decimal number between 0 and 1 to a fraction with a power of 10 in the denominator.

Examples: 0.67 is converted to 67/100.

0.7 is converted to 7/10.

84. For two integers M and N, find the integer quotient and the remainder of M divided by N.

Example: If M = 14 and N = 4, the quotient is 3 and the remainder is 2.

85. List all the common divisors of two integers M and N.

Example: If M = 42 and N = 70, the common divisors are 1, 2, 7, and 14.

86. Convert any decimal integer between 1 and 100 to a binary number.

87. List all of the prime factors of any integer greater than 2.

88. Reduce any fraction to lowest terms.

Example: $\frac{8}{12} = \frac{2}{3}$

89. Do the following sequence of computations, listing each result as it is computed:

Result 1: Compute 1 divided by 3

Result 2: Compute 1 divided by (3 + the previous result)

Result 3: Compute 1 divided by (3 + the previous result)
etc.

90. Write a program, using a counter C = 1, 2, 3, ..., that computes and prints values for y where y is defined as

$$\frac{c}{(\text{previous value of } y)}$$

The initial value of y is 1. Continue the computations until the value of y is greater than 4.

91. The Fibonacci numbers are

1, 2, 3, 5, 8, 13, ...

where each number is found by adding the previous two numbers in the sequence. List the first 15 Fibonacci numbers.

Applications in Calculus

92. Type a list of the first n numbers in each of these sequences.

(a) 1/2, 2/3, 3/4, ...

(b) $1 - \frac{1}{9}$, $1 - \frac{1}{99}$, $1 - \frac{1}{999}$, ...

(c) 1^2, 2^2, 3^2, ...

(d) $\frac{1}{1^2}$, $\frac{1}{2^2}$, $\frac{1}{3^2}$, $\frac{1}{4^2}$, ...

(e) $\frac{1^2}{2^1}$, $\frac{2^2}{2^2}$, $\frac{3^2}{2^3}$, $\frac{4^2}{2^4}$, ...

93. Approximate the sum of the series

$$\frac{1}{3} + \frac{1}{9} + \frac{1}{27} + \ldots$$

by adding one term at a time until the addition of another term will not change the total.

94. Approximate the sum of the series

$$1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots$$

95. The series

$$ar + ar^2 + ar^3 + \ldots$$

converges (has a sum) for $-1 < r < 1$. Approximate the sum of this series for $r = 1/10, 1/100, 1/1000$ and $a = 1, 10, 100$.

96. Approximate the sum of the series

$$\frac{3}{50} - \frac{3}{50^2} + \frac{3}{50^3} - \frac{3}{50^4} + \ldots$$

97. Approximate the sum of the series

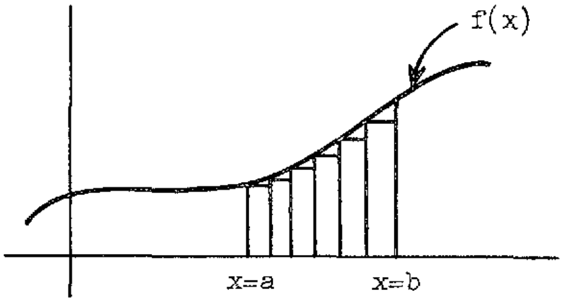
$$\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^4 + \left(\frac{1}{3}\right)^6 + \left(\frac{1}{3}\right)^8 + \ldots$$

98. Approximate the sum of the series

$$\frac{x-1}{3} + \frac{(x-1)^2}{5} + \frac{(x-1)^3}{7} + \frac{(x-1)^4}{9} + \ldots$$

for $x = .6, .7, .8, .9$.

99. The area under a curve can be approximated by finding the area of a set of rectangles as shown below. If the x-axis from a to b is divided into n parts, the approximation is the sum of n rectangles, each of which has an area $f(x) \cdot \frac{(b-a)}{n}$, with $x = a$, $a + \frac{(b-a)}{n}$, $a + 2\frac{(b-a)}{n}$ $a + 3\frac{(b-a)}{n}$, ..., $a + (n-1)\frac{(b-a)}{n}$. By taking n = 1, 2, 3, ... a sequence of closer and closer approximations can be made. Using this method, approximate the area under the curve of $f(x) = \sqrt{x^3 - \cos(x)}$ from x = 1 to x = 2.



100. Approximate the area under the curve of $f(x) = \frac{1}{x^2 + 1}$ from x = 0 to x = 1.

N74-26710

COMPARISON OF STUDENT PERFORMANCE AND ATTITUDE UNDER THREE LESSON-SELECTION STRATEGIES IN COMPUTER-ASSISTED INSTRUCTION

by

Marian H. Beard, Paul V. Lorton, Barbara W. Searle,

and R. C. Atkinson

December 31, 1973

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES,
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

326

TECHNICAL REPORT SUMMARY

This study investigated the effects on student performance and attitude of three different strategies for selecting lessons in a course in computer programming presented by computer. The focus of the investigation was a comparison of computer selection vs student selection of instructional material.[1]

A commonly held belief is that students prefer to exercise control over their course of study; this assumes that they are capable of making such decisions, and that provision for such control will be a motivating factor reflected in an increased rate of learning. Little experimental data exist to support this belief. In fact, it is not even known how much control students will exercise when given the option. This study was designed, in part, to examine the effect of student control on both performance and attitude.

The study was conducted using eight remote terminals linked by telephone lines to the PDP-10 computer at the Computer-assisted Instruction (CAI) Laboratory of the Institute for Mathematical Studies in the Social Sciences (IMSSS) at Stanford University. A simple and inexpensive device (Model-33 teletype) was used as the student terminal. The CAI program imposed no time constraints; students were free to spend as much time as they chose on any lesson.

The course, "Computer Programming in AID," was designed for one quarter or one semester of instruction in the Algebraic Interpretive Dialogue (AID), a mathematically oriented programming language. It consists of 36 parallel sets of short and long lessons as well as tests and extra-credit problems. Long lessons cover the same material as the corresponding short lessons, but in greater detail. An outline of the course is shown in Table 1.

Three experimental conditions were established: free choice, no choice, and program choice. Students in the "free-choice" condition were permitted to alter their position in the course at any time. Students in the "no choice" condition followed a straight path through the long lessons, with a test after every fourth lesson, and were not allowed to alter the sequence of lessons. Students in the "program-choice" condition followed a modified path through the short lessons with a test after every fourth lesson. The progress of these students was monitored by the program, and the corresponding long lesson was presented when a student performed below a set criterion, either in a short lesson or on a test.

Sixty students, distributed between both schools and over the entire 1972-1973 school year, were selected as subjects for this study. Three equal groups were created by random assignment to each selection condition.

The measures used in the analysis were: the Computer Programming Aptitude Battery, two final examinations prepared by the project staff, the responses to an attitude questionnaire, the number of times a student signed on to the course, the number of minutes spent signed on, the number of lessons taken, the number of problems correct, the number of problems attempted, the percentage correct, and the highest lesson completed.

Our results indicated no significant differences among the three conditions on any of the performance or attitude measures. It cannot be said, on the basis of these findings, that a curriculum offering extensive student control is either superior or inferior to a program-controlled sequence. In fact, it appears that the "free-choice" students did not make sufficient use of their choice option to alter dramatically the sequence of lessons.

The implications of these results deserve some discussion. A student's use of choice options is related to the curriculum he is studying, both in its content and in its instructional design. The subject matter taught in the AID course was organized in a hierarchical, cumulative set of lessons, each to some extent dependent on concepts and skills developed in earlier lessons. This inherently linear organization, although quite common in computer programming instruction, does not lend itself to student control over the curriculum, beyond skipping or reviewing items, as evidenced by the similarity of the sequences followed by the subjects in the three groups.

It is possible to construct a fundamentally nonlinear instructional-experimental environment in which program and student strategies can be examined more fully. Building on the results of the current study, we are developing and testing a very different CAI curriculum. The course content will be the same--introductory programming--but one major feature distinguishes the new curriculum from the AID course. The instructional sequence will be intentionally nonlinear, i.e., it will be dependent on students' acquisition of skills in interrelated conceptual areas instead of their progress through a defined series of lessons. The curriculum driver will be capable of making decisions about students' abilities on the basis of an informational network of programming concepts, and will be capable of selecting an instructional task appropriate to students at their particular level. This design implies the possibility of exploring differences in the performance of those students whose selections are made by the program and those who are forced to choose problems that cannot, by the nature of the network design, be sequenced in a preplanned hierarchy. There will be no predetermined, recognizable "default" sequence, and to the students, the curriculum will appear as an individualized sequence of programming tasks.

One planned experiment will again involve program-selection and student-selection modes: in the program-selection mode all instruction, hints, and problems will be generated by the program as determined by its decision-making capabilities. In the student-selection mode, the problems and instructional hints will be specifically requested by the student.

INTRODUCTION AND BACKGROUND

Environment and Equipment

This study was initiated as a prelude to a more elaborate investigation of branching strategies. It was thought that the branching procedure used here could answer certain preliminary questions on the evaluation instruments and on the content of the course itself.

The study was conducted using four CAI terminals located at the University of San Francisco (USF) and four terminals located at De Anza College in Cupertino, California. The terminals were linked by telephone lines to the PDP-10 computer at the CAI Laboratory of IMSSS at Stanford University.

The Stanford CAI communication network supports approximately 200 terminals, ranging from Model-33 teletypes operating at 100 words per minute to high-speed cathode-ray tube displays operating at 10,000 words per minute. Although they provide no audio, visual, or graphic capabilities, teletypes are sturdy, low-cost devices that provide the student with a printed copy of his interaction with the instructional program.

The CAI terminals at USF were located in a classroom near the office of the College of Business Administration, under whose auspices the research program at USF was implemented. On weekdays, students had free access to the CAI terminals from 12 a.m. to 10:00 p.m., and on weekends as permitted by the scheduling of computer down-time.

Schedules were used to apportion terminal time; three terminals were available for advance sign-up in one-hour time blocks. The fourth terminal was available on a first-come, first-serve basis for one-hour periods. Under ideal operating conditions four terminals would have provided 200 hours of terminal time per week, enough to comfortably accommodate the approximately 50 students registered for the course during the fall semester. Scheduling problems did develop, however, and thus enrollment for the spring semester was kept under 25 in order to insure adequate access to the terminals.

The four terminals at De Anza College were located in the Data Processing Laboratory. The course was given by the Business and Data Processing Division and was open to all students. Eighteen students were enrolled for the fall quarter, 14 for the winter quarter, and 16 for the spring quarter. With this number of students no scheduling problems arose.

The CAI program imposed no time constraints on students working at terminals. Students had unlimited time to respond to each question, and to complete a lesson. The process of initiating interaction with the instructional program is called "signing on," and disconnecting from it, "signing off." When a student finished a lesson he was free to sign off, or to continue with another lesson. He was also permitted to sign off in the middle of a lesson.

Curriculum

The course, Computer Programming in AID, was designed for one quarter or one semester of instruction in AID. It consists of 36 sets of lessons plus tests and extra-credit problems. An outline of the course is presented in Table 1. AID resembles BASIC in its use of line numbers and in its relatively simple grammatical rules, but it differs from BASIC in that AID allows recursive procedures. The IMSSS implementation of AID is interpretive and provides students with diagnostic messages and flexibility in changing programs. Topics covered by the curriculum include conditional execution, loops, lists, two-dimensional arrays, standard functions, user-defined functions, and recursive functions (see Friend, 1973).

The AID course was extensively revised for use in this investigation. The revised curriculum is organized into four strands, containing Short Lessons (SL), Long Lessons (LL), Tests (T), and Extra-credit Problems (EX). Lessons in the LL strand cover the same material as those in the SL strand, but in greater detail. The average lesson from the SL strand has about 20 problems, while that from the LL strand has about 30 problems. Many of the problems in both types of lesson have from one to three subproblems.

The test strand contains nine tests. A test is designed to cover the immediately preceding four lessons. It contains 40 items, 10 for each of the four lessons.

The EX strand does not contain a lesson at each level; the EX lessons are listed in Table 1. An EX lesson typically contains from one to five programming problems, some of considerable complexity.

DESIGN AND EXPERIMENTAL PROCEDURES

Subjects

Two groups participated in this study. The first consisted of University of San Francisco students enrolled for academic credit in a course introducing the use of computers in business administration. These students are required to take a programming course, but are free to choose among several options. Thus, enrollment for this course was voluntary. The fall class numbered 49, 30 men and 19 women, and the spring class numbered 23, 16 men and 7 women. Subjects were mostly first-year students and none had prior programming experience.

The second group of students attended De Anza Junior College, and did not fulfill any requirements by enrolling in the AID course. The distribution of students enrolled was (a) for fall, 11 men, 7 women; (b) for winter, 9 men, 5 women; and (c) for spring, 9 men, 8 women.

Sixty students, distributed between both schools and over the 1972-1973 school year, were selected as subjects for the results reported below.

Experimental Conditions

The three experimental conditions designed for this study are Student Selection (SS), No Selection (NS), and Program Selection (PS). The conditions are distinguished as follows:

1. SS. A student in the SS group was permitted to alter his position in the course at any time. The use of three control characters was available to him.

Control Character	Action
CTRL-G	choose a different lesson and/or problem
CTRL-T	have the terminal print the answer to the current problem
CIRL-H	skip the current problem

The SS student was permitted to use AID at any time, whether the current problem involved writing a program or not.

2. NS. Procedures for the NS group were designed to guide the student on a straight path through the LL strand, with a test (T strand) after every fourth lesson. The control characters described above did not operate for the NS Group. A student was not allowed to alter the order in which his lessons were presented and he was permitted to use AID only for programming problems.

3. PS. The student in the PS group followed a modified path through the SL strand with a test after every fourth lesson. The control characters described for the SS student were not available to the PS student, and a student was permitted to use AID only for programming problems. The student's progress through the SL strand was modified in two different situations:

1. At the end of each SL the student's score was checked. If he answered 90 percent or more of the problems in the lesson correctly on the first try, he was sent to the corresponding EX lesson if one was available. If his score was below 75 percent, he was sent to the corresponding LL for further work. In either case, after completing the branch lesson he returned to the next lesson in the SL strand.

2. After each test the student's score was checked for the items related to each of the previous four lessons. He repeated the LL lessons related to those concepts on which his test performance fell below 75 percent. After taking the prescribed reviews the student returned to the next SL lesson following the test.

The 60 students were roughly matched on the basis of their performance on the aptitude battery given as a pretest at the beginning of the course. The three equal groups studied here (SS = 20, NS = 20, PS = 20) were created by random assignment.

Criterion Measures

Students were tested at the beginning of the semester using the Computer Programmer Aptitude Battery (CPAB), published by Science Research Associates. The CPAB is comprised of five separately timed tests, measuring the following skills and aptitudes: verbal meaning, reasoning, letter series (a test of abstract reasoning ability), number ability, and diagramming (using flow charts).

Several instruments were used at the end of the semester to evaluate performance and attitude. The project staff prepared a two-part final examination. Part A was an off-line, closed-book test covering the entire course. It contained 53 questions, some requiring constructed responses, others, multiple choice. It was designed to test (a) knowledge of AID syntax, (b) understanding of program flow, (c) ability to analyze a program and to predict its output, and (d) ability to construct or complete programming algorithms to solve a specific problem. Part B consisted of five programming problems that were to be written at CAI terminals. Students were permitted to use notes and the course handbook. For each problem they submitted a listing of their program and sample output. Parts A and B of the final examination can be found in Appendix A.

An attitude questionnaire was administered to USF students. The questionnaire (Appendix B) is a revision of one developed to evaluate a CAI project at Tennessee State University (see Searle, Lorton, Goldberg, Suppes, Ledet, & Jones, 1973). It contains 12 statements about the student's CAI experience. A seven-point scale was used to indicate the degree of agreement with with each statement.

Various parameters of student performance on the course were used. These performance characteristics were obtained from data collected by the instructional program. The program saved all student responses. Only first responses were used to determine the number of problems correct.

The full list of measures used in the analysis includes:

1. Performance on the CPAB

2. Performance on final examinations

   a. Test A (project off-line, closed-book examination)

   b. Test B (project on-line examination)

3. Responses to the attitude questionnaire

4. Number of times the student signed on to course
   (# SIGN ONS)

5. Total number of minutes spent signed on to course
   (MINUTES)

6. Total number of lessons taken
   (LESSONS)

7. Total number of problems worked correctly (# CORRECT)

8. Total number of problems attempted (# PROBLEMS)

9. Percentage correct (PERCENT)

10. Highest lesson completed (TOP LESSON)

ANALYSIS AND DISCUSSION OF EXPERIMENTAL RESULTS

Aptitude Measures

Scores on the CPAB for students in the three experimental groups are shown in Table 2. The CPAB test manual indicates that percentile norms for experienced computer programmers and systems analysts are based on the scores of personnel from a variety of business and industrial installations, including computer manufacturers. Norms for programmer trainees are based on the scores of applicants for jobs with civil service agencies and persons enrolled in basic-computer-systems training at universities or computer-manufacturer training sites. Approximately 80 percent of the experienced programmers and 50 percent of the programmer trainees were college graduates.

Table 3 shows a comparison between the experimental subjects' scores and the norms of the aptitude battery for both programmer trainees and experienced programmers. The average score for the experimental group, 62.06, lies in the 55th percentile on the scale for trainees and in the 9th percentile on the scale for experienced programmers.

The CPAB manual states that performance on the Letter Series Subtest is least affected by education and experience; this may well account for the experimental group's relatively high percentile rank (57) compared with rankings on other subtests on the experienced programmers' scale.

Performance on the CPAB proves to be a useful predictor of performance on the AID course. The correlations between scores on CPAB subtests and two performance measures, percentage correct in the course and score on Test A, are shown in Table 4.

Total score on the CPAB accounts for 46 percent of the variability in percentage correct in the course, and 32 percent of the variability in Test A scores. The claim by the developers of the CPAB that performance on the Diagramming Subtest is highly related to subsequent success in programming is supported by the results in Table 4. The two subtests with lowest predictive ability are verbal meaning and number ability. The AID curriculum uses numerical examples exclusively in providing programming problems; nevertheless, the subtests that depend on reasoning ability serve as better performance predictors.

Curriculum Performance Measures

Descriptive measures of progress in the curriculum for each experimental group are presented in Table 5. The average percentage correct over all lessons for all students was 72.48. Students signed on for sessions at the terminal an average of 59 times and worked, on the average, a total of 2056 minutes. They attempted, on the average, 1303 problems and covered over 36 lessons (including both short and long lessons). There were no significant differences among the three experimental groups on any of the measures of course usage and progress. The NS students, who took only the long lessons, spent more time at the terminals, and attempted more problems than students in the other two groups, but the differences were small.

Use of Choice Options

The SS students were allowed complete control over the selection of lessons. All students had a list of the lessons in the course and were told how to select lessons. The SS students made little use of this opportunity to control the sequence of lessons and, in effect, to 'individualize' their curriculum. The path through the course of the 20 SS students was compared with the standard order of lessons shown in Table 1 (lessons 1-4, test 1; lessons 5-8, test 2; etc.). Ten students showed no deviations from the standard pattern, three students took one or two lessons out of order, three students took three or four lessons out of order and the remaining four students took more than four lessons out of order. Thus, approximately three-fourths of the students made essentially no use of the freedom to change the order of their lessons.

The paths through the course chosen by the four students who deviated most from the standard order are shown in Table 8. Student 1 used the choice option to take tests out of order; in all but one case, he opted to take the tests early. Student 2 took an essentially straight path though the short and long lessons, occasionally skipping an LL lesson to return to it later, and, twice, to return to an EX lesson. Student 3 skipped ahead to work LL lessons out of order, but returned to work SL lessons systematically, skipping only SL11 and SL16. Student 4 skipped around a bit early in the course, but later used the choice option only to take tests out of order.

In almost no cases did students use the choice option to skip forward in the curriculum. Students were extremely conservative in the use of their freedom to sequence the course; most often they used this freedom to take tests out of order or to return to forms of lessons already taken.

Table 7 summarizes the choice of lesson types for the SS students. Students 1-4 are those whose paths are shown in Table 6. Of the remainder, one took LL lessons only, while five combined a mixture of SL and LL lessons in approximately equal numbers. The rest of the students (with only minor exceptions) worked only SL lessons. Thus, approximately half the students chose the fastest straight path through the course.

Final Examinations

A two-part final examination was administered by the project staff to students in the experiment. Results of this examination are shown in Table 8. Because of scheduling difficulties 13 students were unable to take Test B of the examination.

Although the mean scores for the three experimental groups do not differ significantly, the scores for the NS students were slightly higher on Test A and slightly lower on Test B than for the other groups.

Test A was an off-line, paper-and-pencil examination. Results of a linear regression analysis using performance on Test A as the dependent variable are shown in Table 9. The top lesson taken and the score on the CPAB together account for more than 50 percent of the variability in the Test A score.

Attitude Questionnaire

The attitude questionnaire (Appendix B) contains 12 items ranked by students from strong agreement (1) to strong disagreement (7). The mean response by condition to each question is given in Table 10.

Generalizing over all students, the strongest responses showed agreement with the statements in questions 1 and 3. These were "I worked as hard answering questions in the computer lessons as I do in the classroom" and "I like working at my own pace at the terminal," respectively. PS students agreed more strongly than the other groups with question 1 (means are SS = 2.588, NS = 2.632, PS = 1.824), and SS students agreed more strongly with question 3 (SS = 1.412, NS = 2.421, PS = 2.588).

Both of these results demonstrate favorable attitudes toward particular aspects of the CAI experience. The mean responses do not demonstrate a strong negative feeling toward CAI on any question.

Two of the attitude questions show relatively high correlations with some descriptive measures and with test performance; the results are shown in Table 12. The questions are No. 2, "I learned from the computer lessons as well as I would have learned the same lesson in the classroom," and No. 10, "I would like to participate in another CAI course." Students who took more lessons and answered more problems correctly tended to have favorable attitudes. Performance on Test B correlated with positive attitude on questions 2, 3, and 4.

There were no significant differences between conditions in responses to the questions, as shown by the results of an analysis of variance presented in Table 11. For all of the attitude questions, the between-groups degrees of freedom (d.f.) is 2, and the within-groups d.f. is 50. For significance at the .01 level, an F ratio of 5.06 is needed; at the .05 level, an F ratio of 3.18 is needed. None of the ratios found reach these significant values.

Item Analysis

A master list matching items on Part A of the final examination with the lesson each item tested was prepared by the author of the course, J.E. Friend. Student responses to items for which they had and had not taken the appropriate lesson are shown in Table 13.

The labels in the "Lesson Status" column of Table 13 are independent of the three experimental conditions. Each item in the examination tested material covered by both an SL and an LL lesson. For each item, each student falls into one of the "Lesson Status" categories by virtue of those lessons he completed. For example, the "Not Taken" category includes students from all three experimental conditions. The "SL Only" includes only SS and PS students; the "LL Only" includes only SS and NS students; and the "SL & LL" includes only SS and PS students.

Table 13 shows, for example, that of the 1367 incorrect responses tallied on the examination, 462 were made by students who had not taken either SL or LL lessons associated with the items, 455 were made by students who had taken the associated SL lesson only, 274 were by students who had taken the associated LL lesson only, and 176 were by students who had taken both the SL and the LL lessons associated with the item. There were 98 items skipped by students who had taken the lessons on which they were based, compared with 195 items skipped by students who were unfamiliar with the material on which the item was based. There were 349 correct responses made by students who had not taken the appropriate lessons for the items. An examination of these responses revealed that 215 of them were to six questions that gave the student a binary choice (true-false, correct-incorrect), and it is likely that guessing played a large role in producing these correct answers.

Table 14 shows the percentage correct, incorrect, and not tried for all students, and percentage correct and incorrect based on total attempts. Apparently students who took only the LL lesson did substantially better (61.8 to 38.2 percent) than students who took only the SL lesson (51.7 to 48.3 percent). Students who took both the SL and LL lessons fell in between. This is not a surprising finding since most of those who took both lessons needed extra review and were thus not likely to be the best students.

SUMMARY AND CONCLUSIONS

The focus of this investigation was a comparison of computer-program-controlled selection and student-controlled selection of instructional material during one quarter or one semester of instruction in AID. The performance and attitude of 60 students were examined: 20 in the "student-selection" condition, 20 in the "no-selection" condition, and 20 in the "program-selection" condition.

Results indicated no significant differences among the three conditions on any of the performance or attitude measures, although there are interesting correlations among the measures over all students. On the basis of these findings, a curriculum offering extensive student control cannot be demonstrated to be either superior or inferior to a program-controlled sequence.

It is clear that the SS students did not make sufficient use of their choice option to alter dramatically the sequence of lessons, and in this sense, the original question of student vs program control cannot really be examined properly from the data collected.

A student's use of choice options is related to the curriculum he is studying, both in its content and in its instructional design. A curriculum may incorporate various degrees of linearity, branching facility, remedial content, dialogue capability, student performance analysis, parallel content strands, etc, and these features may be developed and combined so that they motivate a student either to exercise options or to accept obvious choices as they are offered.

The subject matter taught in the AID course was organized in a hierarchical, cumulative set of lessons, each to some extent dependent on concepts and skills developed in earlier lessons. This inherently linear organization, although fairly common in conventional instruction in the subject, does not lend itself to the exercise of student control of the curriculum beyond skipping or reviewing, as evidenced by the performance of the subjects of this study. The most effective lesson sequence, in their view, is the straight line of the original conceptual design. The SS students were explicitly encouraged to develop their own alternative strategies, and during the year this encouragement was repeated many times. Thus, it must be concluded that the linear paths were chosen in conscious preference to any individually developed algorithms, which resulted in some disappointment to the experimenters.

The experiment, therefore, does not properly attack the question of modes of control. However, it is possible to construct a fundamentally nonlinear instructional-experimental environment in which program and student strategies can be examined more fully. Partly on the basis of the inconclusive results of the current study, a very different CAI curriculum is being developed and is now in the initial testing stage. The course content will be the same--introductory programming--but one major feature distinguishes the new curriculum from the AID course. The instructional sequence will be intentionally nonlinear, i.e., it will be dependent on students' acquisition of skills in interrelated conceptual areas instead of their progress through a defined series of lessons. The curriculum driver will be capable of making decisions about students' abilities on the basis of an informational network of programming concepts, and will be capable of selecting an instructional task appropriate to students at their particular level. This design implies the possibility of exploring differences in the performance of those students whose selections are made by the program and those who are forced to choose problems that cannot, by the nature of the network design, be sequenced in a preplanned hierarchy. There will be no predetermined, recognizable "default" sequence, and to the students, the curriculum will appear as an individualized sequence of programming tasks. Instruction will be given only in response to the students' difficulties and requests.

The new course, which will teach the BASIC programming language, is being designed to test selection strategies in a more fluid environment. In the PS mode, all instruction, hints, and problems will be generated by the program as determined by its decision-making capabilities. Note that this requires considerable error diagnosis and interactive capabilities. In the SS mode, the problems and instructional hints will not be given automatically by the program, but must be requested specifically by the student.

It is hoped that this design will facilitate experimentation with instructional control strategies in a technical field, and at the same time allow enough freedom in the curriculum to make a "strategy" meaningful and necessary.

TABLE 1

AID Lessons

		Lesson identifiers		
Topic	Test	Short lesson	Long lesson	Extra credit
1 How to use the instructional program		SL 1	LL 1	-
2 Using AID for arithmetic: The TYPE command		SL 2	LL 2	-
3 Order of arithmetic operations		SL 3	LL 3	-
4 Exponents and scientific notation		SL 4	LL 4	-
Test 1	T1			
5 The SET and DELETE commands		SL 5	LL 5	EX 5
6 Indirect steps, the DO command, the FOR clause		SL 6	LL 6	-
7 Stored programs: Parts and files		SL 7	LL 7	-
8 The DEMAND command and the TIMES modifier		SL 8	LL 8	EX 8
Test 2	T2			
9 Relations and the use of the "if" clause		SL 9	LL 9	EX 9
10 The TO command		SL 10	LL 10	EX 10
11 Debugging techniques		SL 11	LL 11	-
12 The indirect use of DO		SL 12	LL 12	EX 12
Test 3	T3			
13 More on debugging		SL 13	LL 13	-
14 The FORM statement		SL 14	LL 14	EX 14
15 Absolute value		SL 15	LL 15	EX 15
16 Loops		SL 16	LL 16	EX 16
Test 4	T4			
17 More on loops		SL 17	LL 17	EX 17
18 Loops and the FOR clause		SL 18	LL 18	EX 18
19 Debugging tools: STOP and GO		SL 19	LL 19	-
20 Loops with a DEMAND command		SL 20	LL 20	EX 20
Test 5	T5			
21 Lists		SL 21	LL 21	EX 21
22 More on lists		SL 22	LL 22	EX 22
23 Arrays		SL 23	LL 23	-
24 Nested loops and nested DO commands		SL 24	LL 24	EX 24
Test 6	T6			
25 More on arrays		SL 25	LL 25	EX 25
26 The LET command		SL 26	LL 26	-
27 Standard functions: SQRT, IP, FP, SGN		SL 27	LL 27	EX 27
28 SUM, PROD, MAX, and MIN		SL 28	LL 28	EX 28
Test 7	T7			

TABLE 1 (cont.)

Topic	Test	Short lesson	Long lesson	Extra credit
29 Conditional functions		SL 29	LL 29	-
30 Standard functions: DP, XP		SL 30	LL 30	EX 30
31 Boolean expressions: AND, OR, and NOT		SL 31	LL 31	-
32 More on Boolean expressions: LET and TV		SL 32	LL 32	-
Test 8	T8			
33 The function FIRST		SL 33	LL 33	EX 33
34 Standard functions: SIN and COS		SL 34	LL 34	EX 34
35 Standard functions: EXP and LOG		SL 35	LL 35	-
36 Recursive functions		SL 36	LL 36	-
Test 9	T9			

TABLE 2

Scores on the Computer Programmer Aptitude Battery

	Experimental condition					
	SS		NS		PS	
Part	Mean	S. D.	Mean	S. D.	Mean	S. D.
Verbal Meaning	12.90	4.15	13.35	6.36	14.35	4.29
Reasoning	9.15	3.51	9.15	3.97	9.00	4.43
Letter Series	11.00	4.03	11.05	5.31	12.65	4.08
Number Ability	11.60	3.58	11.10	3.22	10.40	3.18
Diagramming	15.80	8.77	17.40	10.39	17.80	9.44
Total	60.45	16.10	62.05	23.54	63.70	19.43

TABLE 3

Comparison of Subject and Test Norms

Computer Aptitude Battery

Subtest	Mean raw score	Percentile ranking	
		Scale 1[a]	Scale 2[b]
Verbal Meaning	13.53	46	15
Reasoning	9.10	61	17
Letter Series	11.56	66	57
Number Ability	11.03	54	20
Diagramming	17.00	54	9
Total	62.06	55	9

[a] Based on programmer trainee norms.

[b] Based on experienced programmer norms.

356

TABLE 4

Correlations Between Performance on CPAB Subtests and Two Course Performance Measures

Subtest	Percent correct	Test A
Verbal Meaning	.315	.295
Reasoning	.554	.585
Letter Series	.560	.394
Number Ability	.280	.312
Diagramming	.643	.492
Total	.666	.564

357

TABLE 5

Measures of Progress in the Curriculum

	Experimental Condition			
	SS	NS	PS	Total
No. sign-ons	53.15	63.85	60.50	59.16
Minutes	1995.96	2187.55	1984.18	2055.89
Lessons	35.00	36.90	38.65	36.85
No. correct	876.10	1075.95	891.10	947.71
No. problems	1242.30	1479.00	1188.90	1303.40
Percent correct	71.20	71.80	74.45	72.48
Top lesson	25.30	29.45	24.30	26.35

TABLE 6

Choice of Path Through the Curriculum for SS Students

STUDENT 1

Lesson	SL[a]	LL	Test	EX
1	1			
2	2			
3	3			
4		4		
T1			5	
5		6		
6		7		
7		8		
8		9		
T2			10	
9		11		
10		12		
11	13			
12		14		
T3			15	
13		16		
14		17		
15		18		
16		19		
T4			21*	
17		20		
18		22		
19	23			
20		24		
T5			26*	
21		25*		
22		27		
23		28		
24		29		
T6			31*	
25		30		
26		32		
27		33		
28	34			
T7			39*	
29		35		
30		36		
31		37		
32		38		
T8			41*	
33		40		
34		42		
35		43		
36		44		

[a] Numbers show the order in which lessons were taken.
* Starred lessons were taken out of order.

TABLE 6 (cont.)

STUDENT 2

Lesson	SL	LL	Test	EX
1	1	2		
2	3	4		
3	5	6		
4	7	8		
T1			9	
5	10	11		12
6	13	14		
7	15	16		
8	17	18		
T2			19	
9	20	21		22
10	24	23		
11	25	30*		
12	26	31*		
T3				27
13	28	29		
14	32	33		
15	34	35		
16	36	37		
T4			38	
17	39	40		41
18	42	43		46*
19	44	45		
20	47	48		49
T5			51*	
21	50	52		
22	53	55*		59*
23	56	54		
24	58	57		
T6			60*	
25	61	62		63
26	64	65		

TABLE 6 (cont.)

STUDENT 3

Lesson	SL	LL	Test	EX
1		1		
2	2			
3	3	5		
4	6			
T1				
5	7	10*		8
6	9	4*		
7	13	17*		
8	14	11*		15
T2			16	
9	18	12*		
10	19	20		
11		24		
12	26	25		
T3			23*	
13	27	28		
14	30	29		
15	31	21*		
16		22*		
T4				
17	32	33		
18	34			
19	35			
20	36	37		
T5			38	
21	39			
22	40			
23	41			
24	42			
T6			43	

TABLE 6 (cont.)

STUDENT 4

Lesson	SL	LL	Test	EX
1	1	3*		
2	2	4		
3	10*	6		
T1			5*	
5	16*	8		
6	13	12		
7	17	14		
8	18	15		
T2			9*	
9	19	20		
10		21		
11		22		
12		23		
T3			24	
13		25		
14		26		
15		27		
16		28		
T4			29	
17		30		
18		31		
19		32		
20		33		
T5			34	
21		35		
22		36		
23		37		
24		38		
T6			39	
25		40		
26		41		
27		43		
28		44		
T7			42*	
29		45		
30		46		
31		47		
32		48		
T8			11*	
33		49		
34		50		
35		52		
36		53		
T9			51*	

TABLE 7

Types of Lessons Taken by SS Students

Student	Number of lessons	
	SL	LL
1	5	22
2	26	26
3	21	16
4	8	27
5	24	0
6	3	6
7	23	0
8	21	0
9	0	22
10	21	0
11	20	4
12	16	6
13	11	17
14	16	13
15	12	20
16	10	13
17	23	6
18	18	1
19	26	1
20	27	2

TABLE 8

Scores on Project-designed Final Examination, Number Correct

	Condition					
	SS		NS		PS	
	N	Mean	N	Mean	N	Mean
Test A	20	22.70	20	27.85	20	24.50
Test B	15	15.73	16	13.00	16	14.37

TABLE 9

Step-wise Regression Summary Table with Test A

as Dependent Variable

Step	Variable	Multiple r	Multiple r^2	Last regression coefficient
1	Top lesson	.5650	.3192	.5364
2	Total problems	.7296	.5323	.2277
3	Sign-ons	.7534	.5676	.0429
4	Experimental condition	.7556	.5709	.8018
5	Total lessons	.7573	.5735	-.0866
6	Total minutes	.7581	.5747	.0005

Note.--Last constant used = -5.3006.

TABLE 10

Scores on Attitude Questionnaire Items

Question	Condition			Total	Positive or or negative statement (P,N)
	SS	NS	PS		
1	2.588	2.632	1.824	2.358	P
2	3.294	3.105	3.941	3.434	P
3	1.412	2.421	2.168	2.151	P
4	5.059	4.474	4.471	4.660	N
5	4.471	3.579	3.529	3.849	P
6	4.118	4.105	5.059	4.415	P
7	4.882	4.632	5.529	5.000	N
8	3.176	3.263	3.824	3.415	P
9	3.647	4.263	3.059	3.679	N
10	4.176	3.368	4.588	4.019	P
11	3.765	4.526	3.706	4.019	N
12	4.294	3.737	4.000	4.000	P

366

TABLE 11

Analysis of Variance Among Experimental Conditions

on Attitude Questionnaire

Question	F Ratio
1	1.483
2	0.844
3	2.702
4	0.440
5	1.137
6	1.333
7	1.573
8	0.481
9	1.729
10	1.298
11	1.457
12	0.336

TABLE 12

Correlations Between Attitude and Performance Measures

Question	Measure	Correlation
Question 2.		
"I learned from the computer lessons as well as I would have learned the same lessons in the classroom."	Lessons	-.4484
	No. correct	-.5418
	Top lesson	-.4929
Question 10.		
"I would like to participate in another CAI course."	Lessons	-.4951
	No. correct	-.5307
	Top lesson	-.5451
Question 3.		
"I like working at my own pace at the terminal."	Test B	-.5036
Question 4.		
"I would prefer competing with my fellow students in the classroom rather than working at computer lessons."	Test B	.4094

TABLE 14

Responses to Final Examination Items: Percentage of Students Responding Correctly and Incorrectly

Lesson status	Percentage				
	Based on total taking test			Based on total attempting item	
	Correct	Incorrect	Not tried	Correct	Incorrect
Not taken	34.7	49.5	19.4	43.0	57.0
SL only	49.7	46.4	3.9	51.7	48.3
LL only	58.1	35.9	6.0	61.8	38.2
SL and LL	52.0	44.4	3.5	53.9	46.1
Total	50.4	40.8	8.8	55.2	44.8

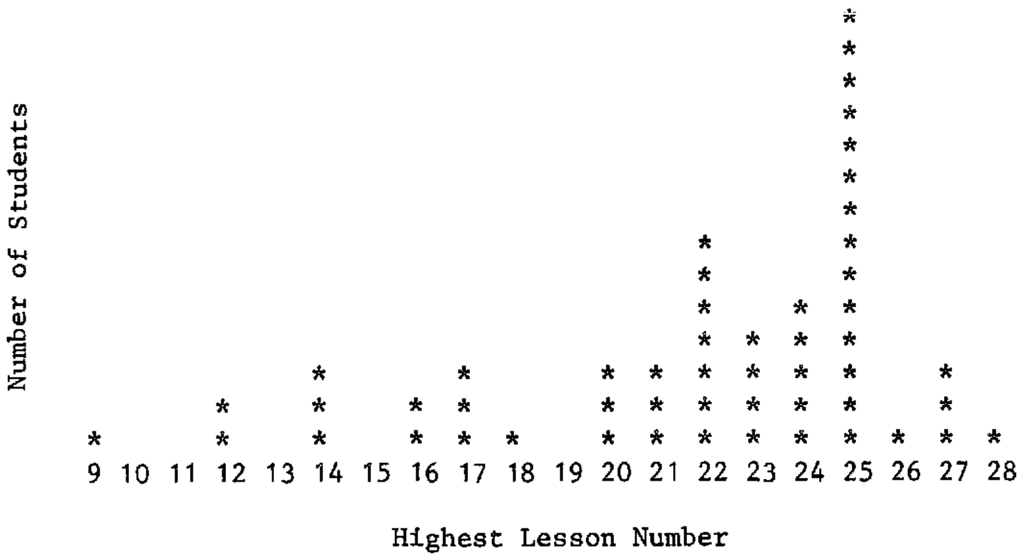


Fig. 1. Highest lesson completed in AID course.

370

Appendix A

Introduction to AID Programming 1972-73
Final Examination

Instructions to Examiners

The final examination for the course "Introduction to AID Programming" consists of two parts: Part A is a 50-minute paper-and-pencil test, and Part B is a 50-minute open-book programming test. If the two parts must be given on the same day, they should be given in two separate sessions with a 5-10 minute rest period between sessions.

Part A. No books or notes of any kind are to be allowed during Part A of the final examination. The students are not to be allowed to use a teletype. All that is needed is a copy of Part A and a pen or pencil.

Hand out the copies of Part A with instructions not to open the test until told to do so. Ask the students to read the instructions on the cover page. Allow about 1 minute for this before giving the signal to start the test. Allow 50 minutes for Part A.

There are 50 test items in Part A. Each correct answer counts 1 point, for a total of 50 points. No partial credit will be given for the items in Part A. There will be no penalty for incorrect guesses (no points will be subtracted for wrong answers).

Part B. Students should be told beforehand that Part B will be an open-book test. They should be asked to bring any books and notes that they wish, including the Supplementary Handbook for Introduction to AID Programming.

At least two days before the students are to take Part B of the final examination, but after their last working session, inform your Stanford representative of which students will take Part B, and when. The computer record for each student will be set so that the next time he signs on he will be automatically switched to the AID interpreter so that he will be able to do the programming problems in Part B.

Before handing out copies of Part B, ask the students to sign on. Check to be sure each student has been automatically switched to the AID interpreter. If this does not happen, call Stanford immediately.

After each student is signed on, and is in communication with the AID interpreter, hand out the copies of Part B with the instruction not to open the test until told to do so. Allow the students time to read the instructions on the cover page--about 1 minute--and then give the signal to start. Allow 50 minutes for Part B.

There are 5 programming problems in Part B. Each problem counts 10 points, for a total of 50 points. Partial credit will be allowed for partially correct programs.

Here is a brief grading guideline to help you answer questions that students may ask during the final examination:

(1) The programs are expected to function correctly only for the range of values of the input variables specified in the problem. Thus, for Problem 2, the program need not cope with negative values of H, and for Problem 3, the program need function correctly only for weights between 0 and 16 ounces, inclusive.

(2) The length of the program is immaterial, only the correctness of the results will be considered in grading.

(3) There are several methods of solving each of the problems in Part B, and no one method is preferred. Any method that provides a general solution and produces correct results will be considered correct.

(4) For Problems 1, 2, 3, and 4, specific test values of the input variables are given. However, a program that produces correct results for these test values only, and not for other values of the input variables, will not be considered a correct solution; the program must provide a general solution.

TURN IN ALL TEST PAPERS TO STANFORD. These tests will be used for research purposes and will not be returned. If you wish to use these tests for assigning grades to your students, you may grade the tests and record the grades before you turn them in to Stanford; otherwise, you need not grade the tests.

Introduction to AID Programming 1972-73

Final Examination

Part A

(50 points)

****

DO NOT TURN THE PAGE.

****

Name ______________________________

Student number ______________________________

Instructor's name ______________________________

Name of school or college ______________________________

Date ______________________________

Instructions: You may not use books, notes, or other materials during this part (Part A) of the final examination. There are 50 test items in Part A. No partial credit will be given. You will not be penalized for guessing (no points will be subtracted for wrong answers). You will have 50 minutes to complete the test.

****

DO NOT TURN THE PAGE UNTIL INSTRUCTED TO DO SO.

****

373

Rewrite each command correctly.

1. IF X < 2 DUE PART 3

____

2. DO STP 3.6 FOR X = 1 TO 100

____

3. TYPE X(Y+Z) AND X(Y-Z) AND X(Y ÷ Z)

____

Select the expression(s) that are equivalent to the given expression.

4. A/B - C/D + E

___ (A/B) - (C/D + E)

___ (A/B) - (C/D)+ E

___ (A/B - C/(D + E)

5. U/V/W/X

___ (U/V)/(W/X)

___ ((U/V)/W)/X

___ (U/V)/W/X

Indicate whether each command is correct or incorrect.

	Correct	Incorrect
6a. FILE PART 6, A AS ITEM 3	___	___
6b. LET H(X) = X * 10 IF Y < 100	___	___
7a. TYPE F(2) * 10↑4 IF 6 < 3 IS FALSE	___	___
7b. DISCARD PART 3	___	___
8a. TYPE FORM 8, X - 98.6, STEP 14.4	___	___
8b. RECALL PART 5	___	___
9a. SET M = M + 1 IF N(I) < TRUE	___	___
9b. SET L(N+1) = N + 1	___	___

Write each of the following expressions in AID notation.

10. $\sqrt{a^2 - b^2}$

____

11. $\frac{2x + 3y}{xy}$

____

12. $|m + n + p|$

____

13. $3x^2 - 2x + 5$

____

14. $(8.9054) \times 10^{-8}$

____

15. $(x_1 + x_2) \div (x_3 + x_4)$

____

16. $x \leq y + 10$

____

17. $a + b \neq c$

____

18. $x = \pm 1$

____

Write the formula for each of the following, using AID notation.

19. The average of the numbers w, x, y, and z.

____

375

20. The total price of an item including sales tax if the base price is P and the sales tax is 5%.

____

For each of the following commands, indicate whether a step number is required.

	Must have step number	Must not have step number	May or may not have step number
21. DEMAND X	___	___	___
22. TO STEP 16.2	___	___	___
23. STOP	___	___	___
24. DO PART 1	___	___	___

Give the truth value of each of the following expressions.

	T	F
25. NOT 4 < 3 OR 3 > 4	___	___
26. X = 12 Y = 15 X < Y OR X+X > Y	___	___

For each of the following programs, list the step numbers in the order in which they would be executed.

27.

```
12.8 DEMAND Q
12.9 SET R = Q + 1
12.10 DEMAND Z
12.95 TYPE R - Z
DO PART 12
```

____

28.

```
42.1 SET Z = 5
42.2 TO STEP 42.4 IF Z > 0
42.3 SET Z = -Z
42.4 TYPE Z
DO PART 42
```

____

```
29.  22.1 SET L = 3
     22.9 SET L = L + 1
     22.75 SET L = L + 1
     22.81 DO PART 33 IF L < 5
     22.99 TYPE L
     33.25 SET L = L + 1
     33.35 TYPE L
     22.95 SET L = L - 1
     DO PART 22
```

For each of the following sets of commands, what numeric result would be typed?

```
30.  LET F(X) = X + 10
     TYPE F(2/10)
     F(2/10) = ____________________________

31.  SET A = 16
     LET S = A > 10
     SET B = TV(S)*A + TV(NOT S)*A*2
     TYPE B
     B = ____________________________

32.  SET X = 43.1
     SET Y = IP(X)
     SET Z = FP(X)
     TYPE Y/Z
     Y/Z = ____________________________

33.  TYPE PROD(I = 2, 6, 11: I/2)
     PROD(I = 2, 6, 11: I/2) = ____________________________

34.  SET X = 4596.032
     SET Y = DP(X)*10
     TYPE Y
     Y = ____________________________

35.  LET F(X) = (X < 10: X+10; X/2)
     TYPE F(12)
     F(12) = ________________________

36.  7.1 SET X = 0
     7.2 DO PART 8 FOR N = 1(1)5
     7.3 TYPE X
     8.1 SET X = X + N
     DO PART 7
     X = ____________________________

37.  3.1 SET N = 843.6
     3.2 SET P = N/10
     3.3 TYPE P IN FORM 3
     FORM 3:
     P EQUALS ← ← . ←
     DO PART 3
     P EQUALS _______________________
```

```
38.  5.1 SET N = 1
     5.2 SET K = 0
     5.3 SET F = 5
     5.4 SET K = K + N
     5.5 SET N = N + 1
     5.6 TO STEP 5.4 IF K < F
     5.7 TYPE K
     DO PART 5
     K = ______________________________
```

```
39.  17.1 DO PART 18 FOR I = 1(1)25
     17.2 TYPE L(7)
     18.1 SET L(I) = I + 2
     DO PART 17
     L(7) = ___________________________
```

```
40.  22.1 SET T = 0
     22.2 DO PART 23 FOR I = 1(1)5
     22.3 TYPE T
     23.1 DO PART 24 FOR J = 1(1)3
     24.1 SET T = T + 1
     DO PART 22
     T = ______________________________
```

```
41.  34.1 SET X = FIRST(I = 1(1)10: I/2 - 1 > 2.7)
     34.2 SET Y = X/2 - 1
     34.3 TYPE Y
     DO PART 34
     Y = ______________________________
```

Rewrite each set of commands, using the fewest possible commands, preserving all indicated actions.

```
42.  DELETE X
     DELETE Y           ____________________________________________
     DELETE Z
     SET Z = 2.5        ____________________________________________

                        ____________________________________________

                        ____________________________________________
```

43.

```
SET W = X + 1
SET W = W/2
SET W = 5 - W
TYPE W
```

44.

```
SET X = 5
DO PART 2
DELETE X
SET X = 6
DO PART 2
DELETE X
SET X = 7
DO PART 2
```

45. Write the AID commands that would cause Part 8 to be put into permanent storage.

46. Write the AID command that would print the value of the natural logarithm (to the base e) of 4.75.

47. Complete step 3.1 in program B below so that programs A and B are equivalent.

Program A	Program B
1.1 SET A = 1	3.1 DO PART 4 FOR A = ____________
1.2 TYPE A/3	4.1 TYPE A/3
1.3 SET A = A + 1	DO PART 3
1.4 TO STEP 1.2 IF A < 10	
DO PART 1	

380

48. Suppose two 9 by 17 arrays A and B are given. The following program produces a new array C such that each element in C is the sum of the elements in the corresponding positions in A and B. Complete step 29.2.

```
27.1 DO PART 28 FOR I = 1(1)9
28.1 DO PART 29 FOR J = 1(1)17
29.2 SET ______________________________________________
DO PART 27
```

49. Write the command that will cause Part 12 to be executed 5 times.

____

50. The factorial function n! is defined to be n·(n-1)·(n-2)···3·2·1. For example, 5! = 5×4×3×2×1 = 120. Write a definition in AID notation of a function f such that f(n) = n!.

____

Introduction to AID Programming 1972-73

Final Examination

Part B

(50 points)

****

DO NOT TURN THE PAGE.

****

Name ____

Student number ____________________________________

Instructor's name __________________________________

Name of school or college ____________________________

Date ____

Instructions: Part B is an open-book test; you may use any books, notes, or other materials that you wish. There are 5 programming problems in this part of the final examination. Each problem counts 10 points, and you will be given partial credit for partially correct solutions.

Before you open the test you should be seated at a terminal and signed on. As soon as you sign on, the AID interpreter will start automatically so that you can do the programming problems. If the AID interpreter does not start, raise your hand to get help before the instructor gives the signal to start the test.

For each problem you will be asked to list (print) the completed program and execute it for given values to demonstrate that your program works correctly. This listing and demonstration must be attached to this test and turned in to your instructor for grading. You will have 50 minutes to complete the test.

****

DO NOT TURN THE PAGE UNTIL INSTRUCTED TO DO SO.

****

382

1. Write a program or a function that will convert degrees Fahrenheit to degrees Kelvin. (From degrees Fahrenheit, subtract 32, multiply by 5/9, and add 273.)

   To turn in for grading: When the program is finished, list it by giving this command:

   TYPE ALL

   Execute the program for 38°F, 0°F, and -41°F. Turn in this part of the teletype paper to your instructor for grading, and then delete the program. (DELETE ALL)

2. Write a program that will compute the wages due, to the nearest penny, for H hours of work if the rate of pay is
   $4.37 per hour for 40 hours or less,
   Time-and-a-half for each hour over 40 hours up to and including the 48th hour,
   Double-time for each hour over the 48th hour.

   To turn in for grading: When the program is finished, list it by giving this command:

   TYPE ALL

   Execute the program for H = 37.25, 42.5, and 52.33 hours. Turn in this part of the teletype paper to your instructor for grading, and then delete the program. (DELETE ALL)

3. Write a program that will calculate postage for a piece of air mail weighing up to and including 16 ounces if the rates are
   11¢ per ounce or fraction of an ounce for 0 to 8 ounces,
   $1.00 total for over 8 ounces up to and including 16 ounces.

   To turn in for grading: When the program is finished, list it by giving this command:

   TYPE ALL

   Execute the program for these weights: 5.2 ounces, 8.7 ounces, 3 ounces. Turn in this part of the teletype paper to your instructor for grading, and then delete the program. (DELETE ALL)

4. Write a program that will calculate the mean and standard deviation of a list $x_1, x_2, x_3, \ldots, x_{10}$ of ten numbers. If M is the mean of the numbers $x_1, x_2, x_3, \ldots, x_{10}$, the formula for the standard deviation is

$$\sqrt{\frac{(x_1 - M)^2 + (x_2 - M)^2 + (x_3 - M)^2 + \ldots + (x_{10} - M)^2}{10}}$$

(continued)

To turn in for grading: When the program is finished, list it by giving this command:

TYPE ALL

Execute the program for this list of numbers:

68
69
72
35
81
53
27
68
73
98

Turn in this part of the teletype paper to your instructor for grading, and then delete the program. (DELETE ALL)

5. Write a program that will approximate the sum of this series:

$$1, \frac{1}{\sqrt{2^2}}, \frac{1}{\sqrt{3^3}}, \frac{1}{\sqrt{4^4}}, \frac{1}{\sqrt{n^n}}, \ldots$$

To approximate the sum, compute successive partial sums until the last partial sum computed is equal to the preceding one, that is, until the n^{th} partial sum is equal to the $(n-1)^{st}$ partial sum. Report the $(n-1)^{st}$ partial sum, and the number of members of the series that were summed to arrive at that approximation.

To turn in for grading: When the program is finished, list it by giving this command:

TYPE ALL

Execute the program to demonstrate that it works correctly. Turn in this part of the teletype paper to your instructor for grading, and then delete the program. (DELETE ALL)

384

Appendix B

STUDENT EVALUATION FORM
COMPUTER-ASSISTED INSTRUCTION (CAI)

Please read each statement and circle the number on the scale that best describes your feelings.

SCALE

1	Strongly agree
2	Moderately agree
3	Slightly agree
4	Uncertain
5	Slightly disagree
6	Moderately disagree
7	Strongly disagree

1. I worked as hard answering questions in the computer lessons as I do in the classroom. 1 2 3 4 5 6 7

2. I learned from the computer lessons as well as I would have learned the same lesson in the classroom. 1 2 3 4 5 6 7

3. I like working at my own pace at the terminal. 1 2 3 4 5 6 7

4. I would prefer competing with my fellow students in the classroom rather than working at computer lessons. 1 2 3 4 5 6 7

5. Working with computer lessons is like having my own tutor. 1 2 3 4 5 6 7

6. Four hours a week is sufficient time to keep up with the course. 1 2 3 4 5 6 7

7. I found the computer lessons too easy. 1 2 3 4 5 6 7

8. I think working with computer lessons is an exciting way to learn. 1 2 3 4 5 6 7

9. I found working at the terminal more frustrating than worthwhile. 1 2 3 4 5 6 7

10. I would like to participate in another CAI course. 1 2 3 4 5 6 7

Appendix B (cont.)

11. I found the computer lessons too hard. 1 2 3 4 5 6 7

12. The CAI system provides the student with more feedback than classroom instruction. 1 2 3 4 5 6 7

13. Use the back of this sheet to make any comments you wish concerning the CAI program.

References

Friend, J. Computer-assisted instruction in programming: A curriculum description. Technical Report No. 211. Stanford: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Searle, B., Lorton, P., Jr., Goldberg, A., Suppes, P., Ledet, N., & Jones, C. Computer-assisted instruction program: Tennessee State University. Technical Report No. 198. Stanford: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Footnote

[1] The authors extend their appreciation to William J. Regan, Dean, College of Business Administration; Professor John Hoff, Chairman, Computer Science Department, University of San Francisco; and Professor Carl Grame, Chairman, Business and Data Processing Division, De Anza College, Cupertino, California.