A CAI System for Basic

Harry V. Aharonian Jr.

University of Rhode Island
A CAI SYSTEM FOR BASIC

BY

HARRY V. AHARONIAN, JR.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

1970
MASTER OF SCIENCE THESIS
OF
HARRY V. AHARONIAN, JR.

Approved:

Thesis Committee:
Chairman

Dean of the Graduate School

UNIVERSITY OF RHODE ISLAND
1970
ABSTRACT

Computer-assisted instruction is a means of teaching a student by interaction with a computer through a remote console. This thesis will discuss teaching BASIC, a simple programming language, through computer-assisted instruction. The methods of instruction have been carefully planned for and tested with high school students, many of whom have had no computer programming experience.

The project involves three phases. The first phase introduces the student to BASIC by means of a self-instructional manual containing all the fundamentals one needs to know to program a fairly sophisticated problem in BASIC. Contained in the manual are a series of questions which the student should answer to check his progress. Having finished this manual and studied the illustrative examples in the manual, he is now ready to be examined in this the second phase. A programmed examination is used to test each student; the questions for the exam are chosen at random from a bank of questions. The third phase involves problems stored in the computer which contain errors and deletions that the student must locate and correct. This third phase tests the student's programming ability, whereas, the programmed examination tests his understanding of various aspects of the language. The last two phases are more specifically classified as computer-managed instruction (CMI).
The results, after having implemented this three phase plan, were quite gratifying having used high school students from many different schools in Rhode Island. All the students were able to solve the suggested problems at the end of the self-instructional manual which clearly indicates that the primary objective had been reached; i.e., to teach a beginner how to program an intelligent problem in BASIC. Students also learned good programming techniques, such as conservation of storage; however, this accomplishment would be most difficult to measure as it varied from student to student. The project was also implemented in a short course for high school teachers with much the same successful results. All who participated enjoyed the CAI method of instruction.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. PURPOSE OF STUDY</td>
<td>6</td>
</tr>
<tr>
<td>III. EXPLANATION OF CAI SYSTEM</td>
<td>9</td>
</tr>
<tr>
<td>IV. EXPERIENCES WITH CAI</td>
<td>18</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>21</td>
</tr>
</tbody>
</table>

**APPENDIX A:** INSTRUCTIONS FOR PREPARING PROGRAMS

**APPENDIX B:** SELF INSTRUCTIONAL MANUAL

**APPENDIX C:** PROGRAMMED EXAMINATION LISTING WITH QUESTIONS AND SAMPLE EXAMS

**APPENDIX D:** GENERATING PROGRAM FOR UNSOLVED PROBLEMS WITH ORIGINAL PROBLEMS AND SAMPLE EXERCISES

**APPENDIX E:** PROGRAM TO SEPARATE PREPARED UNSOLVED PROBLEMS
I. INTRODUCTION

Computer-assisted instruction (CAI) is a far reaching entity in today's computer world involving many people working on many diverse projects. For example, in a booklet titled "Feasibility Study and Recommended Plan for Establishing an Institute for Information Systems in Higher Education" prepared by Associated Universities Inc. for the National Science Foundation (1), some 74 projects were listed in CAI ranging from "computer assisted teaching of mathematics" to "medical training simulations for anesthesiologists." To define CAI would be most difficult due to the many definitions already present for it. Peingold in his discussion of PLANIT, a CAI language (6), defines CAI in a most general way by saying it is a "means using the computer to control or monitor the presentation of some portion of information". CAI is an attempt at individualized instruction through automation, a process emphasized in (3) as being a "means of assisting the teacher" with no mention made of CAI actually replacing the teacher. The term "CAI" is sometimes used loosely and may mean a number of different things. Frye, in his discussion of choosing a computer language for communication in (8), clarifies this ambiguity by outlining three major categories that the CAI field may be subdivided into:
1) Problem-solving
2) Tutorial instruction
3) Computer-managed instruction

The first category, "problem solving", utilizes the computer to solve problems for a user by means of an interactive facility. "Tutorial instruction" employs the computer as a teacher, presenting the student with new material only when he has satisfactorily learned previous material. The interaction takes place between the student and computer only. When the word "CAI" is used, most often it is in reference to this particular application. Finally, "computer-managed instruction (CMI)" deals with the computer processing and analyzing data for the teacher with less emphasis being placed on learning directly through lessons printed at a console.

In such a learning environment, the student learns at his own rate and as mentioned in (15), the feedback to the student is almost immediate, whereas it is much longer in conventional teaching methods. Suppes in discussing some of his projects in (17), which include

1.) drill-and-practice program in elementary mathematics
2.) tutorial mathematics program for grade 2
3.) tutorial reading program for grade 1
4.) tutorial logic and algebraic programs grades 5 to 8

mentions tests and exercises were produced for students by
the computer, these tests being commensurable to the student's ability. These projects involved use of remote terminals located in California, Mississippi, and Kentucky. Understanding the problems in education today, Suppes is quick to mention in (18) that this testing by the computer is an attempt at decreasing the burden of paperwork on the teacher. CMI is a very important aspect of CAI because its purpose is to account for the results of the examinations. In a discussion of individualized instruction in *Science* (3), the authors emphasize that CMI must be able to collect and analyze the data on request in addition to storing it. It is not economically feasible to have a computer do the job of a clerk. In this article, the "Oakleaf School project" is cited as an example of a CMI type system. The project involved use of the University of Pittsburgh's IBM 360 model 50 with terminals tied into the 360 system by leased-lines. After a student finishes a unit of instruction, his performance is stored on a scratch file on disk. At the end of the day, this information is run with previously stored information concerning each student and an updating process takes place with the results stored back again on tape. As a result, the teacher is able to request this information regarding the progress of each student at any time.

International Business Machines (IBM) is one computer manufacturer who became involved in CAI about 1964 with the purchase of Science Research Associates Inc. In 1966, IBM implemented the IBM 1500 CAI System. IBM has also developed
a software package called "COURSEWRITER", a language suitable to their CAI needs (4). Studies have since been made by IBM at Yorktown Heights, New York using the IBM 7010 (5). Today, IBM is still involved in CAI. For example, the University of Rhode Island Computer Laboratory recently installed an IBM 2314 disk drive. The IBM customer engineer assigned to the University attended a school to learn about the IBM 2314. The method of instruction was CAI. The engineer worked strictly with remote terminals spending approximately 32 hours on the course, working at his own rate. To save time at the terminal, he was issued a manual to read instead of having the terminal reproduce the manual for him, a very time consuming and costly operation.

The University of Illinois has developed the PLATO system mentioned in (5). The system primarily involves the simulation of a classroom. PLATO is used both as an instructional tool (CAI) and a data processing tool (CMI). Currently, there are criticisms of CAI involving its high cost. Yet, Professor Bitzer, Director of the Computer-Based Education Research Laboratory at the University of Illinois, claims in the 1970's, PLATO will cost approximately 27¢ per student hour (2). It should be noted that, as mentioned in (1), the cost of computers is decreasing, whereas the cost of humans is increasing.

The study of foreign languages is another area of interest in CAI. Suppes feels the study of foreign languages through CAI is important and comments on the fact in (17)
that Russian is usually more difficult for Americans to learn than German, French, or Spanish. He concludes that individualized instruction would be an attempt at correcting this.

Other computer manufacturers have become interested in CAI. General Electric is one manufacturer making use of teletypewriters (10). In their library of programs related to foreign language CAI programs, are the following which deal with French (9):

- FRPROG1 Produces questions at random
- FRPROG2 Questions sequentially
- FRVOC-1 Vocabulary drill in French
- FRVERB-1 Verb drills in French

The complete list of projects and participants in CAI is much too long to mention here; however, a great deal of work still needs to be explored. Some CAI researchers feel that CAI can not thrive on research alone but must be implemented and tested in a real-world environment. It is the opinion of Kanner in (13) that "there should also be the recognition that favorable research results are only a preliminary to many long years of adaptation and acceptance within current educational practices."
II. PURPOSE OF STUDY

The purpose of this study is to teach through computer-assisted instruction the programming language BASIC, an interactive computing language developed at Dartmouth University by Professors John G. Kemeny and Thomas E. Kurtz for the GE-265 TSS (time-sharing system) (7) (8). The authors note that the name "BASIC" (Beginner's All-purpose Symbolic Instruction Code) is a misnomer because it is being used by experts as well as novices. BASIC is relatively easy to learn as proven by Kemeny and Kurtz who at Dartmouth taught the freshman class how to program BASIC in two one-hour lectures (14).

In order to best accomplish the purpose of this study, a complete CAI system was designed with the use of the components of a CAI system mentioned in Chapter I. The format for the system calls for a three phase plan. The initial phase involves the student reading a self-instructional manual discussing the ideas and concepts of BASIC. The manual is designed to have questions after most sections to insure the understanding of the reader. This technique is analogous to the objectives of "tutorial instruction" mentioned in Chapter I. The purpose of the manual is not to have the remote terminal produce a manual for each student. The reasons being that the time spent on typing the manual
is great and the cost of computer time is rather high. At the conclusion of the manual are suggested problems for the student to try his problem-solving ability.

The second and third phases of the CAI system involve testing and recording student data by means of CMI. The second phase is a programmed examination covering material one needs to know in order to program in BASIC, whereas, the third phase has the student complete a program which is presented to him with errors and deletions. These two phases can be employed off-line as well as on-line.

For phase two, questions are chosen at random from a bank of questions which were supplied as data. The number of questions to be asked on the exam is a variable parameter which can be controlled by whoever is preparing the examinations. Once the student makes a response and enters it, it is checked and, if correct, another question is presented. If on the first try an incorrect response is made, a second answer is allowed; however, if the correct answer has not been given after the second try, the correct answer is then given. At the end of the examination, a summary is given as to the number of questions answered correctly the first time, the number answered correctly the second time, and finally, those questions to which the correct answer had to be given. A proficiency rating follows as to whether the student performed "excellent", "good", "fair", or "poor". Thus, the exams are graded for the teacher. A routine is incorporated to
provide the teacher with information about how each question in the library was answered.

The final phase of the CAI project is to have the student "debug" programs presented to him on the remote console. He is to examine the coding (either on-line or off-line) and decide what changes need to be made. There are various types of changes the student must make. These changes are described in the next chapter. Once the changes are made, the student checks his work by running the corrected program and examining the results. It is very possible that he must make further corrections due to his having missed an error or due to one of his corrections being incorrect. There are a number of these programs which may be prepared with varying complexity. (See Appendix A).

This three phase project not only teaches a beginner BASIC but tests his proficiency in it by examination and program "debugging" with the results passed along to the teacher. The routine that records the responses for each question is of great help to the teacher who prepared the questions. A question answered incorrectly all the time may be too difficult or worded incorrectly. Similarly, one that is answered correctly all of the time is of little value.
The self-instructional manual entitled, "Self Instruction for BASIC Programming" was written primarily for the junior and senior high school student wanting to learn the elementary ideas and concepts of a relatively simple programming language. The format of the manual provides for questions at the conclusion of most sections in order to check the understanding of the reader. Following the questions are a list of answers and explanations to these questions. This manner of teaching is similar in purpose although different in format to (11), (12), and (16).

Chapter I deals with symbols, constants, and variables. The symbols discussed are both the equality and inequality symbols as well as all the arithmetic symbols that the BASIC system uses. The section dealing with constants describes what the BASIC system recognizes as a valid number. Similarly, the section on variables describes rules to follow when forming variables and illustrates valid and invalid variables.

Chapter II deals entirely with statements, i.e., commands which one can use in BASIC. Beginning with the format of a statement, each statement is handled in its own section with questions appearing after most sections. The question-
aspect is very crucial since material discussed in one section may depend on material appearing in a previous section. Simple problems are coded for illustration purposes. Having completed this chapter, the student should be able to understand and use the BASIC commands.

Chapter III lists the arithmetic and trigonometric functions which are available to the BASIC user. These include sine, cosine, and tangent of an angle, as well as square root, absolute value, and sign of a number. These functions are illustrated in sample programs.

Chapter IV contains five problems which again were chosen so as not to present much difficulty to the reader. Following a description of the problem is the solution programmed in BASIC as a student might solve it. This is not the only solution!

For completeness, Appendices A and B of the self-instructional manual contain a list of matrix operations available and suggested problems, respectively. The matrix operations are not discussed since most students at this level are not familiar with matrices.

The second phase is a programmed examination which presently chooses ten questions at random from a bank of forty questions. Both these parameters may be altered if desired. Each question has a corresponding solution statement which may or may not have to be supplied to the student and a list of possible answers for each question. These answers are presented in multiple choice form. A key is
also supplied with the questions indicating where the correct solutions lie. Each examination generates a random key in addition to choosing questions at random. For example, the answer to the question chosen to be the fifth question may be number one.

5. WHICH IS THE CONVERSATIONAL READ STATEMENT IN BASIC

1) INPUT 2) DATA 3) READ 4) PRINT

If the random key does not indicate the solution for question five to be a "one", then the correct response is moved to the prescribed location by exchanging choices. If the random key chooses number "three", then the question will appear as follows:

5. WHICH IS THE CONVERSATIONAL READ STATEMENT IN BASIC

1) READ 2) DATA 3) INPUT 4) PRINT

Should the fourth choice be "none of these", it is never moved from this fourth position.

After each question, the student enters the response "1, 2, 3, or 4" corresponding to his choice. If any other character is entered, a message will appear indicating only a "1, 2, 3, or 4" is a valid response. If the choice is correct, the next question will appear; otherwise, the student is allowed a second try. This second try may not be the same response given the first time for the program checks to make sure two separate choices are made. If the second choice is also incorrect, the correct answer to the question will appear.
A summary is given at the end of the examination indicating how many questions were answered correctly the first time, how many the second time, and how many were answered unsuccessfully. A record is kept as to how each question was answered, allowing the instructor to view the data at his request and evaluate the questions in the examination. Following the summary is a proficiency rating indicating how well the student performed.

Should a student desire to repeat the examination, ten questions will again be chosen at random along with a new key. It is likely that a question or questions may appear that the student has already seen but, he will probably find the correct answer to be another choice. Expanding the bank of forty questions would lessen the chances of reoccurrence of questions.

The third and final phase is to have the student attempt to correct a given program which has a variable number of errors and deletions in it. The programs chosen are not very difficult, and involve basically the same ideas as the problems the student tried in Appendix B of the self-instructional manual. Appendix A outlines the method for preparing these problems.

The BASIC statements are classified as follows:

CLASS I LET GOTO IF PRINT
CLASS II FOR NEXT READ DIM
CLASS III DATA INPUT
The types of deletions which may occur are:

**TYPE I** ENTIRE LINE IS DELETED

**TYPE II** ALL INFORMATION EXCEPT KEY WORD IS DELETED

**TYPE III** KEY WORD IN LINE IS DELETED

Presently six programs exist and are prepared simultaneously with the following input parameters available to the user preparing the programs. These parameters allow one to vary the complexity of the problems.

- **MAX** - MAXIMUM NUMBER OF ERRORS PER PROGRAM
- **JUNK** - MAXIMUM NUMBER OF ERROR STATEMENTS TO BE INSERTED
- **CLASS1** - NUMBER OF CLASS I STATEMENTS TO BE DELETED
- **CLASS2** - NUMBER OF CLASS II STATEMENTS TO BE DELETED
- **CLASS3** - NUMBER OF CLASS III STATEMENTS TO BE DELETED
- **TYPE1** - NUMBER OF LINES TO BE COMPLETELY BLANKED
- **TYPE2** - NUMBER OF LINES WITH JUST KEY WORD APPEARING
- **TYPE3** - NUMBER OF LINES WITH KEY WORD MISSING

The user inputs values for these parameters conversationally when he is preparing the program. (See Appendix A). The number "zero" is a valid designation for any of the parameters.

Class III statements are handled first. The parameter "Class3" causes a random process to begin whereby as many
Class III statements are generated as specified by the variable "Class3". Each of the Class III statements generated are scanned for in each of the six programs sequentially. When the particular statement is encountered, the entire line is blanked, i.e., TYPE1 change. All Class III statements are completely blanked out this way.

Class II statements are considered next. As was the case for Class III, randomly, statements of Class II are generated. Each program is scanned for the Class II statement at hand and the changes made are in the order of Type III, Type II, and finally Type I. If a particular statement scanned for appears more than once, a random process is employed to select one. For example, if the Class II statement generated were "READ", the statement found in a program may be:

12 READ N

If the TYPE3 parameter was not zero or had not been exhausted, the statement would be changed to look like:

12 N

If the TYPE3 parameter was not available, the TYPE2 parameter would be considered and if possible the change would be:

12 READ

If neither of these types are available, the entire line is omitted leaving just the statement number:

12

Class I statements are considered last and are handled exactly the same as Class II statements.
After the three classes of statements have been implemented according to the specified type changes, the maximum error per program parameter may not be satisfied in all programs. Incorporated as data to the master program is a list of BASIC statements which are correct but if inserted in any of the six programs will cause errors to occur if undetected. The final parameter to be discussed is one indicating the number of these lines to be inserted in the preparation of the six programs.

If the number of irrelevant statements to be inserted is four, then four random numbers are generated between one and six, each representing one of the problems. If the random numbers generated were "1, 6, 2, 1," this would mean program "one" will get two lines replaced if possible and program "two" and "six", one each if possible. Otherwise, no replacement takes place.

It may be that just program "six" has not reached the maximum change parameter. If so, one line is chosen at random from the data and all the statements like this one in program "six" are considered likely candidates to be replaced. If the statement chosen was a "GOTO" statement, program six may contain three "GOTO" statements. One is chosen at random and replaced by the new "GOTO".

```
GOTO 17 (GOTO chosen from list of statements randomly)
GOTO 12 )
GOTO 19 ) All located in program six
GOTO 24 )
GOTO 24 Chosen as one to be replaced by GOTO 17
```
Now statement "GOTO 24" is replaced by "GOTO 17" thus causing an improper branch. All lines inserted causing errors are handled this way.

Finally, a check is made to see if the six programs have reached their maximum number of changes. If not, a scan is made for "IF" and "FOR" statements in these programs where another change can be made. The equality/inequality symbol is removed from the "IF" statement and the starting, ending, and incrementing values from the "FOR" statement.

For example:

16  IF  K=N  THEN  12  (Original)
16  IF  K  N  THEN  12  (Change)
18  FOR  I=1  TO  N+1  STEP  2  (Original)
18  FOR  I=  TO  STEP  (Change)

If at any time there is more than one candidate for a change in one program, one will be chosen at random. Suppose an "IF" statement is chosen as an irrelevant statement to be inserted into program five. If program five has not reached the maximum number of changes, it is possible that this line will become part of program five and furthermore, the equality/inequality symbol stripped off.

15  IF  I+1<K  THEN  20  (Original "IF" statement)
    IF  I<=K  THEN  18  ("IF" chosen to be inserted replacing above)
15  IF  I<=K  THEN  18  (New line 15)
15  IF  I  K  THEN  18  (Program allows "IF" to be altered)
Note that the number of spaces constituting a blank do not necessarily indicate the number of characters missing. Great pains have been taken to insure that variables in these line insertion statements appear in all six programs, thus making their detection more challenging.

This third phase tests the student's understanding of a problem and its solution in BASIC. The problems, simple in nature, can be made very challenging.
IV. EXPERIENCES WITH CAI

For ten weeks during the summer of 1969, this author had the opportunity to work with high school students from all parts of Rhode Island, teaching them EASY (Elementary Assembler and Machine System) and FORTRAN IV. This program was part of the National Science Foundation’s Cooperative College-School Science program. Support for this high school program was through a National Science Foundation grant (GW-4150) and the University of Rhode Island Computer Laboratory. Most students were beginners having had no previous computer science background while others had a semester course in programming at their respective high schools. Much insight was gained into the capability of today’s high school student and what type of mathematical training he has. It was interesting to note that the beginners did almost as well as others with some experience since the course was taught at the beginner’s level. This is not to say that those with experience were bored; rather, they were able to add to their previous knowledge. The awareness of these students and their high retention rate was also observed. It was these observations which led to the development of this CAI system of teaching BASIC to high school students.

During the spring semester of the 1969-70 school year,
the University of Rhode Island Computer Laboratory and the National Science Foundation grant (GW-5111) provided funds for a short course to high school teachers. These short courses were taught at South Kingstown, Middletown, and Portsmouth High Schools, one hour a week for four weeks. The text used in the course was the self-instructional manual of this project. Most teachers were able to write programs by the second week. In their attempts to learn BASIC, they provided excellent criticisms of the manual which they were using. These helpful criticisms led to a later revision of the manual. The courses were successful since those completing it were able to program fairly well by the end of the fourth week.

Another workshop for high school students was offered again during the summer of 1970 sponsored by the latest National Science Foundation grant and University of Rhode Island Computer Laboratory. The students in this program were given an opportunity to use the entire CAI system to learn BASIC. Due to a time factor, the teachers participating in the short courses were only exposed to the self-instructional manual.

The students were allowed one day to read the manual. Once they finished and their questions answered, they were assigned the eight problems in Appendix B of the manual. The solutions to these problems raised more questions from the students and those questions were also resolved. Now, having done some programming, the students were ready for the second and third phases of the system.
The programmed examinations, as well as the unsolved problems, were a challenge to many. It was interesting to view the students who were being tested in a new way. Criticisms drawn from the class at the end of the program indicated that they were very much in favor of being taught by a self-instructional manual and tested at a remote terminal. This CAI project added to the success of the workshop whose prime objective was to teach BASIC. This objective was more than met.
LIST OF REFERENCES


APPENDIX A

INSTRUCTIONS FOR PREPARING PROGRAMS
APPENDIX A: INSTRUCTIONS FOR PREPARING PROGRAMS

In order to prepare an examination for a student, all one needs to do under the University of Rhode Island Computer Laboratory RAX System is to include the name "BASICX" in an input file and run this file in the following manner. Type in the commands:

```
/INPUT
/INCLUDE BASICX
/INSERT KEEP
/END
```

After the system types "BEGIN ACTIVITY", type
```
/PURGE KEEP (LOCK CODE)
```
where the "LOCK CODE" is a four digit number designated for each user of the system. The file "KEEP" is purged so that a new one can be saved after the examination. Initially input a file "KEEP" of forty blank lines. This file is for accounting purposes so as to keep track of how students respond to each question.

After the "BEGIN ACTIVITY" message, type
```
/RUN
```

Now the directions for the exam will be typed out on the console. Each of the ten questions will be typed out with their appropriate choices. When a question is asked, an underline character is typed out indicating the keyboard is
locked and the student may enter his choice for this particular question (see Appendix C). After the examination, the message "BEGIN ACTIVITY" will appear again. Type 

/SAVE KEEP(LOCK CODE),SV

in order to save the file of question responses.

The programmed problems appear in the system solved correctly under the file name of "ORGPGS" (original programs). Presently, there are six different problems in this file. The program which causes changes to occur in these programs is called "PREP" (see Appendix D). Thus to prepare the six problems, type

/INPUT
/INCLUDE PREP
/INCLUDE ORGPGS
/END RUN

Now the user must supply various parameters by means of the console on request. The parameters include maximum number of errors per program, number of irrelevant lines to be inserted in the preparation of the programs (both I2 formats), number of each class of statements to be changed (312 format), and the number of each type of change (3I2 format). When "BEGIN ACTIVITY" appears, save the prepared work which is on a work file in some file of your choosing (suggest ALLSET). To separate this file which contains all six programs joined together, call a program "SLICE" which will take each file and save it on a work file for the user to save under an appropriate name (see Appendix E).
The user must supply the first and last line numbers of the file he is extracting. Type

/INPUT
/INCLUDE SLICE
/INCLUDE ALLSET
/END RUN

after which will appear an underline symbol allowing the user to type in the two limits (both I3 format). Repeat this process for each of the six programs till each is saved under its own name.
APPENDIX B

SELF INSTRUCTIONAL MANUAL
SELF INSTRUCTION

FOR

BASIC

PROGRAMMING

HARRY V. AHARONIAN, JR.

REVISED EDITION

JUNE 1970
Preface

This text is designed to teach a beginning programmer the basic concepts and ideas of the programming language BASIC (Beginner's All-purpose Symbolic Instruction Code). A relatively new language, BASIC was designed by Professors John G. Kemeny and Thomas E. Kurtz of Dartmouth University for application to computing environments supporting remote consoles. The purpose of this manual is to instruct one interested in programming the basic principles and statements of the BASIC language needed to solve a problem via programming methods.

In addition to this text, there is available programmed examinations to test the knowledge of the student. Questions for the examination are chosen at random to insure different exams for each student. Once the exam is completed, a report is given indicating how well the student performed. Information concerning examinations may be obtained through the author of this manual.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1</th>
<th>Symbols/Constants/Variables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Symbols</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Constants</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Variables</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 2</th>
<th>Statements</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Statements</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>READ</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>DATA</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>RESTORE</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>PRINT</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>LET</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>GOTO</td>
<td>13</td>
</tr>
<tr>
<td>2.8</td>
<td>IF</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>FOR</td>
<td>15</td>
</tr>
<tr>
<td>2.10</td>
<td>NEXT</td>
<td>16</td>
</tr>
<tr>
<td>2.11</td>
<td>END</td>
<td>17</td>
</tr>
<tr>
<td>2.12</td>
<td>INPUT</td>
<td>18</td>
</tr>
<tr>
<td>2.13</td>
<td>DEF</td>
<td>20</td>
</tr>
<tr>
<td>2.14</td>
<td>STOP</td>
<td>21</td>
</tr>
<tr>
<td>2.15</td>
<td>REM</td>
<td>22</td>
</tr>
<tr>
<td>2.16</td>
<td>DIM</td>
<td>22</td>
</tr>
<tr>
<td>2.17</td>
<td>GOSUB</td>
<td>23</td>
</tr>
<tr>
<td>2.18</td>
<td>RETURN</td>
<td>23</td>
</tr>
<tr>
<td>CHAPTER 3</td>
<td>Functions</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.1</td>
<td>Built-In Functions</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Numeric Applications</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 4</th>
<th>Problems and Solutions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Square Root Problem</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Prime Number Problem</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Area-Perimeter Problem</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Euclid's Algorithm</td>
<td>29</td>
</tr>
<tr>
<td>4.5</td>
<td>Sort Problem</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>APPENDIX A</th>
<th>Matrix Operations</th>
<th>33</th>
</tr>
</thead>
</table>

| APPENDIX B | Suggested Problems               | 34   |
1.1 Symbols

The language BASIC offers very little change to the common notation used in elementary mathematics to express relationships between numbers. The first type of symbols we will mention deals with equality and inequality. The symbols, their meaning, and usage are summed up in Table 1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equals</td>
<td>A=B</td>
<td>A equals value of B</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>A&lt;B</td>
<td>A is less than B</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>A&gt;B</td>
<td>A is greater than B</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>not equal to</td>
<td>A&lt;&gt;B</td>
<td>A is not equal to B</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>A&lt;=B</td>
<td>A is less than or equal to B</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>A&gt;=B</td>
<td>A is greater than or equal to B</td>
</tr>
</tbody>
</table>

Table 1. Equality/Inequality Symbols

In addition to the equality and inequality symbols, we have arithmetic symbols. These symbols are used to denote arithmetic operations. Several symbols may be combined to form an expression in BASIC.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>A+B</td>
<td>add A and B</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>A-B</td>
<td>subtract B from A</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>A*B</td>
<td>multiply A by B</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>A/B</td>
<td>divide A by B</td>
</tr>
<tr>
<td>^</td>
<td>exponentiation</td>
<td>A^B</td>
<td>raise A to the power B</td>
</tr>
</tbody>
</table>

Table 2. Arithmetic Symbols
Questions from Section 1.1

True - False

1. One indicates inequality in BASIC by "≠".
2. Exponentiation utilizes the "/" character.
3. BASIC allows more than one arithmetic symbol in one expression.

Explanation

1. False. Inequality may be tested only by the "<>" symbols.
2. False. Exponentiation may only be indicated by a "|^|".
3. True. This form is allowable; for example, to add three variables we have: A+B+C

Note: Operations are performed with the following priorities

1. Evaluate expressions in parenthesis
2. Exponentiation
3. Multiplication and Division
4. Addition and Subtraction
1.2 Constants

Constants may contain from one to nine digits with a decimal point if needed. Only the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 may be used. Examples of valid and invalid constants are given in Table 3.

763 (VALID)
947.2 (VALID)
216347889 (VALID)
15x10 (INVALID)
6.3.2 (INVALID)
1134482006 (INVALID)

Table 3. Constants

Questions

Indicate which of the following are valid constants. If invalid give reason.

a. 2001
b. 9.80**2
c. 19.86727649
d. .89999

Explanation

a. VALID
b. INVALID since asterisks are not allowed.
c. INVALID because of too many characters.
d. VALID
1.3 Variables

In elementary BASIC as in many programming languages, variables are used to represent values we store in the computer for later calculations. If, for example, we wanted to compute the area of a circle, we would need to know the radius or diameter to which we would assign a name to hold the value. In light of the fact that this radius or diameter will vary for different circles, we have the term "variable name". To create a valid variable name, we simply use a letter of the alphabet or a letter immediately followed by a digit from 0 - 9. Thus in any program, there is a total of 286 possible variable names. There is another class of variables to which we may assign a subscript called "subscripted variables" (Section 2.16). Table 4 illustrates some valid and invalid variables.

Table 4. Variables

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>VALID</td>
</tr>
<tr>
<td>K7</td>
<td>VALID</td>
</tr>
<tr>
<td>XT</td>
<td>INVALID</td>
</tr>
<tr>
<td>7B</td>
<td>INVALID</td>
</tr>
<tr>
<td>X33</td>
<td>INVALID</td>
</tr>
</tbody>
</table>
Questions from Section 1.3

Indicate which of the following variable names are valid and invalid. If invalid, give reason.

a. D4  VALID
b. Z   VALID
 c. A9  VALID
d. JI   INVALID since I is not a digit.

Explanation

a. VALID
b. VALID
c. VALID
d. INVALID since I is not a digit.
2.1 Statements

Statements in BASIC are instructions which direct the computer to execute various computations. For our purpose, we shall study seventeen types of BASIC statements which will provide a great deal of flexibility in solving problems. There is one important fact that is common to all BASIC statements and that is each must have a statement number which must not exceed four digits. As we study each statement, you will find the statement number always appearing first. Most statements are referred to as "executable" statements; however, statements such as "END" and "DIM" are not executable.

Questions

1. Classify BASIC statements into two general categories.
2. What do statements consist of?

* * * * *
#----#----#

Explanations

1. Executable and non-executable
2. Statement number, symbols, numbers and variables.
2.2 READ

The READ statement is used for inputting values into the computer. The values that are to be read in are assigned to variable names, these values coming from DATA statements (Section 2.3). Following the statement number, we have the word READ and a list of variables or perhaps just one variable which seek values for later computations. The variables are separated by commas.

116 READ H

124 READ H, G, I, T6

2.3 DATA

The DATA statements form a block of information which is referenced by the READ statements. The data block is a list of values to be assigned to variables located in READ statements. This list of values is stored in the machine. The "data block" is created this way; whenever a DATA statement is encountered, the values on the DATA statement are added to the list of values from previous DATA statements. The values are also separated by commas.

170 DATA 23.6, 42, 16.1

"Data block" looks like 23.6
42
16.1

171 DATA 43.7, 5000

"Data block" looks like 23.6
42
16.1
43.7
5000

106 READ X2, J

108 DATA 7, 2
These two statements illustrate the assignment of the value 7 to the variable X2 and the value 2 to the variable J.

Examine the following:

```
100 READ A,B,C
150 READ X,Y,Z
180 READ J
200 DATA 20,30
206 DATA 3,6,9
208 DATA 11,12
```

Once the machine examines these instructions, we have the following:

- A=20
- B=30
- C=3
- X=6
- Y=9
- Z=11
- J=12

A better way to write this would be:

```
100 READ A,B,C,X,Y,Z,J
200 DATA 20,30,3,6,9,11,12
```

2.4 RESTORE

Whenever a READ statement is encountered, the value immediately following the last value read in is now read. To return to the beginning of the list of values assigned to this data list, the programmer simply inserts a RESTORE command.

```
300 RESTORE
```
For example, if the data block consisted of the following values

\[
\begin{align*}
4 \\
6 \\
3 \\
9
\end{align*}
\]

and if the first two values had already been read, then the next variable in a READ statement would be assigned the value "3". If, however, a RESTORE command were executed, the value assigned to the variable in the next READ statement would be "4" instead of "3".

2.5 PRINT

The PRINT statement is used to print out remarks and values assigned to variables. To print out the answer to a given problem whose solution is stored in \(X\), one may code the following:

\[
\begin{align*}
120 & \text{ PRINT "ANSWER IS"} \\
130 & \text{ PRINT } X
\end{align*}
\]

Notice that remarks are printed by simply enclosing in quotation marks the desired characters. This same example may be coded

\[
\begin{align*}
140 & \text{ PRINT "ANSWER IS", } X
\end{align*}
\]

The comma provides long spacing between the remark and the value of \(X\). Short spacing is accomplished by replacing the comma by a semicolon. The primary difference between the above examples is that the value of \(X\) will be printed below the remark in the first example and next to the remark in the second example. Short spacing simply means that the arguments
in the PRINT statement are printed closer together. Similarly, long spacing provides wider spacing on the printed line of the arguments in the PRINT statement.

Questions

1. Can we use a READ statement without having defined a data block?

2. What value will be printed out after the following instructions are executed?

```
161 READ A, B1, D, G9, K, Y2
162 DATA 6.35, 900, 42.0, 8.0
163 PRINT Y2
164 DATA 82.9, 0.1, 62, 7, 0
```

Explanations

1. No. The READ statement takes the values in the data block and assigns them to variables.

2. The value of only Y2 is printed. Since the data block consists of 6.35, 900, 42.0, 8.0, 82.9, 0.1, 62, 7 and 0, we observe that the sixth value is being assigned to Y2, i.e., 0.1.
2.6 LET

The LET statement is used to assign values to variables, or to compute values and then assign the result to variables.

LET variable = expression

Examples:

100 LET A=10 (assigns value 10 to variable A)
120 LET G=G+1 (increments value of G by one)
140 LET M=B+C (assigns M the sum of B and C)

Note that the second example does not appear to be mathematically correct. The equal sign should be interpreted as "is replaced by" and not "is equal to". The variable G is increased by 1 and this result replaces the previous value of G.

The BASIC language assumes that any variable not previously defined has a value of "zero". Thus the programmer may initialize a variable to contain a "zero" by

LET (variable) = 0

or completely omit this statement in which case the variable will be "zero".
**Question**

Determine what value is printed after execution of the following:

```plaintext
660   READ X,Y
662   DATA 3.0,4.0
664   LET Z=5.0
666   LET A=(X+Y+Z)/3.0
668   PRINT A
```

**Explanation**

Initially the variables X and Y are 3.0 and 4.0 respectively. The variable Z is assigned the value of 5.0 and the variable A is simply the average of X, Y, Z. Therefore, we print \((3+4+5)/3=4\).
Often it becomes necessary to divert from the instruction stream. This is a technique called "branching" which is performed by the GOTO (or GO TO) statement. If we wish to not execute the next sequential instruction, we insert a GOTO statement which transfers control to some other instruction. This instruction in question must have the corresponding statement number as the statement number in the GOTO statement; therefore, the GOTO statement appears as follows:

175 GOTO (statement no. of statement control is to be passed to)

If control is presently at instruction 200 and we wish to branch to the instruction with statement number 500, we have

200 GOTO 500

You may branch to an instruction already executed or ahead to one further along in the instruction stream.

2.8 IF

The IF statement is another type of control statement and is used for testing various relationships, allowing branching to occur providing certain conditions are satisfied. The format of the IF statement is as follows:

IF (statement) THEN (statement number)

The statement portion of the statement may take on many forms where we have variables from the program and symbols mentioned in Section 1.1.
The statement number refers to the statement to which control is passed if the expression is affirmative.

Examples:

a. 1 IF A=2 THEN 200
b. 1 IF A<4 THEN 205
c. 1 IF 0<=7 THEN 106
d. 1 IF D>8 THEN 110
e. 1 IF D>=1 THEN 180
f. 1 IF E<>10 THEN 250

In example a), if the variable A had been assigned the value 2, we have an affirmative result and control is passed to the instruction at statement number 200. If A is anything else, control passes to the next executable statement. In example f), if E=10 then control passes to the next executable statement whereas for all other values of E, the branch to statement 250 will take place.

Question

Write a BASIC statement to do the following:

Branch to statement 700 if the sum of variables X1 and X2 is negative.

Explanation

Using an IF statement, we sum the variables and test for negativity.

360 IF X1+X2<0 THEN 700
2.9 FOR

To execute a series of instructions a number of times, a loop is formed to avoid repetitious coding. The loop is initiated by the FOR statement and terminated by the NEXT statement. The format:

```
FOR (variable)=(starting) to (final) STEP (increment)
```

The variable must be a valid BASIC variable which has not been used or one that will not be needed for later calculations. This variable will be assigned the "starting value" as the statements are executed until the NEXT statement is encountered. When this occurs, the loop is completed and the variable is incremented (or decremented) by the "increment" (i.e., may be < 0). If this new value does not exceed the "final value", then the entire loop is executed again with this new value for the variable; otherwise, control passes outside the loop to the next executable statement after the NEXT statement.
This statement acts to signal the end of a loop. Only the variable used in the FOR statement appears in this statement.

Example:

```
FOR T=1 to 5 STEP 1
    ...
    ...
    ...
    NEXT T
```

The statements from the FOR to the NEXT statement are executed five times. The first time T will be assigned the value one and the fifth time, T is five. If the "STEP" were two, then the loop would be executed three times with T having values 1, 3 and 5. You need not specify a STEP of 1, i.e., "FOR T=1 to 5" we assume STEP 1.

**Question**

1. Determine the purpose of the following coding:

   163  FOR S=1 to 10
   164  LET E=S+E
   165  NEXT S

2. Change statement 163 so as not to change the problem (2 ways).
1. The variable S is assigned values of 1 - 10 and each time the value is added to the sum of previous values. Thus we have the sum of the numbers from 1 to 10:

\[ \sum_{S=1}^{10} S \]

2. Statement 163 may be written by including the assumed STEP size and also by changing initial and final values with a negative step.

163 FOR S=1 TO 10 STEP 1

or

163 FOR S=10 TO 1 STEP -1

2.11 END

The first phase that a program encounters is the compilation phase where all the instructions are translated into machine (object) coding in order to have a "machine program" for the second phase, execution. The END statement is used to signal end of compilation in the initial phase. Once compilation ceases and there have been no errors found, the program is executed; otherwise, a list of errors is printed and execution does not take place. There may be only one END statement in a BASIC program and it must be the last statement of the program. This statement is categorized as non-executable.
2.12 Input

This statement acts as a conversational read statement which simply means that when executed, the console will pause and wait for the user to input data. For example, INPUT Y when executed will cause the program to stop and seek data which will be supplied by the user through the keyboard of the remote console.

To read six variables in this manner, we may have the following:

97 INPUT N,U,M,B,E,R

Each value typed in on the console must be separated from other values by a comma. The first value will be assigned to N, the second to U, etc. It is wise to print a literal message just before this statement indicating exactly what input is desired.

Questions

1. Explain the differences between the READ and INPUT statements.

2. Write a short program to continually read in from the console two numbers and print out their sum only if it is positive.

* * *
Explanations

1. The INPUT statement allows one to assign values to variables during execution whereas the READ statement must have the values for the variables in the data block before execution. The INPUT statement doesn't use the data block at any time.

2. 
100 INPUT G,H
200 LET F=G+H
300 IF F<=0 THEN 100
400 PRINT F
500 GOTO 100
600 END
One may define a function which can be used repeatedly in the program by using a function name consisting of FN and some other letter and the following:

(statement no.) DEF (function name) = (function)

We may define the function $x^2 + 2x + 1$ the following way:

160 DEF FNT(X) = X^2 + 2*X + 1

Using this function and the following coding, what would be the value of B?

161 LET A = 2
162 LET B = FNT(A)

The correct answer is 9 since the value of A (2) will be used for X in the FNT function.

i.e., $2^2 + 2*2 + 1 = 4 + 4 + 1 = 9$

**Question**

Write a function for the following:

$$
\frac{y^3 + 25y^2 - (y+1)}{7(y^2+6)^2}
$$

**Explanation**

An acceptable result would be

100 DEF FNZ(Y) = (Y^3 + 25*Y^2 - (Y+1)) / (7*(Y^2+6)^2)
The **STOP** statement signals the termination of execution. Unlike the **END** statement, there may be more than one **STOP** statement in the program. The **STOP** statement is categorized as a control statement and may appear anywhere in the program.

**Question**

Find the errors in the following **BASIC** program.

```
100 DATA 0.0,1.6,7,11
200 READ X,P,T,A,B,C,Z
300 PRINT Z
400 DATA 6.00,9,800,61,98.7,0
500 DEF FNC(A)=A|2
600 LET W=2+3|3*2
700 PRINT W
800 STOP
900 END
```

**Explanation**

There are no errors in this program. The data block consists of 10 values and since we wish to print out Z, the seventh value, we have an answer of 800. The variable is computed by considering the priorities mentioned in Chapter 1. First we exponentiate, then multiply, and finally add. Thus the value for W is 56.
To insert comments into the program, we use the REM statement. Once the REM is encountered, the remaining information on that line is treated as comment and is not compiled into the program. This is used to aid you in better describing your program. Example:

```
999 REM THIS IS A REMARK
```

Variables may have subscripts attached to them by having the variable followed by a digit enclosed in parentheses. These variables are called subscripted variables. Storage space must be reserved for each subscripted variable. This is done automatically if the largest subscript does not exceed 10. Otherwise the DIM statement must be used to reserve this storage. If the variable S were to be assigned a maximum of 20 values, namely, S(1), S(2), ..., S(20), then we would need the following: DIM S(20) at the beginning of the program. If the variable V were to be treated in the same manner, we would have

```
DIM S(20), V(20)
```

Therefore, if a variable is to be treated as a subscripted variable and we do not wish to assign more than 10 distinct values, we may omit the DIM statement.
2.17 GOSUB

In programming languages, a term which commonly arises is "subprogram", of which there are various types. We will mention only the subroutine which is a block of coding to perform a particular function. This subprogram is branched to by means of "GOSUB" and the statement number of the first instruction of the subroutine. When completed, control returns to the statement immediately following the GOSUB statement.

2.18 RETURN

The last statement of every subroutine is the RETURN statement which simply transfers control back to the statement immediately following the GOSUB statement.
3.1 Built-In Functions

Mentioned in Section 2.13 is a method of creating functions. There are functions which are already available to BASIC programmers. These functions are called built-in functions. They are:

- **SIN(X)**: sine of X
- **COS(X)**: cosine of X
- **TAN(X)**: tangent of X
- **ATAN(X)**: arctangent of X
- **ABS(X)**: absolute value of X
- **INT(X)**: largest integer not greater than X
- **LOG(X)**: natural log of X
- **EXP(X)**: $e^X$
- **SQR(X)**: square root of X
- **SGN(X)**: yields -1, 0, or +1 for X negative, zero, or positive, respectively
- **RND**: generates random number between 0 and 1

To determine the square root of X and assign the result to Y, we have

```
100 LET Y=SQR(X)
```

To evaluate $e^{X+Y}$, and assign the result to A, we have

```
100 LET A=EXP(X+Y)
```

All the other functions work this way.
3.2 Numeric Applications

In this section we shall illustrate how useful the built-in functions are by programming some simple examples.

a. Find the square root of the numbers from 1 to 20

```
510 FOR I=1 TO 20
520 LET J=SQR(I) } PRINT SQR(I)
530 PRINT J
540 NEXT I
550 STOP
560 END
```

b. Find the SIN, COS, and TAN of 20, 25, 30, 35, 40 and 45 degrees.

Assume \( \frac{180}{\pi} = 57.3 \)

```
210 FOR A=20 TO 45 STEP 5
220 PRINT SIN(A/57.3), COS(A/57.3), TAN(A/57.3)
230 NEXT A
240 STOP
250 END
```

c. Show that the random number generator generates a positive value.

```
250 LET Z=RND
251 LET S=SGN(Z)
252 PRINT S
253 STOP
254 END
```
Problem 1

SQUARE ROOT PROBLEM

A perfect square may be defined to be a number whose square root is an integer. The numbers 1 and 4 have square roots 1 and 2 respectively whereas 2 and 3 have square roots 1.414 and 1.732 respectively. The numbers 1 and 4 are thus perfect squares. Write a BASIC program to find all the perfect squares in any given interval. The input should be via the remote console and should accept the beginning and ending value of the interval.

Solution 1

The following program determines all the perfect squares in a given interval.

```
100 INPUT S,L
200 PRINT "THE PERFECT SQUARES BETWEEN";S;"AND";L
300 FOR X=S TO L
400 IF SQRT(X)<INT(SQRT(X)) THEN 600
500 PRINT X
600 NEXT X
700 STOP
800 END
```

Programming Note: Notice in statement 400 the nesting of the built-in functions INT and SQRT. A loop to consider all numbers between the end points is constructed from statement 300 to statement 600.
Problem 2

PRIME NUMBER PROBLEM

A prime number is one which is divisible only by 1 and itself. Examples of prime numbers are 2, 3, 5, 7, 11, 13, ... Write a BASIC program to read via terminal a number and then determine whether or not it is a prime number. Print out the number and some remark as to whether or not it is prime.

Solution 2

The following program determines whether a given number is prime or not.

```
100 INPUT X
200 FOR T=2 TO X-1
300 IF X/T=INT(X/T) THEN 700
400 NEXT T
500 PRINT X;"IS A PRIME NUMBER"
600 GOTO 100
700 PRINT X;"IS NOT A PRIME NUMBER"
800 GOTO 100
900 END
```

Programming Note: Note that the limiting value in statement 200 is an expression and not a variable.
Problem 3

AREA-PERIMETER PROBLEM

For a rectangle of length \( l \) and width \( w \), area is defined to be \( l \times w \) and perimeter \( 2l + 2w \). Write a program in BASIC that will accept from the terminal a value for the length and a value for the width. Print out the length, width, area, and perimeter of the rectangle with appropriate labels. Write so as to handle an infinite number of rectangles.

Solution 3

The following is a solution to finding the area and perimeter of a rectangle:

```
100 INPUT L,W
200 PRINT "LEN WIDTH AREA PERIMETER"
300 PRINT L;W;L*W;2*L+2*W
400 GOTO 100
500 END
```

Programming Note: Unlike some programming languages, BASIC allows you to print out expressions as well as variables. FORTRAN, for example, has the expression assigned to a variable name which in turn is printed out. Note that the first PRINT statement acts as a heading for the four computed values.
Problem 4
EUCLID'S ALGORITHM

Euclid's algorithm is a technique used to determine the greatest common divisor of two integer numbers. The procedure is quite simple. First divide the larger by the smaller to obtain a quotient and remainder. If the remainder is not equal to zero, replace the larger by the smaller and the smaller by the remainder (in that order) and repeat the division. When the remainder is zero, the greatest common divisor (gcd) is the smaller at that time, not necessarily the smaller number at the beginning of the problem. The following flowchart will aid you in solving the problem. It is assumed the numbers are in descending order; otherwise a test would be necessary and this is quite trivial. Test your program with numbers 21 and 20.
Solution 4

The following is a solution to find the greatest common divisor of two numbers using Euclid's algorithm. Involved is a loop to handle more than one set of numbers.

```
100 INPUT L,S
110 PRINT "THE GCD OF";L;" AND ";S;" IS"
120 LET R=L-INT(L/S)*S
130 IF R=0 THEN 170
140 LET L=S
150 LET S=R
160 GOTO 120
170 PRINT S
180 GOTO 100
190 END
```

**Programming Note:** Notice that this program doesn't contain a STOP statement. Involved is a branch to statement 100 for a new set of numbers after one set is computed. To stop this program, you must cancel it. To consider just one set of numbers, replace statement number 180 by a STOP.
Problem 5

SORT PROBLEM

Some problems demand the use of a routine to sort numbers. For example, a teacher may wish to have the scores from an exam listed in descending order. Write a BASIC program that will input five numbers in any order and print them out in descending order. (Be sure to print out the array before the sorting takes place.)

Solution 5

The following program will sort five numbers into descending order.

100 INPUT A(1),A(2),A(3),A(4),A(5)
200 PRINT "ORIGINAL ARRAY"
300 PRINT A(1);A(2);A(3);A(4);A(5)
400 FOR I=1 TO 4
500 FOR J=I+1 TO 5
600 IF A(I)>A(J) THEN 1000
700 LET B=A(I)
800 LET A(I)=A(J)
900 LET A(J)=B
1000 NEXT J
1100 NEXT I
1200 PRINT "SORTED ARRAY"
1300 PRINT A(1);A(2);A(3);A(4);A(5)
1400 STOP
1500 END
**Programming Note:** The array is not dimensioned since we are assigning less than ten values to it. The two loops, one within the other is called "nested" loops. The inner most loop is executed to completion while the outer loop stays fixed. Then the outer loop is increased and the inner one repeated. Completion is reached when the outer loop has reached its limit and the inner has been executed for this value of the outer limit.
# Appendix A

## Matrix Operations

<table>
<thead>
<tr>
<th>Statements</th>
<th>Purpose</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAT READ A</td>
<td>read a matrix</td>
<td>Reads from data statement values and stores them in matrix A row-wise.</td>
</tr>
<tr>
<td>MAT PRINT A</td>
<td>print a matrix</td>
<td>Print out all values that are assigned to matrix A.</td>
</tr>
<tr>
<td>MAT A=B+C</td>
<td>matrix addition</td>
<td>Add corresponding elements of matrix C and matrix B, storing results in matrix A.</td>
</tr>
<tr>
<td>MAT A=B-C</td>
<td>matrix subtraction</td>
<td>Subtract corresponding elements of matrix C from matrix B, storing results in matrix A.</td>
</tr>
<tr>
<td>MAT A=B*C</td>
<td>matrix multiplication</td>
<td>Multiply corresponding elements of matrix C by matrix B, storing results in matrix A.</td>
</tr>
<tr>
<td>MAT A=(B)*C</td>
<td>multiply matrix by scalar</td>
<td>Multiply matrix C by the number (or expression) B and store results in matrix A.</td>
</tr>
<tr>
<td>MAT A=INV(B)</td>
<td>inverse of matrix</td>
<td>Take the inverse of matrix B and store in matrix A.</td>
</tr>
<tr>
<td>MAT A=TRN(B)</td>
<td>transpose of matrix</td>
<td>Store the transpose of matrix B in matrix A.</td>
</tr>
<tr>
<td>MAT A=CON</td>
<td>matrix of unity</td>
<td>Set all elements of matrix A to &quot;1&quot;.</td>
</tr>
<tr>
<td>MAT A=ZER</td>
<td>zero matrix</td>
<td>Set all elements of matrix A to &quot;0&quot;.</td>
</tr>
<tr>
<td>MAT A=IDN</td>
<td>identity matrix</td>
<td>Set matrix A to identity matrix (i.e., main diagonal elements are &quot;1&quot;; off-diagonal elements are &quot;0&quot;)</td>
</tr>
</tbody>
</table>
Appendix B

Suggested Problems

1. Write a BASIC program to print out a table of sines, cosines, and tangents for angles of 0° to 90°.

2. Write a program to find the factorial of an integer N.

3. Write a program to compute the interest for rates from 1% to 25% using a constant principal and time.

\[ \text{INTEREST} = \text{PRINCIPAL} \times \text{RATE} \times \text{TIME} \]

4. Write a program to compute the mean and standard deviation of N numbers.

\[ \sigma = \sqrt{\frac{\sum x^2}{N} - \bar{x}^2} \]

5. Write a program to print a table for converting 1-12 inches to centimeters (1 inch = 2.54 centimeters).

6. Write a program to evaluate \( \text{SUM} \) where

\[ \text{SUM} = \sum_{J=1}^{N} 3\pi(2)^J \cos(42°) \] \( \text{where} \ \pi = 3.14159 \]

7. Write a program to find the square root of any number, except zero.

If the number is negative, print out its square root as an imaginary number, e.g. \( \sqrt{-4} = 2i \).

8. Write a program to determine the following ratio for values of \( X \) from 100 to 1000 in increments of 100

\[ \frac{A \cdot \log(X)}{X} \]

where \( A \) is the number of prime numbers less than \( X \).
APPENDIX C

PROGRAMMED EXAMINATION LISTING WITH QUESTIONS AND SAMPLE EXAMS
m.0073 action in progress.
1.0001 /fte
1.0002 dimension key(10),msl(50,20),lans(50),ques(50,20),answer(50,20)
1.0003 integer bank,pick(10),select(50),acct(50,4)
1.0004 c
1.0005 c instructions and description
1.0006 c
1.0007 write(6,1000)
1.0008 c
1.0009 c initial values for counters
1.0010 c
1.0011 k1=0
1.0012 k2=0
1.0013 k3=0
1.0014 k4=0
1.0015 c
1.0016 c input questions, solution choices, description of correct answers
1.0017 c
1.0018 read(5,1800)bank,noask
1.0019 read(5,1900)(lans(i),i=1,bank)
1.0020 c
1.0021 read(5,1200)(ques(i,j),j=1,20)
1.0022 read(5,1200)(answer(i,j),j=1,20)
1.0023 1 read(5,1200)(msl(i,j),j=1,20)
1.0024 c
1.0025 7 read(5,1600)(acct(i,j),j=1,4)
1.0026 c
1.0027 c generate a key for this exam as well as ten random questions
1.0028 c
1.0029 call time(lu)
1.0030 lu=lu*2+1
1.0031 v=2.0**32
1.0032 lx=5**15
1.0033 do 500 i=1,noask
1.0034 lu=lu*lx
1.0035 500 key(i)=fix((lu/v+0.5)*lx)+1
1.0036 do 600 i=1,noask
1.0037 600 key(i)=fix((lu/v+0.5)*bank)+1
1.0038 if(select(1posse)=601,602
1.0039 601 select(1posse)=1
1.0040 600 pick(i)=1posse
1.0041 c
1.0042 c prepare each question into format specified by key
1.0043 c ask questions and determine if question is to be asked again
1.0044 c
1.0045 c
1.0046 do 100 nq=1,noask
1.0047 if pick(nq)
1.0048 if(msl(1p,13)+707340859)=2,3,2
1.0049 3 if(lans(1p)-4)=4,5,4
1.0050 5 key(nq)=4
I.

\begin{verbatim}
 1.0051  go to 11
 1.0052  4 if(key(nq)=4)2,6,2
 1.0053  6 key(nq)=lans(lp)
 1.0054  go to 11
 1.0055  2 do 10 i=1,k
 1.0056  4 k=lans(lp)*4-(11-1)
 1.0057  6 if(key(nq)=4*(i-1)
 1.0058  4 m=msl(lp,k)
 1.0059  4 msl(lp,k)=m
 1.0060  10 msl(lp,i)=m
 1.0061  11 write(6,1500)npos,ques(lp,i),i=1,20)
 1.0062  12 write(6,1008)(msl(lp,j),j=1,16)
 1.0063  100 continue
 1.0065  c all questions have been asked
 1.0066  c output results
 1.0067  c
 1.0068  c
 1.0069  write(6,1700)
 1.0070  1700 format(/'you have just completed the first exam in basic',/)
 1.0071  write(6,1701)
 1.0072  1701 format(/5x,'summary',/)
 1.0073  write(6,1702)(k1,k2,k3
 1.0074  1702 format('/correct choice first try',13,'correct choice second try',13,/'answer was supplied',13,
 1.0075  1703 format(5x,'proficiency',5x,'***********',/)
 1.0076  c compute student proficiency
 1.0077  c
 1.0078  p=(k1+k2/2)*(100/noask)
 1.0079  If(p<90)60,331,331
 1.0080  60 if(p<70)70,333,333
 1.0081  70 if(p<60)80,335,335
 1.0082  c ratings
 1.0083  c
 1.0084  1704 format(10x,'poor')
 1.0085  go to 20
 1.0086  1705 format(10x,'fair')
 1.0087  go to 20
 1.0088  1706 format(10x,'good')
 1.0089  go to 20
 1.0090  1707 format(10x,'excellent')
 1.0091  20 write(6,1708)
 1.0092  1708 format(/)
 1.0093  write(10,1600)((acct(l,j),j=1,4),i=1,rank)
 1.0094  stop
\end{verbatim}
I.0104       1000  format('1','this program is designed to test your knowledge of bas
I.0106       1110c',/,'once you have completed the exam, a summary will appear',/,'2
I.0108       1200c       'showing how many questions were answered correctly on the first
I.0109       1300c       'attempt and how many on the second attempt. If the',/,'correct
I.0110       1400c       'choice has not been made at this point, a full descripti
I.0111       1500c       'on for the correct answer. Answer each',/,'question as quick
I.0112       1600c       'ly as possible',/)
I.0113       1700c       1008  format(5x,'1',4a4,'2',4a4,'3',4a4,'4'),4a4,/) 1.0114       1800c       1200  format(20a4)
I.0115       1900c       1500  format(1x,12,1x,20a4)
I.0116       2000c       1600  format(415)
I.0117       2100c       1800  format(212)
I.0118       2200c       1900  format(4012)
I.0119          end
I.0120       2300c       /ftc
I.0121       2400c       subroutine subhy(key,nq,k1,k2,k3,k4,lly,answer,lp,acct)
I.0122          dimension key(10),answer(50,20)
I.0123          integer acct(50,4)
I.0124          99   write(6,100)
I.0125     100   format('answer')
I.0126          end
I.0127          c
I.0128          c          Input and correct student answer
I.0129          c          kk=k4+1
I.0130     1000   read(9,200)input
I.0131     2000   format(11)
I.0132          go to (205,205,205,205),input
I.0133          write(6,105)
I.0134     105   format(/,'Invalid response',/,'you must indicate your answer by th
I.0135          le number 1, 2, 3, or 4',/)
I.0136          go to 99
I.0137     205   if(k4-1)298,298,301
I.0138     298   lrec=input
I.0139          go to 303
I.0140     301   if(lrec-input)303,302,303
I.0141     302   write(6,600)
I.0142     600   format(/,'this answer has already been given. try again.',/)
I.0143          go to 1000
I.0144     303   if(input-key(nq))1,928,1
I.0145          1   write(6,300)
I.0146     300   format(/,'wrong answer',/)
I.0147     302   if(k4-2)2,935,2
I.0148          2   kk=k4+1
I.0149          write(6,400)
I.0150     400   format(/,'try this one again',/)
I.0151          go to 1000
I.0152     929   if(k4-3)3,930,3
I.0153          3   k2=k2+1
I.0154     acct(lp,3)=acct(lp,3)+1
numbers may contain up to ---- digits.
numbers in basic vary from 1 to 9 digits.
six two nine ten
variables always start with a letter and are sometimes followed by a ---.
variables may be followed by a single digit.
a letter a digit $ none of these
---- has the highest priority of all arithmetic operations.
exponentiation has the highest priority of all the operations.
addition multiplication exponentiation division
statement numbers have a maximum of ---- digits.
statement numbers can not exceed 4 digits.
four three two three two four ten
values assigned to variables in a read statement appear in a ---- statement.
information for a read statement is gotten from the data statement.
let print next data
the ---- statement acts as a conversational read statement.
the input statement allows you to read conversationally.
read Input find none of these
each 'for' statement has an associated ---- statement.
a loop must be completed by a 'next' statement.
next let read If
decision making utilizes the ---- statement.
a decision is made via the 'if' statement.
for let gosub If
to increase the variable x by 1, which is correct
the correct form is 'let variable=expression'.
let x=x+1 let x+1=x x=x+1
a dim statement is needed for arrays with more than ---- values.
for arrays of size 10 or less, no 'dim' is needed.
9 of the following is invalid in basic
the variable 'area' is invalid in basic because of too many characters.
let area=1 let c=19.3 rem If a=0 then 414
the last statement of every basic subroutine must be ----.
all subroutines are ended with a 'return' statement.
going back end gosub return
1.0209 an option on the 'for' statement is the ---- parameter.
1.0209 In a loop, one may specify the interval size by the step parameter.
1.0211 a ---- statement is needed in order to use the 'data' list more than once.
1.0212 the 'restore' statement returns the user to the beginning of the data block.
1.0213 restore reread data input
1.0214 a ---- causes short spacing to the next element.
1.0216 a comma is for long spacing and a semi-colon for short spacing.
1.0217 literals appear in 'print' statements enclosed in ----.
1.0219 commas appear in quotation marks.
1.0220 for ---- expressions, the value of the innermost expression is calculated first
1.0221 expressions within expressions are referred to as being 'nested'.
1.0222 abbreviated simple nested basic
1.0223 ---- are used to separate variables within the formula.
1.0224 variables may be separated by arithmetic operators.
1.0225 parenthesis dashes blanks none of these
1.0226 the subscript of an array may be a variable or a ----.
1.0227 subscripts may be variables, constants, or expressions.
1.0228 * / none of these
1.0229 a basic statement must be contained on ----.
1.0230 basic statements may not exceed one line.
1.0231 one page one line three lines 15 columns
1.0232 the trigonometric functions require one argument in ----.
1.0233 all trigonometric functions require an argument in radians.
1.0234 degrees degs rads radians none of these
1.0235 the sgn function produces one of ---- different values.
1.0236 the 'sgn' function produces +1, 0, or -1.
1.0237 two three four none of these
1.0238 a ---- is a type of subprogram.
1.0239 a subroutine is classified as a type of subprogram.
1.0240 matrix gosub subroutine none of these
1.0241 a rem statement is used to insert ---- into the program.
1.0242 rem statements allow the programmer to comment his program.
1.0243 comments errors print outs clarity
1.0244 one may define a maximum of ---- functions.
1.0245 since 'function' statements must be fn and a letter, max. no. is 26.
1.0246 five two one hundred none of these
1.0247 the stop statement signals end of ----.
1.0248 stop is used to end execution of the program.
1.0249 compilation execution printing reading data
1.0250 the maximum number of equality/inequality symbols allowed per statement is
1.0251 one three two none of these
1.0252 the statement number that must follow statement 99 is ----.
1.0254 presently, statement numbers need not be in sequence.
1.0255 one hundred ninety eight one none of these
1.0256 the sign of the step increment of a 'for' statement must be ----.
1.0257 the 'step' parameter may be either positive or negative.
1.0258 positive negative pos. or neg. none of these
1.0259

the
variable
for
the
next
statement
is
set
with
a
for
statement
and
ended
with
a
next
statement.

1.0261

there
are
a
total
of
six
arithmetic
symbols
in
basic.

1.0262

five
inequality
symbols
must
be
used.

1.0263

three
variables
may
have
either
one
or
two
characters
but
never
more.

1.0264

one
may
not
take
place
in
a
let
statement.

1.0265

none
of
these
arithmetic
operations
may
or
may
not
execute.

1.0266

none
of
the
forms
for
matrix
operations.

1.0267

none
of
the
forms
for
matrix
operations.

1.0268

none
of
the
forms
for
matrix
operations.

1.0269

none
of
the
forms
for
matrix
operations.

1.0270

none
of
the
forms
for
matrix
operations.

1.0271

none
of
the
forms
for
matrix
operations.

1.0272

none
of
the
forms
for
matrix
operations.

1.0273

none
of
the
forms
for
matrix
operations.

1.0274

none
of
the
forms
for
matrix
operations.

1.0275

none
of
the
forms
for
matrix
operations.

1.0276

none
of
the
forms
for
matrix
operations.

1.0277

none
of
the
forms
for
matrix
operations.

1.0278

none
of
the
forms
for
matrix
operations.

1.0279

none
of
the
forms
for
matrix
operations.

1.0280

none
of
the
forms
for
matrix
operations.

1.0281

none
of
the
forms
for
matrix
operations.

1.0282

none
of
the
forms
for
matrix
operations.

1.0283

none
of
the
forms
for
matrix
operations.

1.0284

none
of
the
forms
for
matrix
operations.

1.0285

none
of
the
forms
for
matrix
operations.

1.0286

none
of
the
forms
for
matrix
operations.

1.0287

none
of
the
forms
for
matrix
operations.

1.0288

none
of
the
forms
for
matrix
operations.

1.0289

none
of
the
forms
for
matrix
operations.

1.0290

none
of
the
forms
for
matrix
operations.

1.0291

none
of
the
forms
for
matrix
operations.

1.0292

none
of
the
forms
for
matrix
operations.

1.0293

none
of
the
forms
for
matrix
operations.

1.0294

none
of
the
forms
for
matrix
operations.

1.0295

none
of
the
forms
for
matrix
operations.

1.0296

none
of
the
forms
for
matrix
operations.

1.0297

none
of
the
forms
for
matrix
operations.

1.0298

none
of
the
forms
for
matrix
operations.

1.0299

none
of
the
forms
for
matrix
operations.

1.0300

none
of
the
forms
for
matrix
operations.

1.0301

none
of
the
forms
for
matrix
operations.

1.0302

none
of
the
forms
for
matrix
operations.

1.0303

none
of
the
forms
for
matrix
operations.

1.0304

none
of
the
forms
for
matrix
operations.

1.0305

none
of
the
forms
for
matrix
operations.

1.0306

none
of
the
forms
for
matrix
operations.

1.0307

none
of
the
forms
for
matrix
operations.

1.0308

none
of
the
forms
for
matrix
operations.

1.0309

none
of
the
forms
for
matrix
operations.

1.0310

none
of
the
forms
for
matrix
operations.

1.0311

none
of
the
forms
for
matrix
operations.

1.0312

none
of
the
forms
for
matrix
operations.

1.0313

none
of
the
forms
for
matrix
operations.

1.0314

none
of
the
forms
for
matrix
operations.

1.0315

none
of
the
forms
for
matrix
operations.

1.0316

none
of
the
forms
for
matrix
operations.

1.0317

none
of
the
forms
for
matrix
operations.

1.0318

none
of
the
forms
for
matrix
operations.

1.0319

none
of
the
forms
for
matrix
operations.

1.0320

none
of
the
forms
for
matrix
operations.

1.0321

none
of
the
forms
for
matrix
operations.

1.0322

none
of
the
forms
for
matrix
operations.

1.0323

none
of
the
forms
for
matrix
operations.

1.0324

none
of
the
forms
for
matrix
operations.

1.0325

none
of
the
forms
for
matrix
operations.
this program is designed to test your knowledge of basic
once you have completed the exam, a summary will appear
showing how many questions were answered correctly on the first
attempt and how many on the second attempt. If the
correct choice has not been made at this point, a full des-
cription is given for the correct answer. answer each
question as quickly as possible

1 to increase the variable x by 1, which is correct
   1)x=x+1          2)let x+1=x    3)let x=x+1          4)1+x=x
answer
2

2 the subscript of an array may be a variable or a ---- .
   1)x             2)/            3)*             4)none of these
answer
1
wrong answer
try this one again
1
this answer has already been given. try again.
2
wrong answer

correct answer is no. 4
subscripts may he variables, constants, or expressions.
3. A valid Basic variable has a maximum of ---- characters.
   1) four  2) two  3) three  4) one
   answer
   4
   wrong answer
   try this one again

4. To test inequality, we must use ---- symbol(s).
   1) two  2) one  3) three  4) none of these
   answer
   1

5. The variable for the 'next' statement corresponds to the variable in the ----.
   1) input statement  2) read statement  3) for statement  4) data statement
   answer
   3

6. The stop statement signals end of ----.
   1) compilation  2) execution  3) printing  4) reading data
   answer
   2

7. Numbers may contain up to ---- digits.
   1) six  2) nine  3) two  4) ten
   answer
   2

8. There are a total of ---- arithmetic symbols in Basic.
   1) three  2) four  3) six  4) five
   answer
   4
9 a ---- statement is needed in order to use the 'data' list more than once.
   1)input       2)reread       3)data       4)restore

answer

4

10 a let statement ---- contains an arithmetic symbol.
   1)occasionally  2)always  3)never  4)none of these

answer

1

you have just completed the first exam in basic

summary

correct choice first try 8

correct choice second try 1

answer was supplied 1

proficiency

********

good

stop   00000
m.0072 begin activity.
/save keep(1602),sv
m.0073 action in progress.
file saved under specified name(lock)
m.0072 begin activity.
this program is designed to test your knowledge of basic
once you have completed the exam, a summary will appear
showing how many questions were answered correctly on the first
attempt and how many on the second attempt. If the
correct choice has not been made at this point, a full des-
cription is given for the correct answer. answer each
question as quickly as possible

1 the subscript of an array may be a variable or a ---- .
   1)+ 2)/ 3)* 4)none of these
answer

2 the stop statement signals end of ---- .
   1)compilation 2)printing 3)execution 4)reading data
answer

3 one may define a maximum of ---- functions.
   1)five 2)two 3)one hundred 4)none of these
answer

4 to increase the variable x by 1, which is correct
   1)1+x=x 2)let x+1=x 3)x=x+1 4)let x=x+1
answer
5 The statement number that must follow statement 99 is ---- .
   1) one hundred    2) ninety eight    3) one    4) none of these

answer 4

6 The sign of the step increment of a 'for' statement must be ---- .
   1) positive   2) negative   3) pos. or neg.   4) none of these

answer 3

7 Decision making utilizes the ---- statement.
   1) for   2) let   3) if   4) gosub

answer 3

8 ---- are used to separate variables within the formula.
   1) parentheses   2) dashes   3) blanks   4) none of these

answer 1

9 A basic statement must be contained on ---- .
   1) one page   2) 15 columns   3) three lines   4) one line

answer 4

10 The 'goto' statement is used for what is termed ---- .
   1) moving along   2) branching   3) going ahead   4) looping

answer 2
you have just completed the first exam in basic

summary

Correct choice first try 10
Correct choice second try 0
Answer was supplied 0

Proficiency
***********
excellent

Stop 00000
m.0072 begin activity.
/save keep(1602),sv
m.0073 action in progress.
File saved under specified name(lock)
m.0072 begin activity.
APPENDIX D

GENERATING PROGRAM FOR UNSOLVED PROBLEMS
WITH ORIGINAL PROBLEMS AND
SAMPLE EXERCISES
program to randomly prepare programs for exercises.

key word table

<table>
<thead>
<tr>
<th>Class</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>let goto</td>
</tr>
<tr>
<td>2</td>
<td>for next</td>
</tr>
<tr>
<td>3</td>
<td>data input</td>
</tr>
</tbody>
</table>

Variable name description

- kfile: number of files
- max: maximum number of errors per file
- junk: maximum number of phonny statements to be inserted
- class1: number of omissions for class 1 statements
- class2: number of omissions for class 2 statements
- class3: number of omissions for class 3 statements
- type1: number of blank line errors
- type2: number of statements with just key word
- type3: number of statements with key word missing

Dimension list(136, 50), kount(10), lIm(10)

Integer statyp(3, 4), class1, class2, class3, flag(10)

Integer type1, type2, type3, phoney(10, 44)

define key words

define key words

statyp(1, 1) = -742005952
statyp(1, 2) = -942212182
statyp(1, 3) = -909754304
statyp(1, 4) = -673502875
statyp(2, 1) = -958989208
statyp(2, 2) = -708450333
statyp(2, 3) = -641351228
statyp(2, 4) = -993405868
statyp(3, 1) = -993926207
statyp(3, 2) = -908732444

set file lengths

lim(1) = 22
lim(2) = 61
lim(3) = 70
lim(4) = 90
lim(5) = 115
lim(6) = 136
kfile = 6

Input parameters
1.0053 c
1.0054 write(6,2000)
1.0055 read(9,3000)max
1.0056 write(6,4000)
1.0057 read(9,3000)junk
1.0058 write(6,5000)
1.0059 read(9,7000)class1, class2, class3
1.0060 write(6,8000)
1.0061 read(9,7000)type1, type2, type3
1.0062 c
1.0063 c
1.0064 c
1.0065 1000 format(a3,a4,48a1)
1.0066 2000 format('Input max no. of errors per file.')
1.0067 3000 format(12)
1.0068 4000 format('Input max no. of junk lines to be inserted.')
1.0069 5000 format('Input no. of each class type.')
1.0070 6000 format(a4,48a1)
1.0071 7000 format(3,12)
1.0072 8000 format('Input no. of each type of change.')
1.0073 c
1.0074 c
1.0075 c
1.0076 read(5,1000)((((list(l,j),j=1,50),l=1,136)
1.0077 read(5,6000)((((phoney(l,j),j=1,49),l=1,10)
1.0078 c
1.0079 c
1.0080 c
1.0081 call time(lu)
1.0082 lu=lu*2+1
1.0083 lx=5**15
1.0084 v=2.0**32
1.0085 c
1.0086 c
1.0087 c
1.0088 if(class3)200,200,100
1.0089 100 do 1 ll=1,class3
1.0090 lu=lu+lx
1.0091 j=iflx((lu/v+0.5)*2+1
1.0092 flag(ll)=statyp(3,j)
1.0093 1 write(6,9000)*flag(ll)
1.0094 9000 format(ll)
1.0095 91 ll=0
1.0096 do 85 jj=1,class3
1.0097 90 ll=ll+1
1.0098 if(ll-6)98,98,89
1.0099 98 if(kount(ll)-max)91,90,90
1.0100 89 ll=0
1.0101 go to 90
1.0102 91 call limit(ll,lim,11,12)
1.0103 94 do 95 j=1,12
1.0104 if(flag(jj)-list(j,2))95,96,95
c handle class 2 statements (priority to type 3 type 2 type 1)

c
1.0116 200 if(class2)300,300,201
1.0117 201 do 2 ii=1,class2
1.0118 lu=lu+1x
1.0119 j=iflx((lu/v+0.5)*4)+1
1.0120 flag(ii)=stityp(2,i)
1.0121 2 write(6,9000)flag(ii)
1.0122 ii=0
1.0123 do 65 jj=1,class2
1.0124 70 ii=ii+1
1.0125 if(ii-6)75,75,77
1.0126 75 if(kount(ii)-max)71,70,70
1.0127 77 ii=0
1.0128 go to 70
1.0129 71 call limlt(ii,im,ii,12)
1.0130 call plck1(ii,12,flag,list,pick,indc,jj,lu,1x,v)
1.0131 j=pick
1.0132 if(indc)76,70,76
1.0133 76 if(type3)66,66,67
1.0134 67 list(j,2)=1077952576
1.0135 type3=type3-1
1.0136 kount(ii)=kount(ii)+1
1.0137 go to 65
1.0138 66 if(type2)69,69,68
1.0139 68 do 63 kk=3,20
1.0140 63 list(j,kk)=1077952576
1.0141 type2=type2-1
1.0142 kount(ii)=kount(ii)+1
1.0143 go to 65
1.0144 69 if(type1)65,65,62
1.0145 62 do 61 kk=1,50
1.0146 61 list(j,kk)=1077952576
1.0147 type1=type1-1
1.0148 kount(ii)=kount(ii)+1
1.0149 65 continue

c handle class 1 statements (priority to type 3 type 2 type 1)

c
1.0153 300 if(class1)400,400,301
1.0154 301 do 3 ii=1,class1
1.0155 lu=lu+1x
1.0156 j=iflx((lu/v+0.5)*4)+1
flag(i1)=statyp(1,i1)
write(6,9000)flag(i1)
li=0
do 25 jj=1, class1
i1=i1+1
if(i1-6)35,35,34
if(kount(i1)-max)31,30,30
i1=0
i1=0
go to 30
31 call limlt(i1,i1m,1,12)
call pickl(i1,12,flag,1list,1pick,1ndct,jj,lu,1x,v)
j=1pick
if(1ndct)36,30,36
36 if(flag(jj)-statyp(1,4))52,29,52
52 if(type3)26,26,27
27 list(j,2)=1077952576
type3=type3-1
kount(ii)=kount(i1)+1
go to 25
26 if(type2)29,29,29
28 do 23 kk=3,20
23 1list(j,kk)=1077952576
type2=type2-1
kount(i1)=kount(i1)+1
go to 25
22 if(type1)25,25,22
21 do 21 kk=1,50
21 list(j,kk)=1077952576
type1=type1-1
kount(i1)=kount(i1)+1
25 continue
20 see if there is room for phoney line(s).
insert phoney line(s).
400 if(junk)99,99,11
11 do 10 li=1,junk
lu=lu*1x
iran1=iflx((lu/v+0.5)*kfl1e)+1
if(kount(iran)-max)12,10,10
12 lu=lu*1x
iran3=iflx((lu/v+0.5)*3)+1
lu=lu*1x
iran6=iflx((lu/v+0.5)*10)+1
call limlt(iran,1lm,1,12)
jj=1
flag(jj)=statyp(1,iran3)
call pickl(i1,12,flag,1list,1pick,1ndct,jj,lu,1x,v)
j=1pick
if(1ndct)17,10,17
17 do 18 kk=2,50
18 list(j,kk)=phoney(iran6,kk-1)
kount(iran)=kount(iran)+1
10 continue
c put miscellaneous error in each of the files
    c provided the maximum parameter is not exceeded.
    c segment of line is blanked out.

    do 50 lik=1,2
    do 40 ii=1,kfile
    if(kount(ii)=max)41,40,41

1.0219  41 call limit(ii,lim,11,12)
    do 49 jj=11,12
    if(1list(jj,2)+958998208)42,43,44

1.0222  42 if(1list(jj,2)+10574304)49,51,49
1.0223  43 do 47 k=5,20
1.0224  44 if(1list(jj,k)+482328512)44,47,44
1.0225  45 if(1list(jj,k)+700432320)45,47,45
1.0226  46 if(1list(jj,k)+499105728)46,48,46
1.0227  47 continue
1.0228  48 kount(ii)=kount(ii)+1
    1list(jj,k+5)=1077952576
1.0231  49 go to 40
1.0232  50 do 53 k=3,30
1.0233  51 if(1list(jj,k)-2118139968)54,55,56
1.0234  52 if(1list(jj,k)-1279279168)56,55,57
1.0235  53 if(1list(jj,k)-1849704512)53,55,53
1.0236  54 1list(jj,k)=1077952576
1.0237  55 kount(ii)=kount(ii)+1
1.0238  56 go to 40
1.0239  57 continue
1.0240  58 49 continue
1.0241  59 continue
1.0242  60 continue
1.0243  61 continue
1.0244  62 c programs prepared
1.0245  63 c
1.0246  64 write(10,1000)((1list(1,j),j=1,50),l=1,136)
1.0247  65 c finished
1.0248  66 c
1.0249  67 c
1.0250  68 stop
1.0251  69 end
1.0252  70 /fnc
1.0253  subroutine limit(ii,lim,11,12)
1.0254  dimension  lim(10)
1.0255  if(ii=1,2,1,2
1.0256  1 ii=1
1.0257  2 ii=1lim(ii-1)+1
1.0258  3 return
1.0259  end
subroutine pick1(i1,12,flag,1ist,lpick,indct,jj,lu,ix,v)

dimension keep(10),l1st(136,50)
integer flag(10)
do 101 k=1,10
101 keep(k)=0

indct=0
1=1

do 100 j=1,12
100 if(flag(jj)-l1st(j,2))100,102,100
102 keep(1)=j
ktotal=ktotal+1
1=1+1
indct=1
continue
103 if(indct)103,104,103
104 return
end

/data
m.0072 begin activity.
this program is designed to input n integers one at a time and compute whether or not it is a prime number or is composite. In either case, output the number with an appropriate label.

Input n integers, one at a time and compute whether or not it is a prime number or is composite. In either case, output the number with an appropriate label.

for l=2 to k-1 step 1
if k/l=int(k/l) then 18
next 1
print k;" is a prime"
s10 to 19
print k;" is composite"
let m=n-1
if n>0 then 12
stop
end

this program is designed to take the square root of five numbers. Program must be able to handle negative and positive numbers. After updating, check results by running using negative and positive data.

let n=5
for l=1 to n step 1
input k
if k<0 then 16
print "the square root of ";k;" is";sqr(k)
s10 to 17
print "the square root of ";k;" is";sqr(-k);" i"
next 1
stop
end

this program is designed to sort the first twelve numbers of an array into ascending order. Numbers are read in as data. Program should sort the data and print out the results.
1.0051 10 dim a(12)
1.0052 11 for i=1 to 12 step 1
1.0053 12 read a(i)
1.0054 13 next i
1.0055 14 data 11,7,3,13,2,17,5,31,19,29,23,37
1.0056 15 for i=1 to 11 step 1
1.0057 16 for k=i+1 to 12 step 1
1.0058 17 if a(i)<a(k) then 21
1.0059 18 let n=a(i)
1.0060 19 let a(i)=a(k)
1.0061 20 let a(k)=n
1.0062 21 next k
1.0063 22 next i
1.0064 23 let i=1
1.0065 24 print a(i);a(i+1);a(i+2);a(i+3);a(i+4);a(i+5)
1.0066 25 if i=7 then 28
1.0067 26 let i=i+6
1.0068 27 goto 24
1.0069 28 stop
1.0070 29 end
1.0071 \!Basic
1.0072 2 rem
1.0073 3 rem
1.0074 4 rem this program is designed to compute the
1.0075 5 rem interest for three different principles to
1.0076 6 rem be inputted via the console with a time of
1.0077 7 rem three years at a rate of four per cent.
1.0078 8 rem
1.0079 9 rem
1.0080 10 read n
1.0081 11 data 3
1.0082 12 input k
1.0083 13 let a=4
1.0084 14 let i=3
1.0085 15 let n=n-1
1.0086 16 print k;i;a;k*i*a/100
1.0087 17 if n=0 then 19
1.0088 18 goto 12
1.0089 19 stop
1.0090 20 end
1.0091 \!Basic
1.0092 2 rem this program is designed to create two sums
1.0093 3 rem by adding 12 elements of an array with six
1.0094 4 rem elements comprising each sum, one sum is the
1.0095 5 rem first, third, fifth... elements and the
1.0096 6 rem second sum is the sum of the second, fourth
1.0097 7 rem sixth... elements. If either sum becomes
1.0098 8 rem negative, print an error and stop.
1.0099 9 rem
1.0100 10 dim a(12)
1.0101 11 for i=1 to 12 step 1
1.0102 12 read a(i)
1.0103 13 next
1.0104 14 for i=1 to 11 step 2
1.0105 15 let k=a(i)+k
1.0106 16 if k<0 then 23
1.0107 17 let n=a(i+1)+n
1.0108 18 if n<0 then 23
1.0109 19 next
1.0110 20 print "odd elements ";k;" even elements ";n
1.0111 21 end
1.0112 data 2,4,1,3,0,7,8,-5,-6,11,7,-10
1.0113 22 print "negative sum encountered"
1.0114 24 goto 21
1.0115 25 end
1.0116 /basic
1.0117 2 rem
1.0118 3 rem
1.0119 4 rem this program is designed to create the first
1.0120 5 rem ten numbers of the fibonacci sequence. given
1.0121 6 rem two start values, each number is gotten by
1.0122 7 rem summing the two previous numbers.
1.0123 8 rem
1.0124 9 rem
1.0125 10 let n(1)=0
1.0126 11 let n(2)=1
1.0127 12 for i=3 to 10 step 1
1.0128 13 let n(i)=n(i-1)+n(i-2)
1.0129 14 next
1.0130 15 let k=1
1.0131 16 print n(k)
1.0132 17 if k=10 then 20
1.0133 18 let k=k+1
1.0134 19 goto 16
1.0135 20 stop
1.0136 21 end
m.0072 begin activity.
This program is designed to take the square root of five numbers. Program must be able to handle negative and positive numbers. After updating, check results by running using negative and positive data.

```
10 let n=5
11 for k=1 to n step 1
12 Input k
13 if k<0 then 16
14 print "the square root of ";k;" is";sqr(k)
15 goto 17
16 print "the square root of ";k;" is";sqr(-k);" !"
17 next l
18 stop
19 end
20 /end
```

---

**Input**

```
重重73 action in progress.
m.0070 action complete.
/mend
m.0072 begin activity.
/update
/display
```

**Run**

```
/insert exam22
m.0073 action In progress.
```

---

**Diagnostics**

```
no statements flagged in this compile
```

```
the square root of -25 Is 5
32 the square root of 32 Is 5.65685
16 the square root of 16 Is 4
-16 the square root of -16 Is 4
0 the square root of 0 Is 0
```
This program is designed to take the square root of five numbers. Program must be able to handle negative and positive numbers. After updating, check results by running using negative and positive data.

```
1.0010 10 let n=5
1.0011 11 for l=1 to step
1.0012 12 input k
1.0013 13 if k<0 then 16
1.0014 14 goto 14
1.0015 15 goto 14
1.0016 17 next l
1.0018 18 stop
1.0019 19 end
1.0020 /end
change 11,11
11 for l=1 to n step 1
/change 14,15
14 print "the square root of ",k," is ",sqr(k)
15 goto 17
/change 16,16
16 print "the square root of ",k," is ",-sqr(-k),"!
/end
```

---

```
......
```

---

```
32  the square root of 32  is  5.65685
25  the square root of -25  is  5  i
25  the square root of 25  is  5
100 the square root of 100  is  10
0   the square root of 0   is  0
```

---
this program is designed to create the first ten numbers of the fibonacci sequence, given two start values, each number is gotten by summing the two previous numbers.

```
1.0001 /basic
1.0002 2 rem
1.0003 3 rem
1.0004 4 rem
1.0005 5 rem
1.0006 6 rem
1.0007 7 rem
1.0008 8 rem
1.0009 9 rem
1.0010 10 let n(1)=0
1.0011 11 let n(2)=1
1.0012 12 for i=1 to step
1.0013 13 let n(i)=n(i-1)+n(i-2)
1.0014 14 next i
1.0015 15 let k=1
1.0016 16 print n(k)
1.0017
1.0018 18 let k=k+1
1.0019 19 goto 16
1.0020 20 stop
1.0021 21 end
1.0022 /end
/change 12,12
12 for i=3 to 10 step 1
/change 17,17
17 if k=10 then 20
/end
```

```
This program is designed to create the first ten numbers of the Fibonacci sequence, given two start values, each number is gotten by summing the two previous numbers.

```
10 let n(2)=1
11 let n(1)=n(1-1)+n(1-2)
```

Then:

```
12 for l=3 to 10 step 1
13 let n(1)=n(1-1)+n(1-2)
end
```

Loop:
```
14 next l
```

Go to 13 if k is equal to 10:
```
15 let k=k+1
16 print n(k)
17 if k=10 then 20
```

End:
```
18 next l
```

Go to 16 if k is equal to 10:
```
19 goto 16
```

End:
```
20 stop
```

No statements flagged in this compile.
/run
m.0073 action in progress.
*** diagnostics ***
  no statements flagged in this compile
0
1
2
3
5
8
13
21
34
m.0072 begin activity.
APPENDIX E

PROGRAM TO SEPARATE PREPARED UNSOLVED PROBLEMS
/display slice
m.0073 action in progress.
1.0001 /ftc
1.0002 c
1.0003 c program to separate files given their limits
1.0004 c
1.0005 dimension a(136,25)
1.0006 read(5,1000)((a(i,j),j=1,25),i=1,136)
1.0007 1000 format(25a3)
1.0008 write(6,3000)
1.0009 3000 format('type in limits')
1.0010 read(9,2000)11,12
1.0011 2000 format(213)
1.0012 write(10,1000)((a(i,j),j=1,25),i=11,1:)
1.0013 stop
1.0014 end
1.0015 /data
m.0072 begin activity.