COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING: AID

BY

JAMESINE FRIEND and R.C. ATKINSON

TECHNICAL REPORT NO. 164

JANUARY 25, 1971

PSYCHOLOGY SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

TECHNICAL REPORTS

PSYCHOLOGY SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

(Place of publication shown in parentheses; if published title is different from title of Technical Report, this is also shown in parentheses.)

(For reports no. 1 - 44, see Technical Report no. 125.)

50 R. C. Atkinson and R. C. Calfee. Mathematical learning theory. January 2, 1963. (In B. B. Wolman (Ed.), Scientific Psychology. New York: Basic Books, Inc., 1965. Pp. 254-275)

51 P. Suppes, E. Crothers, and R. Weir. Application of mathematical learning theory and linguistic analysis to vowel phoneme matching in Russian words. December 28, 1962.

52 R. C. Atkinson, R. Calfee, G. Sommer, W. Jeffrey and R. Shoemaker. A test of three models for stimulus compounding with children. January 29, 1963. (J. exp. Psychol., 1964, 67, 52-58)

53 E. Crothers. General Markov models for learning with inter-trial forgetting. April 8, 1963.

54 J. L. Myers and R. C. Atkinson. Choice behavior and reward structure. May 24, 1963. (Journal math. Psychol., 1964, 1, 170-203)

55 R. E. Robinson. A set-theoretical approach to empirical meaningfulness of measurement statements. June 10, 1963.

56 E. Crothers, R. Weir and P. Palmer. The role of transcription in the learning of the orthographic representations of Russian sounds. June 17, 1963.

57 P. Suppes. Problems of optimization in learning a list of simple items. July 22, 1963. (In Maynard W. Shelly, II and Glenn L. Bryan (Eds.), Human Judgments and Optimality. New York: Wiley. 1964. Pp. 116-126)

58 R. C. Atkinson and E. J. Crothers. Theoretical note: all-or-none learning and intertrial forgetting. July 24, 1963.

59 R. C. Calfee. Long-term behavior of rats under probabilistic reinforcement schedules. October 1, 1963.

60 R. C. Atkinson and E. J. Crothers. Tests of acquisition and retention, axioms for paired-associate learning. October 25, 1963. (A comparison of paired-associate learning models having different acquisition and retention axioms, J. math. Psychol., 1964, 1, 285-315)

61 W. J. McGill and J. Gibbon. The general-gamma distribution and reaction times. November 20, 1963. (J. math. Psychol., 1965, 2, 1-18)

62 M. F. Norman. Incremental learning on random trials. December 9, 1963. (J. math. Psychol., 1964, 1, 336-351)

63 P. Suppes. The development of mathematical concepts in children. February 25, 1964. (On the behavioral foundations of mathematical concepts. Monographs of the Society for Research in Child Development, 1965, 30, 60-96)

64 P. Suppes. Mathematical concept formation in children. April 10, 1964. (Amer. Psychologist, 1966, 21, 139-150)

65 R. C. Calfee, R. C. Atkinson, and T. Shelton, Jr. Mathematical models for verbal learning. August 21, 1964. (In N. Wiener and J. P. Schoda (Eds.), Cybernetics of the Nervous System: Progress in Brain Research. Amsterdam, The Netherlands: Elsevier Publishing Co., 1965. Pp. 333-349)

66 L. Keller, M. Cole, C. J. Burke, and W. K. Estes. Paired associate learning with differential rewards. August 20, 1964. (Reward and information values of trial outcomes in paired associate learning. (Psychol. Monogr., 1965, 79, 1-21)

67 M. F. Norman. A probabilistic model for free-responding. December 14, 1964.

68 W. K. Estes and H. A. Taylor. Visual detection in relation to display size and redundancy of critical elements. January 25, 1965, Revised 7-1-65. (Perception and Psychophysics, 1966, 1, 9-16)

69 P. Suppes and J. Donio. Foundations of stimulus-sampling theory for continuous-time processes. February 9, 1965. (J. math. Psychol., 1967, 4, 202-225)

70 R. C. Atkinson and R. A. Kinchla. A learning model for forced-choice detection experiments. February 10, 1965. (Br. J. math stat. Psychol., 1965, 18, 184-206)

71 E. J. Crothers. Presentation orders for items from different categories. March 10, 1965.

72 P. Suppes, G. Groen, and M. Schlag-Rey. Some models for response latency in paired-associates learning. May 5, 1965. (J. math. Psychol., 1966, 3, 99-128)

73 M. V. Levine. The generalization function in the probability learning experiment. June 3, 1965.

74 D. Hansen and T. S. Rodgers. An exploration of psycholinguistic units in initial reading. July 6, 1965.

75 B. C. Arnold. A correlated urn-scheme for a continuum of responses. July 20, 1965.

76 C. Izawa and W. K. Estes. Reinforcement-test sequences in paired-associate learning. August 1, 1965. (Psychol. Reports, 1966, 18, 879-919)

77 S. L. Biehart. Pattern discrimination learning with Rhesus monkeys. September 1, 1965. (Psychol. Reports, 1966, 19, 311-324)

78 J. L. Phillips and R. C. Atkinson. The effects of display size on short-term memory. August 31, 1965.

79 R. C. Atkinson and R. M. Shiffrin. Mathematical models for memory and learning. September 20, 1965.

80 P. Suppes. The psychological foundations of mathematics. October 25, 1965. (Colloques Internationaux du Centre National de la Recherche Scientifique. Editions du Centre National de la Recherche Scientifique. Paris: 1967. Pp. 213-242)

81 P. Suppes. Computer-assisted instruction in the schools: potentialities, problems, prospects. October 29, 1965.

82 R. A. Kinchla, J. Townsend, J. Yellott, Jr., and R. C. Atkinson. Influence of correlated visual cues on auditory signal detection. November 2, 1965. (Perception and Psychophysics, 1966, 1, 67-73)

83 P. Suppes, M. Jerman, and G. Groen. Arithmetic drills and review on a computer-based teletype. November 5, 1965. (Arithmetic Teacher, April 1966, 303-309.

84 P. Suppes and L. Hyman. Concept learning with non-verbal geometrical stimuli. November 15, 1965.

85 P. Holland. A variation on the minimum chi-square test. (J. math. Psychol., 1967, 3, 377-413).

86 P. Suppes. Accelerated program in elementary-school mathematics -- the second year. November 22, 1965. (Psychology in the Schools, 1966, 3, 294-307)

87 P. Lorenzen and F. Binford. Logic as a dialogical game. November 29, 1965.

88 L. Keller, W. J. Thomson, J. R. Tweedy, and R. C. Atkinson. The effects of reinforcement interval on the acquisition of paired-associate responses. December 10, 1965. (J. exp. Psychol., 1967, 73, 268-277)

89 J. I. Yellott, Jr. Some effects on noncontingent success in human probability learning. December 15, 1965.

90 P. Suppes and G. Groen. Some counting models for first-grade performance data on simple addition facts. January 14, 1966. (In J. M. Scandura (Ed.), Research in Mathematics Education. Washington, D. C.: NCTM, 1967. Pp. 35-43.

91 P. Suppes. Information processing and choice behavior. January 31, 1966.

92 G. Groen and R. C. Atkinson. Models for optimizing the learning process. February 11, 1966. (Psychol. Bulletin, 1966, 66, 309-320)

93 R. C. Atkinson and D. Hansen. Computer-assisted instruction in initial reading: Stanford project. March 17, 1966. (Reading Research Quarterly, 1966, 2, 5-25)

94 P. Suppes. Probabilistic inference and the concept of total evidence. March 23, 1966. (In J. Hintikka and P. Suppes (Eds.), Aspects of Inductive Logic. Amsterdam: North-Holland Publishing Co., 1966. Pp. 49-65.

95 P. Suppes. The axiomatic method in high-school mathematics. April 12, 1966. (The Role of Axiomatics and Problem Solving in Mathematics. The Conference Board of the Mathematical Sciences, Washington, D. C. Ginn and Co., 1966. Pp. 69-76.

(Continued on inside back cover)

COMPUTER-ASSISTED INSTRUCTION IN PROGRAMMING: AID

by

Jamesine Friend and R. C. Atkinson

TECHNICAL REPORT 164

January 25, 1971

PSYCHOLOGY SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

TABLE OF CONTENTS

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.

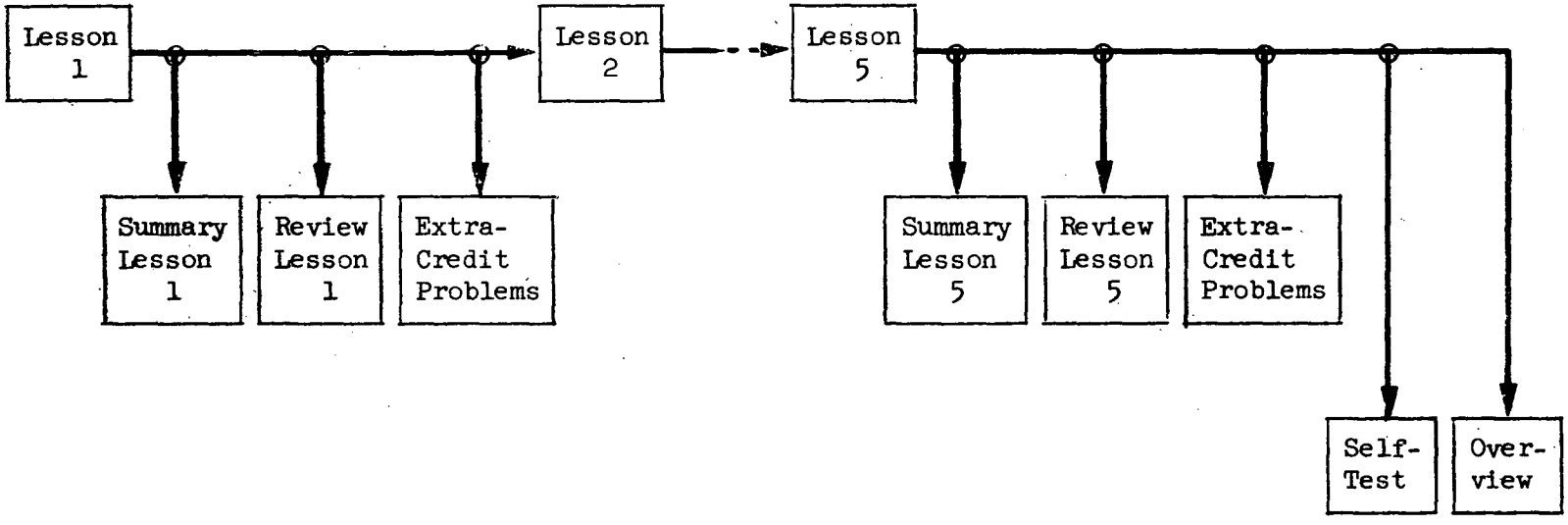
The structure of the course is illustrated in Figure 2 by a block diagram of a set of five lessons (with summaries, reviews and extra-credit problems), followed by a self-test and an overview lesson.

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 a 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

$$5 * 2\uparrow 3$$

(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

$$32/4\uparrow 2$$

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

$$10\uparrow 3 * 2.$$

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 and in 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.

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.

****

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.

****

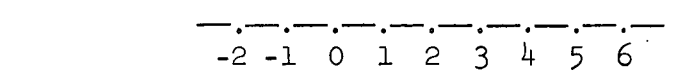
NOT
See Propositions.

****

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:



****

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.

****

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.

****

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.

****

APPENDIX B

AID Documentation

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, non-parenthetical 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

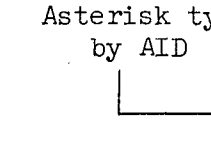
AID indicates that it is ready to receive input by typing out an asterisk (*). Characters may be deleted sequentially backward by striking the RUBOUT key. Typing asterisk (*) at the beginning or end of an input line cancels the line.

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 ≥ 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 ≤ c) permitted. Number sign for not equal.

LOGICAL OPERATIONS:

AND OR NOT

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

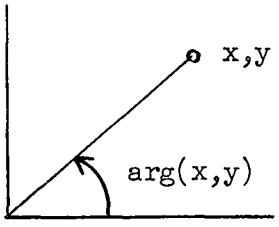
3 + 1/2 + 1/4.5	=	3 + (1/2 + (1/4·5))
-2↑3*4-5	=	$(-(2^3)\cdot 4)-5$
2**3**4	=	$(2^3)^4$
A OR B AND NOT C OR D	=	a or (b and not c) or d

BASIC FUNCTIONS

SQRT(X)	square root, x ≥ 0
SIN(X)	|x in radians| < 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 |true|= 1, |false|=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) =      .577350268
       T(2) =     -.577350268

*
```

APPENDIX C

Excerpts from the Coders' Manual

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.

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

APPENDIX D

Sample Coded Problem

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
```

(Continued from inside front cover)

96 R. C. Atkinson, J. W. Brelsford, and R. M. Shiffrin. Multi-process models for memory with applications to a continuous presentation task. April 13, 1966. (J. math. Psychol., 1967, 4, 277-300).

97 P. Suppes and E. Crothers. Some remarks on stimulus-response theories of language learning. June 12, 1966.

98 R. Bjork. All-or-none subprocesses in the learning of complex sequences. (J. math. Psychol., 1968, 1, 182-195).

99 E. Gammon. The statistical determination of linguistic units. July 1, 1966.

100 P. Suppes, L. Hyman, and M. Jerman. Linear structural models for response and latency performance in arithmetic. In J. P. Hill (ed.), Minnesota Symposia on Child Psychology. Minneapolis, Minn.: 1967. Pp. 160-200).

101 J. L. Young. Effects of intervals between reinforcements and test trials in paired-associate learning. August 1, 1966.

102 H. A. Wilson. An investigation of linguistic unit size in memory processes. August 3, 1966.

103 J. T. Townsend. Choice behavior in a cued-recognition task. August 8, 1966.

104 W. H. Batchelder. A mathematical analysis of multi-level verbal learning. August 9, 1966.

105 H. A. Taylor. The observing response in a cued psychophysical task. August 10, 1966.

106 R. A. Bjork. Learning and short-term retention of paired associates in relation to specific sequences of interpresentation intervals. August 11, 1966.

107 R. C. Atkinson and R. M. Shiffrin. Some Two-process models for memory. September 30, 1966.

108 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the third year. January 30, 1967.

109 P. Suppes and I. Rosenthal-Hill. Concept formation by kindergarten children in a card-sorting task. February 27, 1967.

110 R. C. Atkinson and R. M. Shiffrin. Human memory: a proposed system and its control processes. March 21, 1967.

111 Theodore S. Rodgers. Linguistic considerations in the design of the Stanford computer-based curriculum in initial reading. June 1, 1967.

112 Jack M. Knutson. Spelling drills using a computer-assisted instructional system. June 30, 1967.

113 R. C. Atkinson. Instruction in initial reading under computer control: the Stanford Project. July 14, 1967.

114 J. W. Brelsford, Jr. and R. C. Atkinson. Recall of paired-associates as a function of overt and covert rehearsal procedures. July 21, 1967.

115 J. H. Stelzer. Some results concerning subjective probability structures with semiorders. August 1, 1967

116 D. E. Rumelhart. The effects of interpresentation intervals on performance in a continuous paired-associate task. August 11, 1967.

117 E. J. Fishman, L. Keller, and R. E. Atkinson. Massed vs. distributed practice in computerized spelling drills. August 18, 1967.

118 G. J. Groen. An investigation of some counting algorithms for simple addition problems. August 21, 1967.

119 H. A. Wilson and R. C. Atkinson. Computer-based instruction in initial reading: a progress report on the Stanford Project. August 25, 1967.

120 F. S. Roberts and P. Suppes. Some problems in the geometry of visual perception. August 31, 1967. (Synthese, 1967, 17, 173-201)

121 D. Jamison. Bayesian decisions under total and partial ignorance. D. Jamison and J. Kozielecki. Subjective probabilities under total uncertainty. September 4, 1967.

122 R. C. Atkinson. Computerized instruction and the learning process. September 15, 1967.

123 W. K. Estes. Outline of a theory of punishment. October 1, 1967.

124 T. S. Rodgers. Measuring vocabulary difficulty: An analysis of item variables in learning Russian-English and Japanese-English vocabulary parts. December 18, 1967.

125 W. K. Estes. Reinforcement in human learning. December 20, 1967.

126 G. L. Wolford, D. L. Wessel, W. K. Estes. Further evidence concerning scanning and sampling assumptions of visual detection models. January 31, 1968.

127 R. C. Atkinson and R. M. Shiffrin. Some speculations on storage and retrieval processes in long-term memory. February 2, 1968.

128 John Holmgren. Visual detection with imperfect recognition. March 29, 1968.

129 Lucille B. Mlodnosky. The Frostig and the Bender Gestalt as predictors of reading achievement. April 12, 1968.

130 P. Suppes. Some theoretical models for mathematics learning. April 15, 1968. (Journal of Research and Development in Education, 1967, 1, 5-22)

131 G. M. Olson. Learning and retention in a continuous recognition task. May 15, 1968.

132 Ruth Norene Hartley. An investigation of list types and cues to facilitate initial reading vocabulary acquisition. May 29, 1968.

133 P. Suppes. Stimulus-response theory of finite automata. June 19, 1968.

134 N. Moler and P. Suppes. Quantifier-free axioms for constructive plane geometry. June 20, 1968. (In J. C. H. Gerretsen and F. Oort (Eds.), Compositio Mathematica. Vol. 20. Groningen, The Netherlands: Wolters-Noordhoff, 1968. Pp. 143-152.)

135 W. K. Estes and D. P. Horst. Latency as a function of number or response alternatives in paired-associate learning. July 1, 1968.

136 M. Schlag-Rey and P. Suppes. High-order dimensions in concept identification. July 2, 1968. (Psychom. Sci., 1968, 11, 141-142)

137 R. M. Shiffrin. Search and retrieval processes in long-term memory. August 15, 1968.

138 R. D. Freund, G. R. Loftus, and R.C. Atkinson. Applications of multiprocess models for memory to continuous recognition tasks. December 18, 1968.

139 R. C. Atkinson. Information delay in human learning. December 18, 1968.

140 R. C. Atkinson, J. E. Holmgren, and J. F. Juola. Processing time as influenced by the number of elements in the visual display. March 14, 1969.

141 P. Suppes, E. F. Loftus, and M. Jerman. Problem-solving on a computer-based teletype. March 25, 1969.

142 P. Suppes and Mona Morningstar. Evaluation of three computer-assisted instruction programs. May 2, 1969.

143 P. Suppes. On the problems of using mathematics in the development of the social sciences. May 12, 1969.

144 Z. Domotor. Probabilistic relational structures and their applications. May 14, 1969.

145 R. C. Atkinson and T. D. Wickens. Human memory and the concept of reinforcement. May 20, 1969.

146 R. J. Titiev. Some model-theoretic results in measurement theory. May 22, 1969.

147 P. Suppes. Measurement: Problems of theory and application. June 12, 1969.

148 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the fourth year. August 7, 1969.

149 D. Rundus and R.C. Atkinson. Rehearsal in free recall: A procedure for direct observation. August 12, 1969.

150 P. Suppes and S. Feldman. Young children's comprehension of logical connectives. October 15, 1969.

(Continued on back cover)

(Continued from inside back cover)

151 Joaquim H. Laubsch. An adaptive teaching system for optimal item allocation. November 14, 1969.

152 Roberta L. Klatzky and Richard C. Atkinson. Memory scans based on alternative test stimulus representations. November 25, 1969.

153 John E. Holmgren. Response latency as an indicant of information processing in visual search tasks. March 16, 1970.

154 Patrick Suppes. Probabilistic grammars for natural languages. May 15, 1970.

155 E. Gammon. A syntactical analysis of some first-grade readers. June 22, 1970.

156 Kenneth N. Wexler. An automaton analysis of the learning of a miniature system of Japanese. July 24, 1970.

157 R. C. Atkinson and J. A. Paulson. An approach to the psychology of instruction. August 14, 1970.

158 R.C. Atkinson, J.D. Fletcher, H.C. Chetin, and C. M. Stauffer. Instruction in initial reading under computer control: the Stanford project. August 13, 1970.

159 Dewey J. Rundus. An analysis of rehearsal processes in free recall. August 21, 1970.

160 R.L. Klatzky, J.F. Juola, and R.C. Atkinson. Test stimulus representation and experimental context effects in memory scanning.

161 William A. Rottmayer. A formal theory of perception. November 13, 1970.

162 Elizabeth Jane Fishman Loftus. An analysis of the structural variables that determine problem-solving difficulty on a computer-based teletype. December 18, 1970.

163 Joseph A. Van Campen. Towards the automatic generation of programmed foreign-language instructional materials. January 11, 1971.

164 Jamesine Friend and R.C. Atkinson. Computer-assisted instruction in programming: AID. January 25, 1971.