AN ANALYSIS AND DEVELOPMENT OF A COMPUTER SCIENCE PROGRAM FOR USE IN SECONDARY SCHOOL MATHEMATICS

By

ROBERT D. GUMM

Bachelor of Science
Fort Hays Kansas State College
Hays, Kansas
1955

Master of Science
University of Kansas
Lawrence, Kansas
1961

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the degree of
DOCTOR OF EDUCATION
August, 1969

AN ANALYSIS AND DEVELOPMENT OF A COMPUTER SCIENCE PROGRAM FOR USE IN SECONDARY SCHOOL MATHEMATICS

Thesis Approved:

Kenneth E. Wegner
Thesis Adviser

Victor A. Berg

Lawrence D. Olson

D. D. Durham
Dean of the Graduate College

PREFACE AND ACKNOWLEDGMENTS

The purpose of the study was to develop a computer related course to be used in conjunction with the mathematics curriculum at the secondary school level and to describe some of the alternate computer systems necessary to implement the proposed course of study.

The writer is grateful to Dr. Kenneth E. Wiggins, Chairman of the Advisory Committee for friendly encouragement and guidance in this research. The writer also wishes to express sincere appreciation to Dr. Russell Dobson, Dr. Milton Berg, and Dr. Jack Walden for assistance and advice as members of the committee.

Special thanks are due to Mr. Don Dean and Mr. John Hundley of the IBM Corporation for their assistance in providing information on available computer system and educational materials.

The writer is especially indebted to those who gave the most support and encouragement during the course of study, his children and his wife, Beverly.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Today, over 30,000 electronic digital computers are reported in use. The production rate is near 500 machines per month with a projection of computer installations in the United States to be 60,000 by 1970 and 85,000 by 1975.[1] The phenomenal growth of this technological complex industry places a serious demand for well trained men and women. It is estimated that there are approximately 500,000 trained personnel needed today in the field of computing with a projected need for 650,000 by 1970.[2]

Colleges and universities have responded to this continual advancement of digital computers by initiating instruction in computer science during the past decade. This impact of digital computers has now reached the secondary schools. Numerous secondary schools are realizing that computational facilities can be used in many ways.

Digital computers can be useful in the secondary school level in at least three ways: by students in conjunction with courses; by students in vocational and technical programs; and by the school administration. This study is primarily concerned with the first of these,

---

[1] Berkely, Edmund C., The Reference Diary of the Library of Computer and Information Sciences, The Library of Computer and Information Science, 1966.

[2] Carroll, John M., Careers and Opportunities in Computer Science, E. P. Dulton and Co., New York, 1967.

and specifically with respect to current mathematics curricula.

It is obvious that there exists a need for many well trained professional and technical people in the field of computing. The introduction of computer usage at the secondary school level in vocational and technical programs should help to provide for this demand.[3] There is another important and currently not too well recognized aspect of computer usage, the enrichment of the curriculum for secondary school students. In most subject areas there are problems or projects in which a computer can be effectively utilized to enlarge the vistas of the subject being taught. In particular, by making use of the computer, mathematics students may solve more meaningful problems which, in turn, will lead to better understanding of previous concepts learned in mathematics and to some insight to more advanced mathematical concepts. Hence, the use of the computer would enrich and extend the student's understanding of mathematics by his experiencing certain events which would otherwise be beyond his reach.

Purpose and Significance of the Study

The implementation of a computer science program at the secondary level primarily involves three major problems:

(1) lack of computer facilities;

(2) lack of trained teachers;

(3) lack of course materials at the secondary level.

Some secondary schools have computer facilities available now. Also, the advent of time-sharing capabilities, the availability of

---

[3]Tonne, Herbert A., Data Processing Instruction In High School, The Journal of Business Education, October, 1964.

second-generation computers, and the proliferation of small to medium-size computers throughout industry, colleges, and universities which schools can have access to at reasonable cost, have placed computers within the reach of the secondary schools. The fact that most teachers trained in the field of mathematics in the past five years and those being trained at the present time have training in computer science should provide teachers with at least some experience with computers. The federal government has recognized the need for trained teachers in computer science at the secondary level and is providing training through summer institutes and in-service institutes sponsored by the National Science Foundation. The problem, then, is not so much how to get computers into the high school or who will teach computing, but how to use them once they are there.

Hence, this study is two-fold:

(1) to develop a one-semester course of study that would serve as a text and an aid to secondary school teachers of mathematics who want to teach an introductory computer science class which would at the same time enrich and extend the student's understanding of mathematics;

(2) to identify several feasible alternatives to the implementation of the developed course of study at the secondary level.

It is believed that this study will have significance for computer science and mathematics education at the secondary school level in that it will show how the computer can be used to give a better understanding of some mathematical concepts and at the same time, use mathematics to teach some computer fundamentals. The course will be

so designed as to give students the opportunity to inquire into and discover new concepts in mathematics by the use of the computer much like the laboratory assists the student in physics or chemistry.

Realizing computers play an essential role in the operation of our modern society and affect almost every area of human intellectual endeavors, the course will provide an opportunity for students to acquire a knowledge of computers that will help them be better informed and more productive citizens.

Further, this study should be beneficial to those secondary school systems interested in seeking computer facilities as it will provide the necessary information and costs for establishing such a facility.

Use of Terms

Selected terms used in this study are defined as follows:

Computer. A large scale electronic digital computer.

Computer Science. That discipline involving the design of computers, the art of numerical mathematics, and the representation, storage, manipulation, and presentation of information.

Processing. Reading in a computer program and data by some input device, executing the instructions, and printing on paper or punching in a card the requested information by some output device.

Batch Processing. The processing of several jobs one after the other in a continual job stream.

Terminal. A device remote from a large computer system capable of sending and receiving information from the central

computer facility.

Time-Sharing. A system which shares the computer between all of the users (terminals) at a given time.

Turnaround Time. The elapsed time required from the time of submittal of a program until the program and results are returned to the user.

Limitations

The following will be limitations to the study:

(1) the use of the computer in only the instructional program of the secondary school;

(2) the applications of the computer to only one discipline, mathematics;

(3) the course will be developed in theory and will not be validated by actual presentation to a class.

CHAPTER II

SELECTED REVIEW OF RELATED LITERATURE

Within the past ten years there has been an increasing interest on the part of educators and students in the utilization of computers. This interest in computers has spread throughout the educational system with some high schools introducing computers to their students. The need for the introduction of computing at the secondary level is supported by a report of the High School Advisory Board of the School Mathematics Study Group which warned:

> The number of individuals who will work with or be directly affected by digital computers in the next decade will be so great that the entire high school population needs to learn something about computers.[1]

The introduction of data processing at the secondary level during the last five years has contributed to satisfying this need. However, most programs in data processing at this level have not emphasized the computer and its application to the sciences but to the application of unit record equipment as it relates to business applications.[2]

Schilhab further supports the need for training in computing in his statement:

---

[1]Forsythe, Alexander, Math and Computing in High Schools: A Bethrothal, Mathematics Teacher, Vol. 57, No. 1, January, 1964.

[2]Carlberg, Mona, Survey of Data Processing Instruction in Six High Schools, The Journal of Business Education, March, 1966.

> Since business and industry have taken advantage of computers, it seems essential for students of all districts to be acquainted with this innovation for vocational reasons as well as to discharge their responsibilities as informed citizens.[3]

He also relates the use of computers to the field of mathematics as a means of removing the limitations created by the lack of aids in solving complex mathematics problems. For years, the only aids in solving mathematical problems were logarithms, slide rule, or a desk calculator. With the digital computer, students may now solve more meaningful problems and this, in turn, will lead to better understanding of mathematical and logical concepts.

Forsythe[4] supports the idea of the introduction of computers in mathematics courses because of the numerous algorithms and computations which are explicit or implicit in all mathematics courses. Since the computer, by use of a machine-independent computer language, has been designed for the easy formulation of algorithms, its use in this application can contribute a great deal to the training of mathematics students.

Fred Gruenberger,[5] a mathematician for RAND Corporation, relates the learning of computing to that of a foreign language. He further emphasizes the fact that it has been shown that foreign languages are best learned young, and that this is one of the strongest arguments for introducing computing in high school. He lists three other

---

[3] Schilhab, Elgin, Digital Computing Drops Into The Curriculum Pond, Texas Outlook, Vol. 51, May, 1967.

[4] Forsythe.

[5] Gruenberger, Fred, Computing In The Secondary Schools, DATAMATION, May, 1964.

arguments for introducing this subject in the secondary schools. They are:

(1) the high school student has less preconceptions and misinformation that will only have to be erased;

(2) the college-bound student who has had some computer training in high school has a distinct advantage on his arrival on campus. It is becoming very difficult these days to find a college that is computer-free;

(3) the computer can be used as a teaching device to make the learning of other things (e.g., mathematics) easier or more attractive. There is also a prestige and reward factor to consider in having a computer course available in the senior year.[6]

Perhaps it is too early to try to define exactly what about computers and computing should be taught to high school students, but certainly a wide range of curricula and many diverse approaches should be sought.

---

[6] Ibid.

CHAPTER III

PLAN OF DEVELOPMENT

The content of the course of study and the alternatives for implementation will be determined by the following:

(1) a survey of the literature in the field to help identify the content appropriate for use in secondary schools;

(2) the experience of the author in teaching mathematics and computer science at the secondary and college level;

(3) consultation with different computer manufacturers to determine availability of second-generation computers;

(4) consultations with computer manufacturers and computer centers capable of providing remote terminal capabilities;

(5) consultation with computer centers in industry, colleges, and universities on availability of machine-usage for secondary schools.

A preliminary survey of the existing literature and the experience of the author in teaching mathematics and computer science at the secondary and college level have been used to establish the rudimentary outline of the course.

An introduction to the algorithmic language, FORTRAN, would constitute the first portion of the course. To illustrate the use of the different FORTRAN statements, various mathematical concepts previously studied by the student would be used. Some of these concepts would

include:

From Informal Mathematics

Construction of compound interest tables
Salary problems
Finding square roots by using approximations
Modular arithmetic
Non-commutative arithmetic

From Algebra

Solving quadratic equations
Solving systems of linear equations
Evaluating determinants

From Geometry

Solving right triangles by Pythagorean Theorem
Calculating approximation to π
Calculating areas and volumes

From Trigonometry and Advanced Mathematics

Testing validity of trigonometric Identities
Solving triangles
Matrix addition and multiplication
Statistical calculations (means, median, standard deviation)

After some level of competency is obtained in the use of the various FORTRAN statements, further practice and techniques in programming would be obtained by the use of the computer in studying some more sophisticated mathematical problems. Some of these concepts would include:

Determining if a point lies inside, on, or outside a given circle.
Determine slope of any line given two points on the line.
Determine Y-intercept of any line.
Determine if lines are parallel or same line.
Determine if given points are vertices of square, rectangle, rhombus, or quadralateral.
Approximating various functions by use of infinite series (sin, cos, log, etc.).
Consider sums of infinite series (converging and non-converging).
Computing derivatives (by considering definition as $\Delta x \to 0$).
Topics from number theory.
Elementary topics from numerical analysis.
Numerical integration.

CHAPTER IV

CONTENT ANALYSES AND PROGRAM DEVELOPMENT

Analysis and Development of An Appropriate Computer Language

This course of study is written for use in the secondary schools at the twelfth grade level for students of mathematics desiring to learn something about the use of an electronic digital computer. The twelfth grade is suggested because the course is so written as to require a mathematical background necessary to begin an introductory course in calculus. Also, it is the opinion of the author that the course content would require a certain level of maturity among the students if the course is to be successful. The material is so written as to make use of the computer in solving meaningful mathematical problems and as an aid in understanding abstract mathematical concepts generally introduced in secondary and undergraduate mathematics.

In order to use the computer one must learn an acceptable language that the computer can understand. One way is to learn the particular machine language of a certain computer. However, this would leave the student with the ability to program only one computer which would not be very practical indeed. An alternate solution is to learn a language that can be used on any computer requiring only a compiler program for that language which enables the compiler to translate from this language to its own machine language. One of the more common such languages in use today, especially in scientific applications, is

the FORTRAN language. Although machine languages are highly individual for each computer, the FORTRAN language is really quite uniform. It is designed as a machine-independent language suitable for all computers. However, there are some computer-to-computer differences which can be determined by examining any computer manufacturer's FORTRAN Reference Manual. There are many versions of FORTRAN, commonly named BASIC FORTRAN, FORTRAN II or FORTRAN IV. This course of study describes primarily a subset of FORTRAN IV with particular emphasis given on the use of the computer as an aid to solve problems and to give enrichment to the mathematics curricula. It is the intention of the author to keep this subset as small as possible so the student can concentrate on the use of the computer as an aid to study mathematics rather than attempting to train professional FORTRAN programmers.

Functional Units

There are five basic functional units in any large scale electronic digital computer as shown in Figure 1.

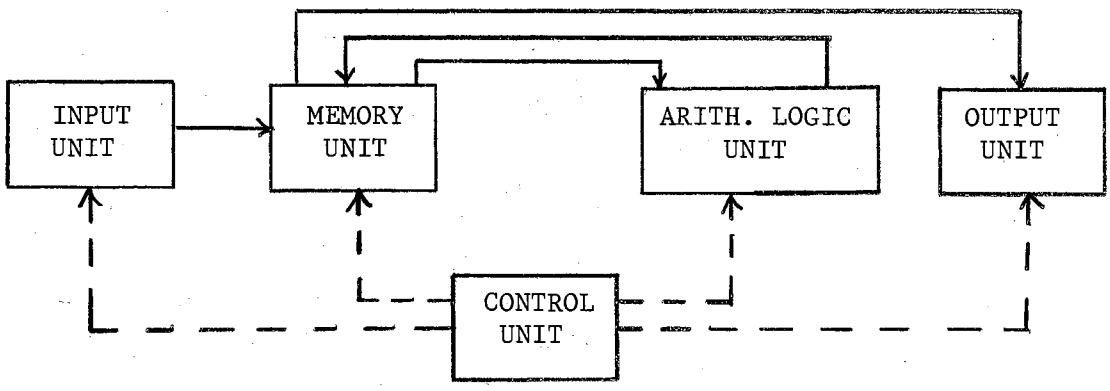


Figure 1. Functional Units

The computer is an organized grouping of these functional units with the units working together to perform certain tasks at remarkable speeds. The purpose of each unit is briefly described below.

Arithmetic-Logic Unit. The arithmetic-logic unit serves as the desk calculator of the computer. It performs the four basic operations of arithmetic; addition, subtraction, multiplication, and division. In addition to doing arithmetic, the arithmetic-logic unit can perform simple logical operations which consist of examining a number and determining whether it is zero or not and it can examine the sign of a number and determine whether it is positive or negative. This unit can do these operations at very high speeds and with perfect accuracy. This is its only function.

Memory Unit. One of the memory unit's main function is to supply the arithmetic-logic unit with numbers and to transport them away again at speeds comparable to arithmetic speeds. It has the following properties: it can retain electronic representations of numbers for long periods of time; it can transmit a number to the arithmetic-logic unit in a few millionths of a second; it can receive a number from the arithmetic-logic unit and store it, in a few millionths of a second.

The memory unit must perform an additional function to the above function. During the solution of a problem, it may be necessary to remember thousands of numbers. The memory unit must be able to identify the values it has stored so it can extract and transmit exactly the one required by the arithmetic-logic unit upon request. This identification is as important as the numeric value associated with it.

The identification problem is solved in the memory unit by being organized into many individual "cells" or "memory locations." Each

cell will hold a fixed number of digits (varies with the computer). Each cell is labeled with an unique "identifier." Whenever a number is stored in a memory cell having a particular identifier, the number can be recovered by referring to the particular identifier. To illustrate, memory may be visualized as shown in Figure 2.

A	X	SUM	AREA	ROOT	NUM
23.1	57.83	80.93	51.2	0.25	24

Figure 2. Memory Cells and Identifiers

The identifier is the quantity shown at the top of each cell: A, X, SUM, AREA, ROOT, NUM, etc. The value of the number stored in each cell respectively is: 31, 57.83, 100.21, 51.2, 0.25, 24, etc. The selection of the names for the memory cell is entirely arbitrary. If we instruct the computer to add the numbers in memory cells A and X, store in memory cell SUM, this result will be saved for later use. Also, the number residing in a memory cell is never destroyed unless the programmer instructs the computer to place a number in that memory cell. For example, if instruction is given to add the numbers in memory cells AREA and ROOT, store the result in memory cell SUM, then memory would look as follows in Figure 3:

A	X	SUM	AREA	ROOT	NUM
23.1	57.83	51.45	51.2	0.25	24

Figure 3. Memory Cells and Identifiers

which particular cell the computer selects is of no importance to the programmer as long as it doesn't assign two different numbers to the same cell. The computer performs this function with complete accuracy. One can think of the memory cells as having "blank" names at the beginning of a problem and as the programmer selects names as cells are needed, the computer places numbers in the appropriate cells for further use.

Control Unit. The memory unit was described as the unit that can provide the arithmetic-logic units at speeds comparable to that of the arithmetic-logic unit. The question is raised: how does the arithmetic-logic unit know what calculations are to be performed? There must be some way to give instructions to the computer. This is the key element which makes the modern computer possible. This problem is resolved by storing the instructions to the computer in the memory unit like problem data. Then these instructions can be retrieved from memory and executed at speeds comparable to the arithmetic speeds. Hence the memory unit serves a dual purpose: stores both problem data (results from calculations) and instructions to the computer. The unit which obtains the instructions from the memory unit, interprets them, and directs the arithmetic-logic unit, is called the control unit. The control unit is to the computer as the human

operator is to a desk calculator. The desk calculator operator follows a list of instructions and translates them into various operations by use of the keyboard. Similarly, control unit takes instructions from a "list" stored in the memory unit and translates them into electrical signals, causing the arithmetic-logic unit to perform the desired operations. Likewise, the control unit includes all the circuitry necessary to perform all functions throughout the computing system.

The question of how the control unit selects the correct instruction at the right time generally is raised. This is done by having all the different instructions to the computer stored internally in sequence, one after the other, in a string of memory cells. The first instruction step is stored in a predetermined cell in memory so that the control unit can go automatically to this cell, retreive the statement, interpret it at electronic speeds, and direct the other units to perform the requested operation. Since the instructions are stored sequentially, the control unit can automatically determine the next instruction. It can then execute each instruction in sequence, one at a time, at the rate of hundreds of thousands each second until the last statement or instruction is executed.

This list of instructions referred to above is called a program. In the case of the FORTRAN language, this program is called the source program. The exact form of these statements or instructions will be discussed later.

Input Unit. The "list" of statements or instructions discussed above and any problem data must somehow be "read" by the computer. The computer cannot read from paper, but can only read from its memory.

The means for rapidly converting alphabetic characters and numbers into a form the computer can accept is done most often by the ordinary "tab card" shown by Figure 4.

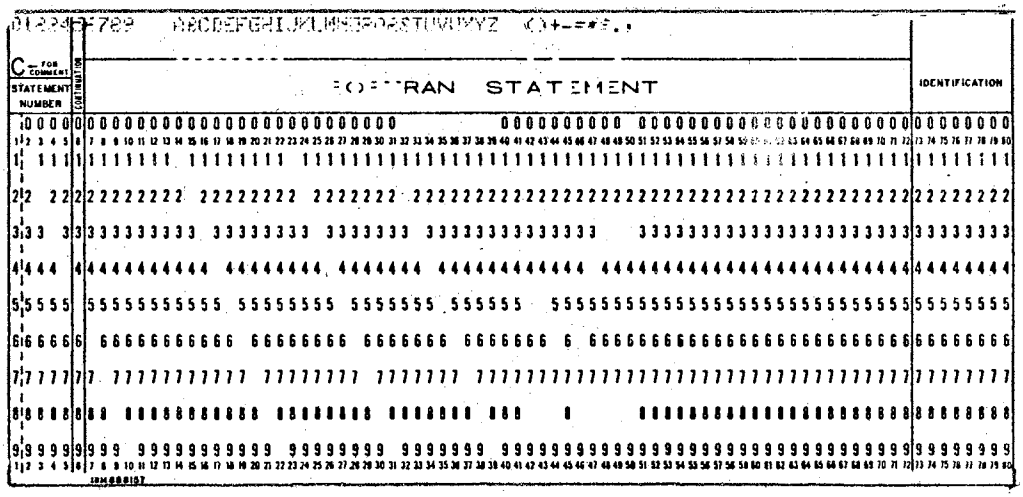


Figure 4. A Punched Card

Each card has 12 rows numbered as indicated and 80 columns numbered from left to right. Each column has the capacity to contain one number, letter, or special character in coded form. The digits 0 1 2 3 4 5 6 7 8 9 are referred to as numeric characters, the letters A B C ... W X Y Z are called alphabetic characters with all other characters called special characters. All acceptable characters numeric, alphabetic, or special are called alphameric characters.

These cards are punched on a machine called a keypunch very much

like a typewriter.

In order for the computer to read the information from the card, it must be equipped with an electromechanical device which can "read" punched cards and convert the "holes" into electrical impulses. This device is called a card reader which can read and transmit up to 64,000 characters (800 cards) per minute to the computer's memory.

The card reader is only one of several input units used today. Below is a list of some other devices which can be called input units:

Paper tape reader
Magnetic tape reader
Magnetic character reader
Optical scanner

The input unit's only job is to read information from outside the computer, properly convert it, and transmit it to the memory unit.

Output Unit. Having performed the instructions and calculations the computer must have a means of providing the requested results to the outside world in some recognizable form.

The computer must be equipped with a device which can now "read" the electronic representations of numbers from the memory unit and convert them into readable form.

The computer must be equipped with a device which can now "read" the electronic representations of numbers from the memory unit and convert them into readable form. The unit which performs this function is called the output unit. This unit reverses the process of the input unit. The output unit most commonly used is the on-line printer which prints out the information using the alphameric characters at the rate of up to 145,200 characters (1100 lines, 132 characters per line) per minute.

Other output units use punched cards, punched paper tape, or

magnetic tape.

Instruction Statements

As was explained previously, the key element which makes the modern computer possible is the ability to store instructions in memory to be retrieved and executed so that the computer can carry out the desired tasks. These instructions in any procedure-oriented computer language is in the form of a statement. In FORTRAN, every instruction statement has two parts:

(1) Statement Number. This number provides a unique "tag" for any statement in a program. This number is used by the program when it is necessary to refer to a particular statement. Normally only those statements which must be referred to within the program are numbered. The statement number may be 1 or any greater number not exceeding 5 digits. The maximum statement number allowed varies with the compiler for the particular computer in use. Generally, this is 99999. Statement numbers may be assigned without regard to order or sequence, but no two statements may be assigned the same statement number. The statement number is keypunched on a FORTRAN statement card right justified in columns 1 to 5. Right justified means the statement number is placed in the five-column field so that the units digit is in column 5.

(2) Statement Body. This is the main part of an instruction statement. It tells the computer what operations to perform. The statements vary in length, but must begin in column 7 on the FORTRAN statement card and may possibly extent to column

72 inclusive. Each statement is on a separate card. It is possible to continue one statement on more than one card, but will be discussed in a later section.

Instructions statements are classified either executable or non-executable statements. The executable statements tell the computer what to do next. The nonexecutable statements are descriptive in purpose. As each of the FORTRAN instruction statements are described in later sections, they will be classified as either executable or nonexecutable.

Instruction statements are written in a list, one after the other. Together, they form a program. Each statement must be written with absolute accuracy. After placing the statements on the FORTRAN statement card as described above, the deck of cards is called the source deck. The order of the cards in the source deck is very important as they will read into memory and execute in that sequence.

An example of FORTRAN source deck is shown in Figure 5.

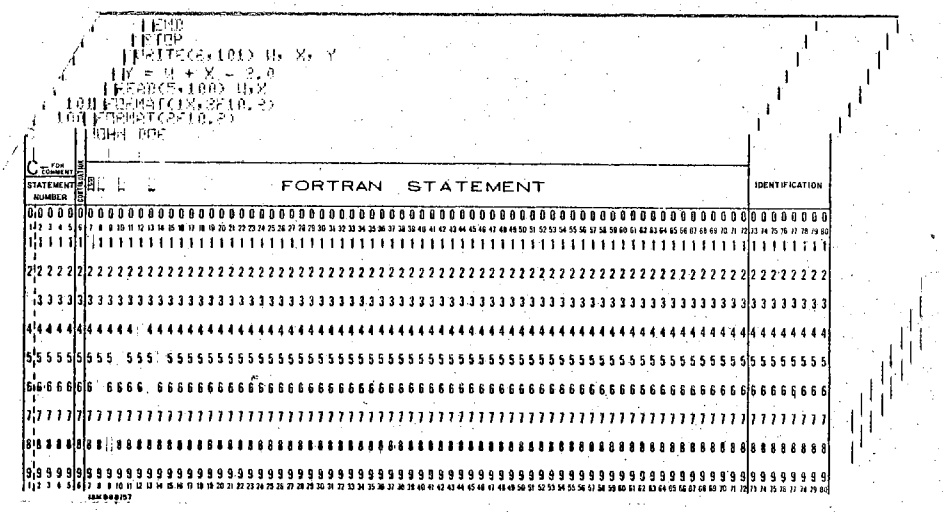


Figure 5. FORTRAN Source Deck

Exercise 1.

1.1 What is meant by an executable and nonexecutable statement?

1.2 What is the purpose of a statement number?

1.3 A number is to be placed in columns 10 to 20 right justified. Explain.

Data

One of the first considerations in writing a program is the input and output data. The input data is recorded on one of the previous mentioned medium and then is "read" into the memory unit. This study will restrict itself to using only the punched or tab card as the input medium.

There are two basic types of data; alphameric (alphebatic, numeric, or special characters) and numeric (numbers). Again, this study will restrict itself to numeric data.

An example of numeric data placed on a card is shown in Figure 6.

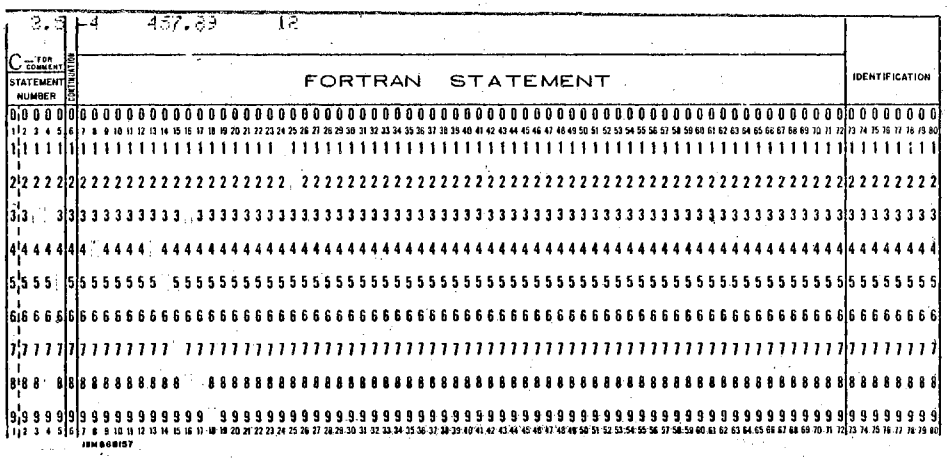


Figure 6. Data Card

The above example shows four separate divisions on the card. Each of these divisions is called a field. Hence, the first field consists of five columns and contains the real number 3.5; the second field consists of three columns and contains the integer -4; the third field consists of ten columns and contains the real number 457.89; the fourth field consists of seven columns and contains the integer 12.

All numeric data prepared for input on a tab card must conform to the following rules:

(1) Numeric values will have an algebraic sign. If a positive number, a + sign or blank may precede the number. If a negative number, the - sign is required.

(2) Blanks are not allowed embedded within a numeric quantity. i.e., 345␣6 (␣ denotes a blank column) would be read as 34506 instead of 3456 since blanks are generally read as zero.

(3) Numeric data may be real or integer (defined in next section).

Integer Data. An integer number is a decimal number (formed with digits 0 through 9 of the decimal system) with no decimal point. The number of digits in the integer and its magnitude are limited and depend on the particular computer in use (check computer manufacturer's FORTRAN Reference Manual). In general, any integer having one to five digits having a magnitude not exceeding 99999 will be acceptable. Some examples are as follows:

5 -325 0 10235 -57

The numbers described above are sometimes referred to as fixed numbers. Fixed in that the decimal point is always considered fixed immediately

following the last digit.

Real Data. A real number is a decimal number with a decimal point. The number of significant digits and its magnitude are limited and again depend on the particular computer in use. In general, any real number having one to six significant digits and its magnitude not exceeding limits $10^{\pm 38}$ will be acceptable. Some examples are as follows:

-4.1 .35 0.0 -8.

Real numbers can be in several different forms. This study is restricted to two different forms. In addition to the above form, real numbers can be in exponential form. For an example, consider the number

.5715E-03

The true value in this case is

$.5715 \times 10^{-3} = .0005714$

Hence, the purpose of the exponent is to relocate the decimal point in the real number. Further examples of numbers in exponential form acceptable both for input and output data are:

E-Form	True Value
2.73E+01	27.3
-0.512E-03	-.000512
5.0E+00	5.0
0.12345E+10	1234500000.

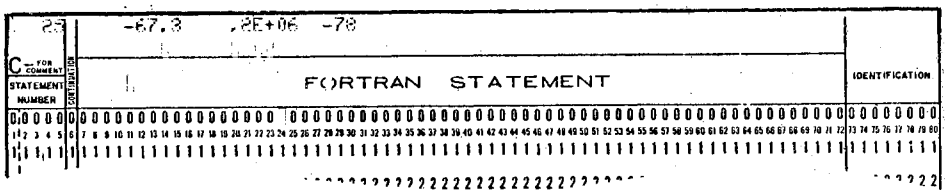
The exponential form is used when dealing with very small and very large real number. By using this form, one is assured of retaining a certain number of significant digits. Generally, the exponential form is used in output data only, but can be used for both input and output data.

Exercise 2.

2.1 What is the main difference between integer data and real data?

2.2 When is the exponential form of a real number generally used?

2.3 The following data card has how many different fields?



2.4 Identify the type of data in each of the fields in the above data card.

2.7 Give the true value of each of the following real data given in exponential form.

a. 35.1E-02 b. -.789E+06 c. 8.356E+00

Input/Output

Once integer or real data is placed on a card, the computer must "read" this information into its memory before any desired calculations or manipulation of this data can be performed. Likewise, a statement is needed to convert data from the form in which they are stored in memory into the form in which they are to appear in the output. To accomplish this task, the FORMAT, READ, and WRITE statements exist. For input the FORMAT statement contains information which is used by the compiler to determine the type data and the location of this data on the card. For output, the FORMAT statement contains information which is used in converting data from the form in which it is in memory into the form they are to appear in the output. The FORMAT statement is a nonexecutable statement which is used in conjunction with the READ and WRITE statements. The general form of the FORMAT statement is:

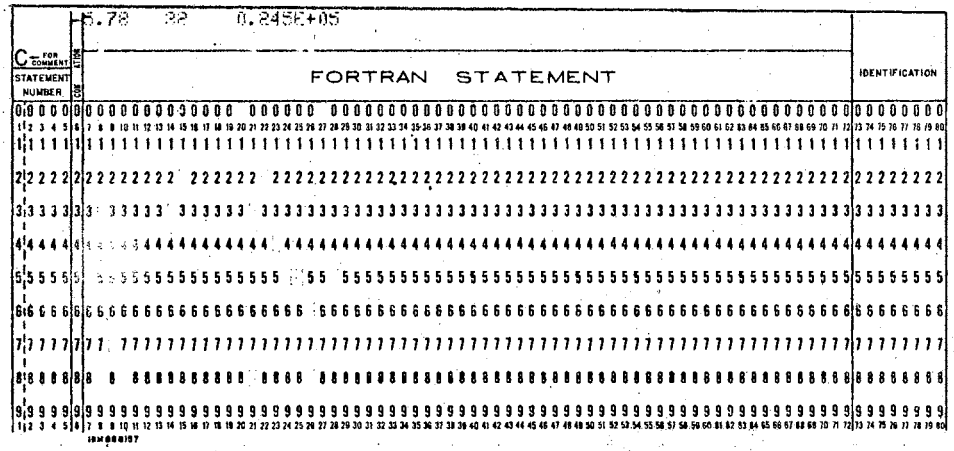
n FORMAT (s_1, s_2, s_3, ... , s_m)

The n is a statement number and the s_1, s_2, s_3, ... , s_m is an ordered series of specifications. The commas separating the specifications are required. Each specification describes the type of data and where each is located on the card. The order of the specifications must correspond to the order of the data on the input or output record.

This study will restrict discussion of only three specification forms for numeric data. The general forms are as follows:

	Specification	Type of Field
(1)	Iw	Integer
(2)	Fw.d	Real without exponent
(3)	Ew.d	Real with exponent

The I, F, and E indicate the type of field, w indicates the width of the field, and d indicates the position of the decimal point. For w, the number of columns in the entire field is given. For d, the number of digits desired to retain to the right of the decimal point. As an example consider the following card.



The card has three numeric fields, call them A, N, and B. A is punched in columns 1 through 10 and has the value of -5.78; N is punched in columns 11 through 15 and has the value of 32; B is punched in columns 16 through 28 and has the value of +0.245E+05. A and B are real data and N is integer data. A FORMAT statement that could be used to read in this card would be:

```
100 FORMAT (F10.2, I5, E13.3)
```

The 100 is the statement number and F10.2, I5, and E13.3 are the specifications describing the data fields on the card from left to right.

> F10.2 describes the first datum as real data with the field width of 10 columns and having 2 digits to the right of the decimal point.
>
> I5 describes the second datum as integer data with the field width of 5 columns.
>
> E13.3 describes the third datum as real data in exponential form with the field width of 13 columns and having 3 digits to the right of the decimal point.

More discussion and examples on the FORMAT statement and specifications will follow throughout the entire course of study.

READ Statement. The FORMAT statement describes to the computer the type data and where it is located on the card. There must be an instruction that will cause the computer to "read" this data from the card and place it in memory. The READ statement performs this task. The general form of the READ statement is as follows:

```
READ (i,n) list
```

Most computer systems have several different type of input units. The i designates the particular input unit. For this discussion, the number 5 will represent the card reader as the input unit. The card reader will be the only input unit used in this study.

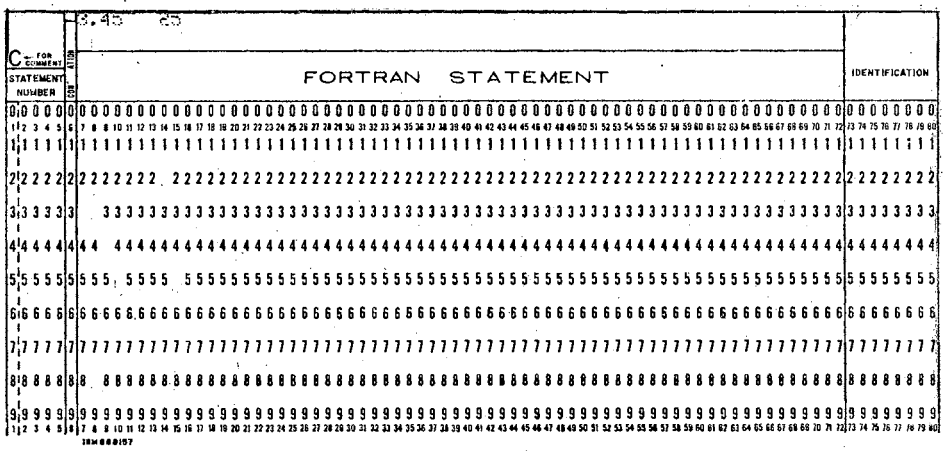
The n represents the statement number associated with the FORMAT statement which applies to the particular input record to be read.

For list the programmer substitutes an ordered series of variable names separated by commas. Valid variable names will be explained in the next section. For the time, single letters of the alphabet will be used for variable names. The letters I, J, K, L, M, and N represent integer quantities while all other letters of the alphabet stand for real quantities. The order of the list must be consistent with the left to right order of the data placed on the data card. When the READ statement is executed, numeric values are transferred from the punched card and appear in the selected memory cells labeled by the variable names in the list. These values remain in the memory cells indefinitely for use by other instruction statements, unless the program does something to change them.

As an example of a READ statement can a FORMAT statement, consider the following:

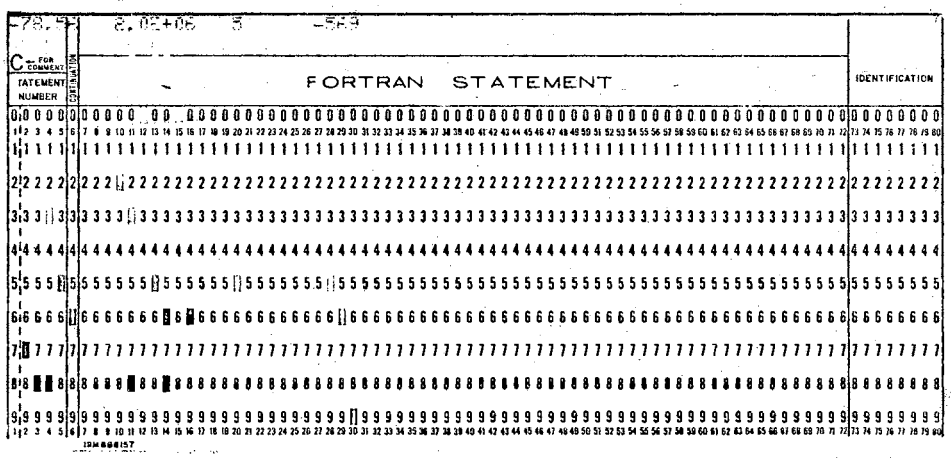
```
100 FORMAT (F10.2, I5)
    READ (5,100) A, N
```

The above READ statement indicates the input unit is to be the card reader (number 5). The FORMAT statement associated with this READ statement is the statement with statement number 100. The list contains two variables, A and N. Note A is a real variable and N is an integer variable. Hence, the specifications must agree with the corresponding variables. Thus F10.2 for the variable A and I5 for the variable N. The number for A is in columns 1-10 and the number for N is in columns 11-15. The particular data card might be as follows:



The FORMAT statement can be placed anywhere in the source program with only a few exceptions which will be discussed in later sections. That is, the FORMAT statement associated with a particular READ statement may be placed before or after the READ statement.

Now consider the following card having real numbers in the first two fields and integers in the next two fields.



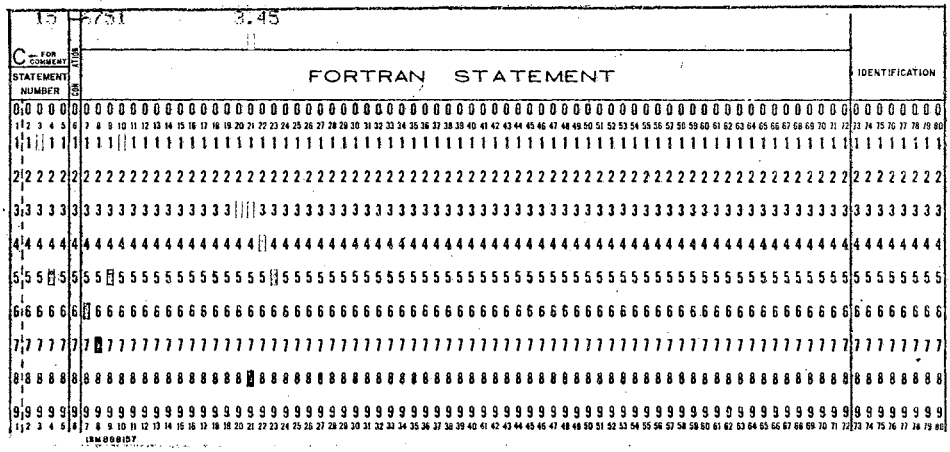
Appropriate READ and FORMAT statements would be as follows:

```
101 FORMAT (F6.2, E10.1, I4, I10)
    READ (5,101) X, Y, I, J
```

The selection of the variables X, Y, I, and J need only be based on the mode or type of numbers being read. i.e., real or integer. After this READ statement is executed, the real number -78.56 will be contained in memory cell named X, the real number 2.0E+06 will be contained in memory cell named Y, the integer 5 will be contained in memory cell named I, and the integer -569 will be contained in memory cell named J. Although reference is made to the variable name, the number associated with the variable name is what is used in the desired operations. Thus, another way to think of the above procedure is as follows:

X is defined as -78.56
Y is defined as +2.0E+06
I is defined as +5
J is defined as -569

As a last example consider the following card and statements:




```
102 FORMAT (I5, F10.3, F15.1)
    READ (5,102) K, A, B
```

After execution of the READ statement

K is defined as +150
A is defined as -6.751

B is defined as +3.4500

Careful note should be made of this example. K is defined as 150 instead of 15 since the number 15 is not right-justified in the field and a blank is read as a zero by most compilers. A is defined as -6.751 instead of -6751 because of the specification F10.3 in the FORMAT statement and the absence of a decimal point. Since d in the general form Fw.d is 3, the decimal point will be placed 3 places to the right of the units digits.

B is defined as 3.45 instead of 34.5 since the presence of the decimal point in the number over-rides the decimal point placement indicated by the format specification. If the decimal point is punched, the placing of the number anywhere in the indicated data field (w in Fw.d) will give the same result. i.e., 3.45 in this case will actually be read in as +3.4500 which is the same value.

WRITE Statement. The purpose of the WRITE statement is to produce on an output medium, information stored in the memory. The general form corresponds to the general form of the READ statement.

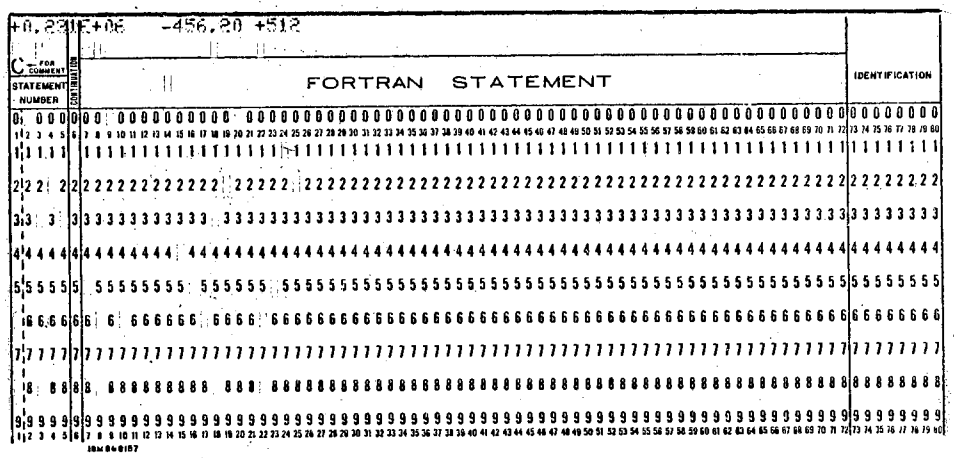
The general form is as follows:

WRITE (i,n) list

The i, n, and list have the same meaning as in the READ statement. The output unit used in this study will be either the card punch (unit number 7) or the on-line printer (unit number 6).

Suppose the memory cells named T, V, and I contain respectively -456.2, 23.1E+06, and 512. i.e., T is defined as -456.2, V is defined as 23.2E+06, and I is defined as 512. After the following WRITE statement was executed, the values would appear on a punched card as follows:

```
104 FORMAT (E10.3, F10.2, I5)
    WRITE (7,104) V, T, I
```



Special care must be made to provide a sufficiently wide output field in the FORMAT statement used for output. If the output field has fewer columns than the value in memory being punched or printed, a truncation may occur or the error may halt execution of the program. When integers are truncated, the leftmost digits are lost whereas the rightmost digits are lost when real numbers are truncated. If a field specification is larger than needed, no error will occur and blanks will appear in the unused columns on the left side of the quantity in the output field.

The exact number of columns for each of the data specifications, I, F, and E, needed on output will vary with the particular computer in use. However, in general the following rules should be followed:

Iw	w should allow for the number of digits in the integer plus one column for the sign.
Fw.d	$w \geq d + 3$. The 3 extra columns allows for the sign, leading zero (if number is less than one), and decimal point.

Ew.d	$w \geq d + 7$. The 7 extra columns allows for the sign, leading zero, decimal point, letter E, exponent sign, and a two digit exponent.

Below are several examples showing the output of the given number for a particular specification.

A = -32.756

Specification	Output
F10.2	␣␣␣␣-32.75
F7.3	-32.756
F6.3	error
F5.0	␣-32.
E10.3	-0.327E+02
E12.5	-0.32756E+02
E9.3	error

N = +127

Specification	Output
I4	+127
I5	␣+127
I3	error
I2	error

STOP and END Statement

It was discussed earlier that a program written in a language such as FORTRAN must be translated into the acceptable machine language for the particular computer before it can be executed. The program used to translate the program is called a compiler. Hence, the source program (FORTRAN program) must be read into memory by the compiler and translated into the object program. The compiler must know when it has

compiled all of the statements in the source program. The END statement (nonexecutable) in the source program serves this function. Therefore it must be the last instruction or statement in every FORTRAN source program. Once a FORTRAN source program is compiled into the object program, the object program is executed. The computer must have an instruction as to when execution is to be terminated. The STOP statement in the FORTRAN source program performs this function. It is possible that a FORTRAN source program will contain more than one STOP statement.

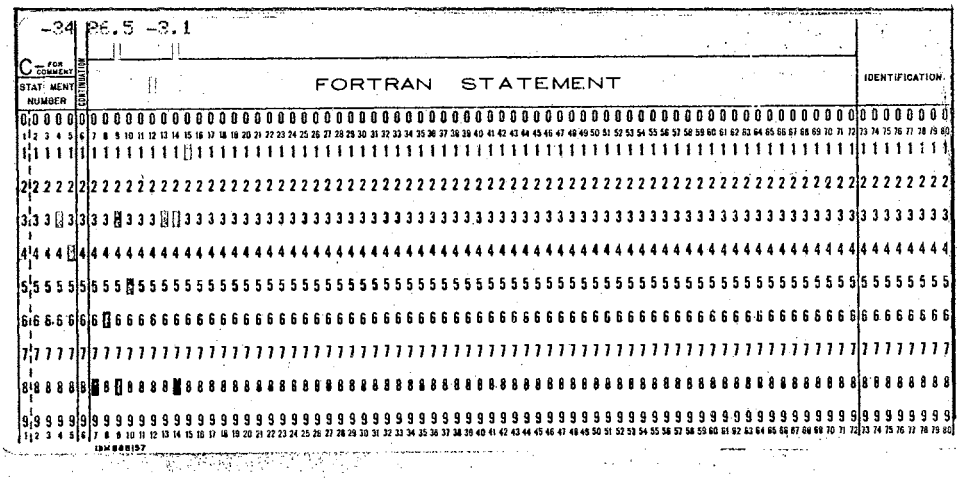
The general form of the STOP and END statements are as follows:

```
STOP
END
```

Note: It is not always true that a STOP statement immediately precedes the END statement, but generally it is the case. Exception to this will be demonstrated in later sections.

Sample Problem 1: Write a program in FORTRAN IV to perform the following tasks:

a. Read from the following data card values for N, A, and B.



b. Write out on the on-line printer the values for N, A, and B as follows:

A in columns 2-10 in F-specification retaining two digits to the right of the decimal point.

B in columns 11-20 in E-specification retaining three decimal digits to the right of the decimal point.

N in columns 21-25 in I-specification.

c. Terminate execution.

Program for Sample Problem 1

```
1   567
C      STUDENT NAME
C      SAMPLE PROBLEM 1
  100 FORMAT(I5, F10.0, F5.0)
  101 FORMAT(1X, F9.2, E10.3, I5)
      READ(5,100)N, A, B
      WRITE(6,101) A, B, N
      STOP
      END
```

The output would be as follows from the on-line printer:

```
                                                                  1
        1         2    2                                          3
1       0         0    5                                          2
    86.50    -3.000  -34
```

Notice the 1X in FORMAT statement having statement number 101. The X is another format specification which is for blank data. The general form is

wX

On input, w columns of the input record are ignored regardless of what they actually contain. On output, w blank characters are inserted in the output record. The 1X places a blank character in the first

column. On an on-line printer, the first column is generally reserved for carriage control. i.e., the character printed in column one determines whether printed lines are single-spaced, double-spaced, etc. A blank character is for single-spaced. For further discussion of carriage control, the FORTRAN Reference Manual should be checked.

Also, notice the letter C in column 1 of the first two lines or cards of the source program. These cards may then be used for written comments. These comments become a part of the source deck, but will not affect operation of the program. Whenever the program is printed, the comments will appear among the instruction statements just as they appear on the cards. Such cards are referred to as comment cards. These cards are convenient as "notes" or "reminders" to the programmer.

Each of the above lines written in the program will be placed on a tab card. Columns 1-5 are used for statement numbers, column 6 should be blank, with the FORTRAN statement beginning in column 7.

Exercise 3.

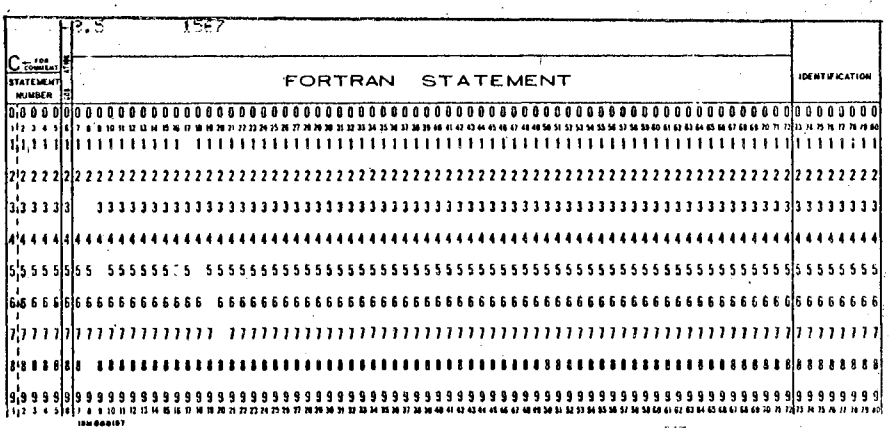
3.1 What are the three specification forms for numeric data?

3.2 The STOP statement is to stop _____.
The END statement is to stop _____.

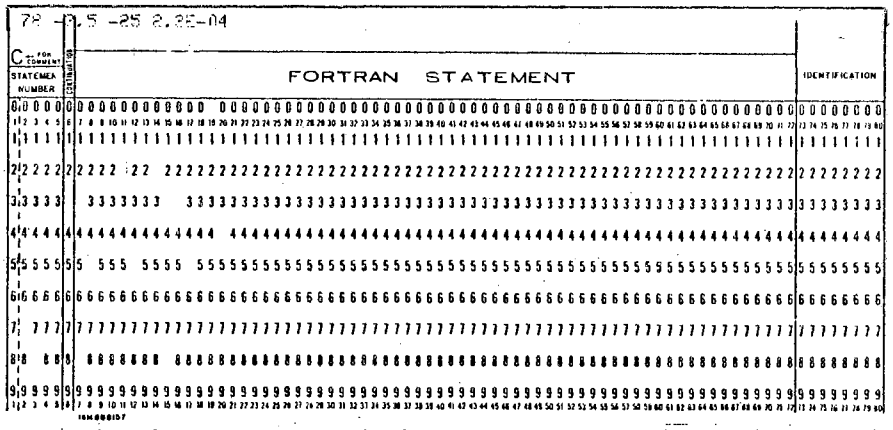
3.3 What is a "list"?

3.4 After the following program segment was executed with the data card as follows, what would the memory cells A and B contain?

```
100 FORMAT(F10.2, F10.3)
    READ(5,100) A, B
```

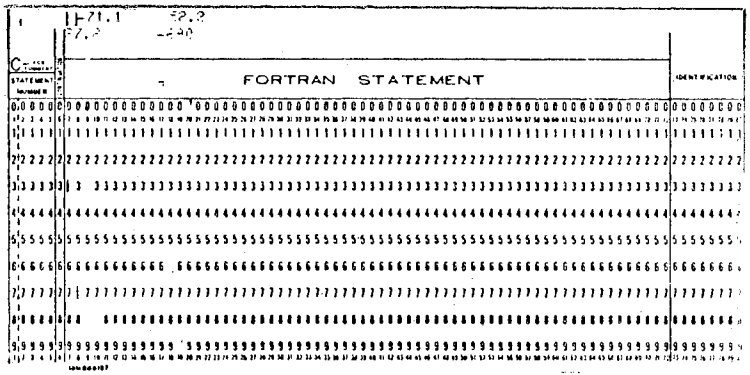


3.5 Write a FORMAT statement and a READ statement to read in the data fields of the following card. (There are 4 data fields)



3.6 Given the data cards and the statements below, what would be the value of the variables after the statements were executed?

```
1 FORMAT(F10.1, F10.1)
  READ(5,1) A, B
  READ(5,1) C, D
```

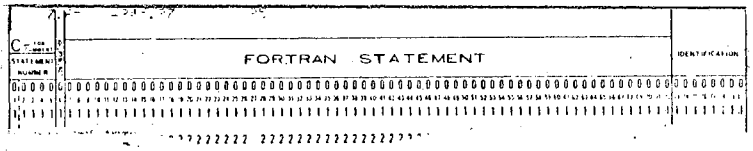


3.7 Suppose the data cards in Exercise 3.6 were reversed (the second card is first and the first card is second). What would the values of the variables be?

3.8 Suppose the two READ statements were reversed in Exercise 3.6. What would the values of the variables be?

3.9 Write a program in FORTRAN IV to perform the following tasks:

a. Read from the data card shown below values for A, B, and M.



b. Read from the second data card values for D, K, and E. D is in columns 11-20 with decimal point punched; K is right justified in columns 31-40; and E is in columns 56-60 with decimal point punched.

c. Write out the values of D and B (in that order) on one line in E-specification (allow for six digits).

d. Write out the values of E and A (in that order); E and A in F-specification with three decimal places on the second line.

e. Write out the values of K and M (in that order) on the third line.

f. Terminate execution.

Constants and Variables

In writing programs, there are some numerical quantities that will remain fixed throughout the program. Such a number is called a constant. Constants are written in the source program and are not input data. Examples of constants are numbers like:

3.1416 5 0.005 -3.E+04

This study will consider only two types of constants, integer and real. An integer constant is a decimal number (formed of digits 0 through 9 of the decimal system) with no decimal point. Numbers of this type are also referred to as fixed point numbers. i.e., the decimal point is always "fixed" at the right end of the number. The number of digits in the integer and its magnitude are limited and depend upon the particular computer in use. However, any integer having from one to five digits with magnitude not exceeding 99999 is generally always accepted. Examples of integer constants are:

-5 60 0 -9987

A real constant is a decimal number with a decimal point. Again, the number of digits and the magnitude are limited and depend on the particular computer in use. In general, any real number having from one to six significant digits with its magnitude not exeeeding the approximate limits 10^{-32} - 1 and 10^{32} - 1 is always acceptable. Examples of real constants are:

0.0 -3.1 +256. -15.0 .0056

There is another form, called the exponential form, of a real number. For further information on this form, refer to the FORTRAN Reference Manual for the particular computer in use.

Also a real constant is sometimes referred to as a floating point

number, because the decimal point is not restricted to any particular position.

A variable in programming has the same meaning as in normal algebraic usage. In the FORTRAN language, a variable (sometimes called a variable name) may be written from one to six letters. These are several different types of FORTRAN IV variables. This study will be restricted to two types of variables. An integer variable is any variable name having as the first character one of the following letters:

I, J, K, L, M, N

Any other variable name beginning with other letters of the alphabet is called a real variable.

For either an integer or real variable name having more than one letter, the other characters (up to a total of six) must be either numeric or alphabetic. Examples of acceptable integer and real variable names are:

Integers	Real
LABOR	HOURS
JOB	A
I	PAY
NUM	XI
L123	DELTAX

In choosing variable names, a programmer should attempt to select names that are meaningful to him. This will help one recall at a later date what the program is accomplishing at different segments of the program.

Arithmetic Expressions

Generally, most people using computers wish to perform arithmetic calculations. This arithmetic expression is the means by which the

programmer can instruct the computer to perform certain calculations and produce a single numerical value equivalent to the value of the expression.

The computer is capable of performing five basic operations. The operations and the corresponding operation symbols are given as follows:

Operation	Symbol
add	+
subtract	-
multiply	*
divide	/
exponentiation	**

A FORTRAN IV arithmetic expression consists of:

(1) Any single constant or single defined variables, or

(2) Any sequence of constants, defined variable names, and operation symbols given above.

Examples of arithmetic expressions are given as follows:

```
RATE
A*B + 3.2
-25
X - Y/W
A**2 + B/C
```

If an arithmetic expression contains more than one operation symbol such as

```
A**2 + B/C - D*E
```

the order of calculations will be according to the following hierarchy:

** exponentiation

- evoluation of unary minus quantities (example: -A*B)

*, / multiplication and division (from left to right)

+, - addition and subtraction (from left to right)

Hence, the arithmetic expression

```
A**2*B+C/D-E+3.1
```

would be calculated as the algebraic expression

$$a^2b + \frac{c}{d} - e + 3.1$$

The heirarchy of operations can be altered by using parentheses as in algebra.

If parentheses are used as follows:

```
A**2*(B+C)/(D-E) + 3.1
```

the expression would be equivalent to the algebraic expression

$$\frac{a^2(b + c)}{d - e} + 3.1$$

Perhaps a good rule to follow is that when in doubt how the expression will be evaluated, use parentheses.

Of course, there are times when the programmer must use parentheses. One case is when it is necessary to alter the heirarchy of the operations as mentioned above. Another case is when two operation symbols are required to be placed next to each other. An example is as follows:

A + - B must be written A + (-B)

or A / - B must be written A / (-B)

Another restriction the programmer must follow in writing arithmetic expressions is not to mix modes. That is, constants and the values assigned to the variable names in one arithmetic expression must all be integer values or all real values. Some compilers will accept mix mode but still requires more time to compile. Examples of mixed mode expressions are as follows:

Invalid	Valid
X + 3	X + 3.0
A*M - B	A *DM - B (DM replaces M)

3.0 * J 3 * J

There is one exception to using mixed mode in an arithmetic expression; namely in exponentiation. The following are the allowable combinations:

A ** B	(A and B real, A>0)
A ** N	(A real and N integer)
M ** N	(M and N integer)
M ** A	(M integer and A real. NOT ALLOWED)

Notice the second case, A ** N. This is the only allowable mixed mode expression for most FORTRAN compilers.

Integer and Real Arithmetic

Integer arithmetic is limited to operations performed on the integers. When integers are added, subtracted, or multiplied, the result will be integer (if not exceeding limit of magnitude on integers). When integers are divided, the result is the largest integer not exceeding the real value of the quotient. No fractional remainder is even developed. Hence, the following integer expression would have the indicated result:

Expression	Result
3/2	1
4/2	2
7/3	2
4/5	0

Real arithmetic is when real constants or variables are used in an arithmetic expression (exception, real number raised to an integer power). Generally all calculations are done using real arithmetic or

the real mode. Hence, the result will always be a decimal number. Remember the real values are also limited in magnitude and in the number of digits. As a real expression is evaluated, each partial result and the final result is developed to the maximum number of digits possible for the particular computer in use.

Examples of expresions in integer or real arithmetic are given as shown:

Expression	Mode
3*M - 4	Integer
A/B + 3.1 *C	Real
X ** 2 + Y	Real
K/2 + 1	Integer

Exercise 4.

4.1 What is a constant? In FORTRAN IV, give the name of the two different types. Give an example of each.

4.2 List the five arithmetic operations along with the corresponding operation symbol.

4.3 Give the allowable exponentiation forms in FORTRAN IV.

4.4 What is meant by a mixed mode expression? Give examples.

4.5 Identify the following as constants or variables and whether of integer or real mode.

a. 3.0
b. JAY
c. AREA
d. 5
e. X54
f. TAX
g. 728.
h. L

4.6 Identify the following as valid or invalid variable names and if valid indicate the type of variable (integer or real).

a. A1234
b. MONEY$
c. 2RATE
d. DISTANCE
e. A*B
f. JA

4.7 Consider the following arithmetic expressions and indicate whether they are valid or invalid. If invalid, rewrite the expression to make it correct.

a. RATE * TIME
b. 5A + 3B
c. .5HGHT * BASE
d. $AX^2 + BX + C$
e. B*(C/2)
f. (DELTA + 1) XBAR

4.8 Write FORTRAN IV real-mode arithmetic expressions equivalent to the following algebraic expressions.

a. $3x^2 + 2x - 5$

b. $a^2 + b^2$

c. $1/3\ h(a + b)$

d. $\frac{a - b}{c + d}$

e. $\frac{a}{n - 2}(3n^2 - 4)$

f. $5n(3x - y)^3$

Arithmetic Assignment Statement

The purpose of the arithmetic assignment statement is to assign a value to a variable name.

The general form of the arithmetic statement is:

a = e

where a is a variable name and e is an arithmetic expression. This statement causes the computer to take numerical values from defined memory cells, perform arithmetic calculations using those values, and transmit the resulting value to a memory cell.

When an arithmetic statement is written they look like ordinary equations in algebra. For an example consider

A = B + C

Although this looks like an equation in mathematics, it has a slightly different meaning. In mathematics, this equality would mean, A has the exact same value as B + C (if working with numeric values).

In FORTRAN it means, "computer, take the value for B + C and place this value in memory cell named A." Thus, the variable has been assigned some numeric value. There are two ways a variable can be defined in FORTRAN: (1) appear in the list of a READ statement (2) appear on the left side of an arithmetic statement.

Some more examples of arithmetic assignment statements are:

```
A = 3.0 * B - C
AREA = 3.14 * R ** 2
PAY = TIME * WAGE
X = 5.0
X = X + 1.0
```

Note the last arithmetic statement in the above examples

```
X = X + 1.0
```

In algebra this would not be possible as $X \neq X + 1.0$ for any X. However, the equal sign in FORTRAN means, take the value in memory cell labeled X, add one, and store the result back in the memory cell labeled X. The equal sign could be read as "replace by." Also, the only quantity allowed on the left side of any equal sign in an arithmetic statement is a single variable name. The following example would be an invalid arithmetic statement:

$$X + Y = W$$

Although, mathematically, $X + Y = W$ and $W = X + Y$ are identical, in FORTRAN the two expressions are entirely different.

It has been mentioned that mixed mode expressions will not be accepted by some compilers. Yet there are times when an integer value needs to be expressed as a real and vise versa. This can be accomplished as follows:

$$A = N$$

This arithmetic statement will take the numerical value of N and store

this value in the memory cell labeled A. Since A is a real variable name, any number stored in this memory cell will have a decimal point.

Likewise the following statement will cause the real number having the variable name B to be stored in memory cell K as an integer.

```
K = B
```

The above two examples should not be confused with an expression such as the following:

```
X = 3 * Y + J
```

Since Y is a real variable name, 3 an integer constant and J an integer variable name, the arithmetic expression on the right side of the equal sign has mixed mode. However, the following example would be a valid arithmetic statement:

```
A = N + 2*M
```

Since the arithmetic expression on the right side of the equal sign is all of the same mode, an integer value will be calculated for N + 2*M and then stored in memory cell labeled A as a real number.

Sample Program 2: An object dropped from a height travels the distance

$$d = 1/2\ at^2$$

where $a = 32.16\ ft/sec^2$ (gravitational constant) and t = time in seconds. Write a program in FORTRAN IV to perform the following tasks:

a. Read in a real value for t in columns 1-5 with decimal point punched.

b. Calculate the distance traveled for the value of t read in step 1.

c. Write out the value for a and t on one line retaining two decimal places in field widths of 10 columns using F-specification.

d. Write out the calculated value of d on a second line retaining 6 significant digits using the E-specification.

e. Terminate execution.

Program for Sample Problem 2

```
1   567
C     NAME
C     DISTANCE TRAVELED BY FALLEN OBJECT
  100 FORMAT(F5.0)
  101 FORMAT(1X, F10.2, F10.2)
  102 FORMAT(1X, E13.6)
      A = 32.16
      READ(5,100) T
      D = (.5 * A) * T ** 2
      WRITE(6,101) A, T
      WRITE(6, 102) D
      STOP
      END
```

Exercise 5.

5.1 Consider the following arithmetic assignment statements. Circle those that are valid. If invalid, write a correct statement.

a.	A = B + 3	e.	AMT = PRIN + INT
b.	X = I + 2	f.	T = U/-V
c.	C + D = E	g.	HOURS = 30
d.	W = Y ** 2 - 5.1 * X	h.	AREA = 1/2 BH

5.2 The area of a trapezoid is given as

$$A = 1/2\ h(b_1 + b_2)$$

where h is the height; b_1 and b_2 are the bases.
Write a program in FORTRAN IV that will do the following tasks:

a. Read in real values for h, b_1, and b_2 from a data card with each in field widths of 10 columns with the decimal point punched.

b. Calculate the area of the trapezoid with hieght h, bases b_1 and b_2.

c. Write out on one line the values for h, b_1, and b_2 retaining 3 decimal places in field widths of 8 columns using F-specification.

d. Write out on a second line the calculated value for the area retaining 2 decimal places using F-specification.

e. Terminate execution.

5.3 Given the quadratic equation

$$ax^2 + bx + c = 0$$

the two real roots of the equation may be found by using the quadratic formula

$$x = \frac{-c \pm \sqrt{b^2 - 4ac}}{2a}$$

if they exist. Write a program in FORTRAN IV to perform the following tasks:

a. Read in real values for a, b, and c in field widths of 10 columns each with the decimal point punched.

b. Calculate the two real roots (assume they exist) of the equation by using the quadratic formula. Name the roots ROOT1 and ROOT2.

c. Write out the values for a, b, and c on one line in field widths of 10 columns each retaining 2 decimal places.

d. Write out the calculated values of ROOT1 and ROOT2 on a second line retaining 6 significant digits using E-specification.

e. Terminate execution.

Transfer Control Statements

Unconditional GO TO. Ordinarily, the executable statements in a source program are executed in the order they appear in the source program. That is, control passes from one executable statement to the next in the order in which the statements are arranged. Sometimes, it is desirable to go back to a preceding statement or to skip ahead to a succeeding statement. There are several statements, called transfer control statements, which will allow for this deviation from the normal order of execution. This study will discuss four such statements.

The first of these will be the unconditional GO TO statement. The general form of this statement is

GO TO n

where n is a statement number of some executable statement. The execution of this statement will cause program control to the statement numbered n. For example

```
GO TO 3
```

will transfer to the executable statement having statement number 3.

To illustrate, consider Sample Program 2 in the preceding section. Reading in one value of t and calculating the distance d traveled will certainly not save the programmer time when the time is considered for writing the program to find the desired calculation. It would have been quicker and easier to find the distance by hand calculation. But, suppose it was desirous to determine the distance traveled for 100 different values for the time t. Certainly it should require the same instructions to determine the distance traveled for any two given times. Hence, the problem is then to write the program so as to transfer back to read in another time t after the required tasks have been performed for the initial time t.

Thus, the following program should find the distance traveled for any time t read in by the READ statement.

```
1   567
C     NAME
C     DISTANCE TRAVELED BY FALLEN OBJECT FOR ANY TIME T
  100 FORMAT(F5.0)
  101 FORMAT(1X, F10.2, F10.2)
  102 FORMAT(1X, E13.6)
      A = 32.16
    5 READ(5,100) T
      D = (.5 * A) *T ** 2
      WRITE(6,101) A, T
      WRITE(6,102) D
      GO TO 5
      END
```

Notice the two differences in the above program from the program written in Sample Program 2. (1) the statement number 5 of the READ

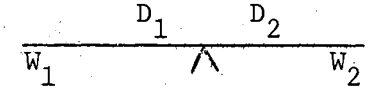
statement (2) replacing the STOP statement with the GO TO 5 statement. The above program will read a value for t, calculate the distance and print out the desired information. Then program control will be transferred back to the READ statement by the GO TO 5 statement. The program will then read another time t and repeat the desired tasks. This program will set the computer in what is called a loop. The program will continue to loop until no data cards are left. When all data cards are read this will cause execution to be terminated. Hence, the STOP statement is not needed. However, it is generally a good idea to always terminate execution by the STOP statement. Termination by the absence of more data cards will give an error message by most compilers. How to alleviate this problem will be discussed in following sections.

Exercise 6.

6.1 Modify the program for exercise 5.3 so that after step d the program will read in another set of data and repeat steps a to d. This should be repeated until all data cards are read and the values of the roots are determined.

6.2 If W_1 and W_2 represent two weights to be placed on a bar, with D_1 and D_2 their respective distances from the pivot point, a basic principle in physics for the bar to balance requires

$$W_1D_1 + W_2D_2 = 0$$



Write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card values for W_1, W_2, and D_2 (non-zero).

b. Calculate the value D_1 the weight W_1 should be placed from the pivot point in order for the bar to balance.

c. Write out the values D_1, W_1, D_2, and W_2 in this order on one line.

d. Program control should transfer to read in another set of data, etc., until all data cards have been processed.

Computed GO TO. The unconditional GO TO statement discussed in the previous section will cause transfer to any executable statement. However, it is not possible to alter this transfer depending on certain conditions within the computer. The computed GO TO statement is a transfer control statement which makes it possible to transfer control forward or backward in a program. The difference in function is that the computed GO TO will at the time of execution of the statement transfer according to certain conditions within the program. i.e., a choice is made concerning the transfer of control at the time of execution of the statement.

The general form of the computed GO TO statement is

$$\text{GO TO } (n_1, n_2, n_3, \ldots, n_m), i$$

where $n_1, n_2, n_3, \ldots, n_m$ are statement numbers of executable statements and the index i is a defined integer variable having a value between 1 and m. ($1 \leq i \leq m$, i an integer) This index value refers to a position in the list of statement numbers counted from left to right. Transfer of control is then to the executable statement having the statement number in the list that the index value refers to.

For an example, consider

$$\text{GO TO } (4,2,10), J$$

If J = 1, control is transferred to statement numbered 4.

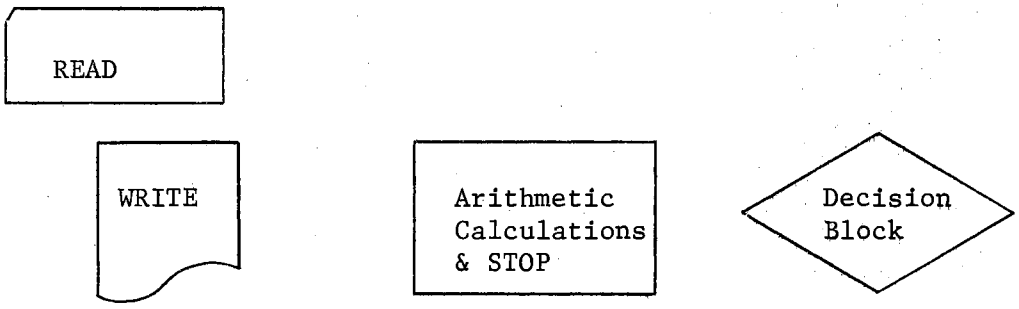
If J = 2, control is transferred to statement numbered 2.

If J = 3, control is transferred to statement numbered 10.

Remember, J must be such that $1 \leq J \leq 3$ in this case. Also, J must be a defined variable name prior to execution of the conditional GO TO statement.

Up to this point, it was very easy to "picture" how a computer

would perform a program. One could easily determine the order in which the instruction statements would be performed by merely writing down the problem in steps. The unconditional and conditional or computed GO TO statements changes this. As transfer of control statements are used and as more complicated problems arise, the programmer needs some way to order the operations of a program. The use of what is called a "flow chart" assists a programmer to determine what order to do things. Below are three conventional symbols and their respective operations used in flow charts:



The flow of operations from one instruction to the next is shown by an arrow drawn from one symbol to the next. A flow chart should not show all small details but merely the logical flow of the program to show the steps of operations.

Sample Problem 3. Draw a flow chart and write a program in FORTRAN IV to perform the following:

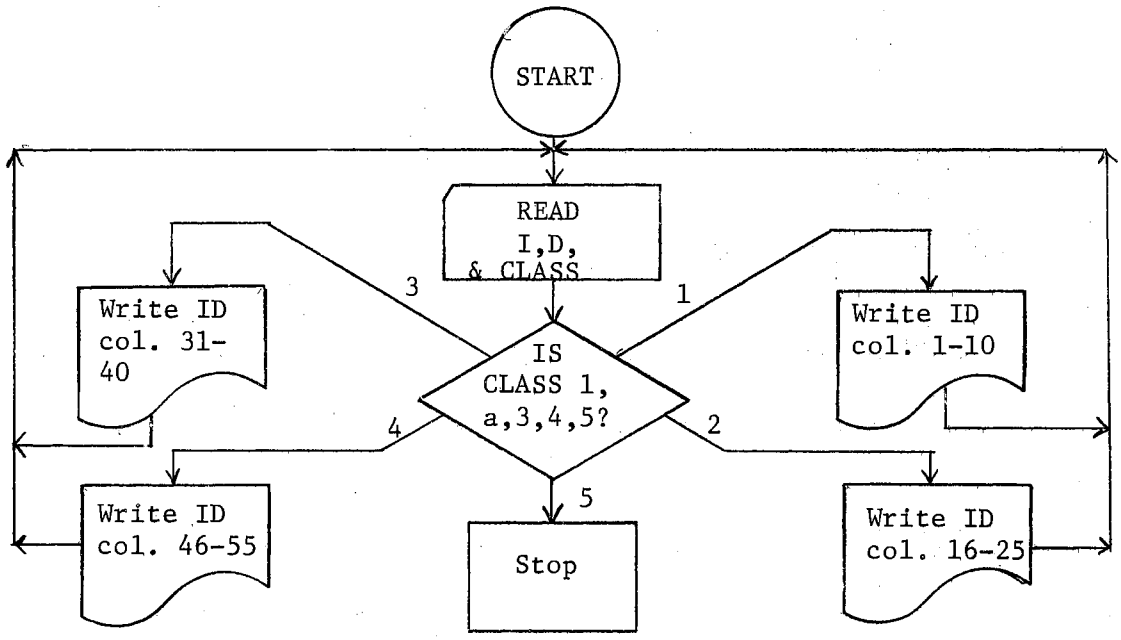
a. Read from a data card a student's I. D. number in columns 1-8 right justified in the field and a class code in column 10 (1 for freshman, 2 for sophomore, 3 for junior, 4 for senior).

b. Write out the students I. D. number in certain columns depending on his classification as follows:

Freshmen	Columns 1-10
Sophomore	Columns 16-25
Junior	Columns 31-40
Senior	Columns 46-55

c. Repeat the program execution until a class code of 5 is read. When the class code is 5, terminate execution.

Flow Chart for Sample Problem 3



Program for Sample Problem 3

```
1   567
C      NAME
C      LIST BY CLASSES
  100 FORMAT (I10,1X, I1)
  101 FORMAT(1X, I9)
  102 FORMAT(15X, I10)
  103 FORMAT(30X, I10)
  104 FORMAT(45X, I10)
    3 READ(5,100) ID, KLASS
      GO TO (10,20,30,40,50),KLASS
   10 WRITE(6,101) ID
      GO TO 3
   20 WRITE(6,102) ID
      GO TO 3
   30 WRITE(6,103) ID
      GO TO 3
   40 WRITE(6,104) ID
      GO TO 3
   50 STOP
      END
```

Exercise 7.

7.1 Consider the following computed GO TO statement

```
GO TO (3,5,1,2), I
```

a. If I = 1, the above statement will cause control to be transferred to statement numbered_____?

b. In the above transfer statement, the range of the variable I is _____?

7.2 Draw a flow chart and write a program in FORTRAN IV that will perform the following tasks:

a. Read from a data card values for ITEST and X. The card format for the variables to be read will be as follows:

ITEST(integer)	columns 1-5 (right justified)
X(real)	columns 11-20 (decimal point punched)

b. If ITEST = 1, calculate $y = 3x^2 - 2x + 5$.
If ITEST = 2, calculate $y = \frac{x^2 - 3.}{x^2 + 4}$

If ITEST = 3, calculate $y = x - \frac{3}{x + 2}$

If ITEST = 4, halt execution.

c. After each of the above calculations, write out the value of ITEST, X, and Y using appropriate field widths retaining 2 decimal places for the real numbers.

Note: The program should read data cards until a value of ITEST is read as 4.

7.3 Draw a flow chart and write a program in FORTRAN IV to perfrom the following tasks:

a. Read from a data card an integer value for n in columns 1-3 right justified and a real value for the radius r of a circle in columns 10-15 with decimal point punched.

b. If n is 1, calculate the area of a circle with radius r.
If n is 2, calculate the circumference of a circle with radius r.
If n is 3, calculate the volume of a sphere having radius r.
If n is 4, calculate the surface area of a sphere with radius r.
If n is 5, halt execution.

c. For each data card read, write out the values of n and r. On a second line write out the requested calculation in E-specification.

Arithmetic IF Statement

The purpose of the arithmetic IF statement is to make a decision. The decision is to determine what statement to execute next. Control is transferred to one of three statements depending upon the minus, zero, or positive result of an arithmetic expression. The general form of the arithmetic IF statement is:

$$IF(a)\ s_1, s_2, s_3$$

The a is called the argument and is any valid arithmetic expression. The s_1, s_2, s_3 are statement numbers of executable statements appearing in the source program such that:

s_1 is the number of the statement to be performed next if the value of a is negative.

s_2 is the number of the statement to be performed next if the value of a is zero.

s_3 is the number of the statement to be performed next if the value of a is positive.

To illustrate, consider the following:

```
IF (A - B) 5, 10, 12
```

If A - B $<$ 0, the next statement executed will be statement numbered 5.

If A - B = 0, the next statement executed will be statement numbered 10.

If A - B $>$ 0, the next statement executed will be statement numbered 12.

The argument of the arithmetic IF statement may be of either mode. That is, a may be an integer or a real arithmetic expression.

Sample Problem 4: Given a triangle with the length of the sides as a, b, and c, the area of the triangle is given by Hero's formula

$$\text{area} = \sqrt{(s - a)(s - b)(s - c)}$$

where

$$s = 1/2\ (a + b + c).$$

Write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card as follows:

   a in columns 1-10
   b in columns 11-20
   c in columns 21-30
   n in column 35

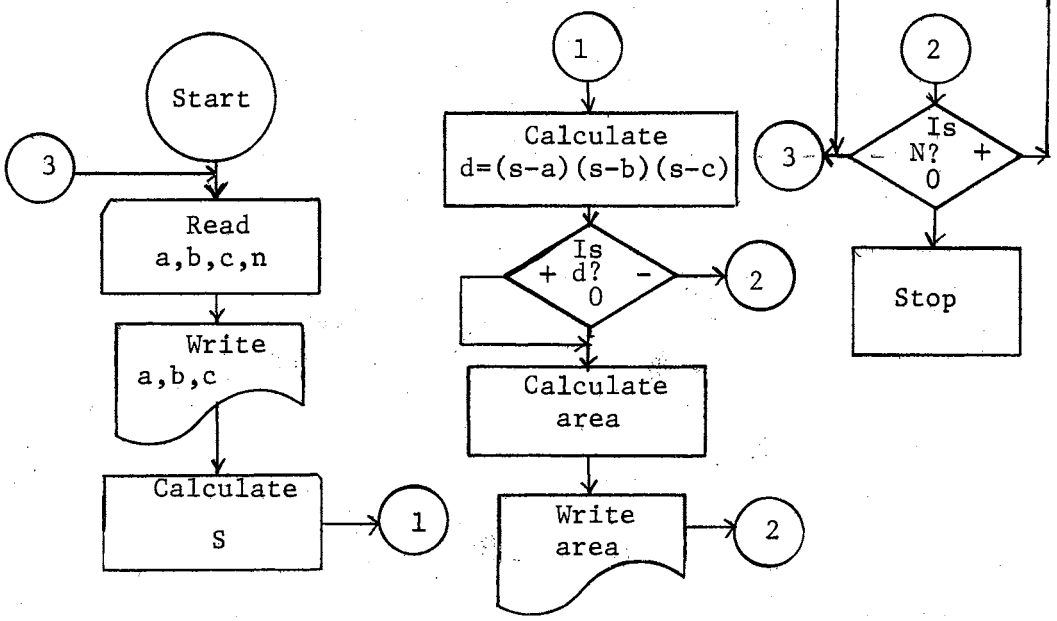
   a, b, and c are real numbers with the decimal point punched. n is an integer having a value of 1 or 0.

b. Calculate the area of the triangle by using Hero's formula. Note: If $(s - a)(s - b)(s - c) \leq 0$, it is not possible for a triangle to have sides having lengths of a, b, and c. The program should check for this condition.

c. Write out the values of a, b, and c on one line retaining 2 decimal places. Write out the calculated value for the area on a second line retaining 4 decimal places. If no triangle exist for a, b, and c, write out the integer value 9999 for the area.

d. If n has a value of 1, go back to read in more data. If n has a value of 0, halt execution.

Flow Chart For Sample Problem 4



Program for Sample Problem 4

```
1   567
C     NAME
C     HERO'S FORMULA FOR AREA OF TRIANGLE
  100 FORMAT(3F10.0, 4X, I1)
  101 FORMAT(1X, 3F10.2)
  102 FORMAT(1X, I5)
  103 FORMAT(1X, F12.4)
      I = 9999
    1 READ(5,100) A, B, C, N
      WRITE(6,101) A, B, C
      S = .5 * (A + B + C)
      D = (S - A)*(S - B)*(S - C)
      IF(D) 2,2,3
    2 WRITE(6,102) I
      GO TO 4
    3 AREA = D ** .5
      WRITE(6,103) AREA
    4 IF(N) 5,6,5
    5 GO TO 1
    6 STOP
      END
```

Note FORMAT statements numbered 100 and 101. The F-specification as written 3F10.0 or 3F10.2 are called repeated specifications. Written as 3F10.0 is the same as three different fields, each F10.0. Thus, FORMAT (3F10.0) is the same as FORMAT(F10.0, F10.0, F10.0). Also, note the IF statement numbered 4. Although N will not be negative in the problem the programmer must have a statement number of an executable statement for this case. Hence, the statement number 5 is used in this statement for both the negative and positive case.

Prior to the IF statement, when the program was put in a "loop" using the unconditional GO TO statement, the only way the program would stop executing was by exhausting the data cards. By the use of the IF statement, normal exit by the STOP statement can be accomplished. The use of the STOP statement should be used to terminate execution at all times if possible.

For some compilers there is another IF statement called the

logical IF. For further information refer to the documentation on the particular compiler in use.

Exercise 8.

8.1 How many arrows must be drawn from a decision block in a flow chart that represents an arithmetic IF statement?

8.2 How are these arrows labeled?

8.3 Given a quadratic equation $ax^2 + bx + c = 0$, the real roots, if they exist, can be found by using the quadratic formula

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Draw a flow chart and write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card, a value for M which gives the number of data cards to follow.

b. Read from each of the following data cards values for a, b, and c. (assume $a \neq 0$)

c. Find the real roots of the quadratic equation and write out the calculated roots in E-specification. If there are no real roots, the program should write out the integer 19191.

d. After all data cards have been processed, halt execution.

8.4 Given any two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ in a coordinate plane, the equation determined by the two points is given by

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1), \text{ if } x_2 - x_1 \neq 0)$$

or solving for y we obtain the form

$$y = \frac{y_2 - y_1}{x_2 - x_1} x - \frac{y_2 - y_1}{x_2 - x_1} + y_2$$

or $y = mx + b$ where $m = \frac{y_2 - y_1}{x_2 - x_1}$ and $b = \frac{y_2(x_2 - x_1) - x_1(y_2 - y_1)}{x_2 - x_1}$

Draw a flow chart and write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card values for x_1, y_1, x_2, y_2 and NDATA as follows:

NDATA an integer in column 1 (either 1 or 0)
x_1 real in columns 5-9, x_2 real in columns 15-19
y_1 real in columns 10-14, y_2 real in columns 20-24

b. If $x_2 - x_1 \neq 0$, determine the value of m and b as given above and write out these values retaining three decimal places as follows:

m real in columns 1-15
b real in columns 21-35

c. If $x_2 - x_1 = 0$, write out the value of x_1 and x_2 retaining three decimal places.

d. If NDATA = 1, go back to read another data card and repeat steps b and c.
If NDATA = 0, terminate execution.

Labeled Output

It is obvious after observing the output on the programs thus far that it is sometimes confusing what the output represents. What is needed is some method to write messages, headings, or labels associated with the output. The H-specification or Hollerith code can be used for this purpose. The general form of this specification is

wH

where w is the number of columns required for the messages, headings, or label to be printed. To illustrate consider the following statements:

```
100 FORMAT(1X, 18HVALUE OF THE SLOPE)
    WRITE(6,100)
```

The above two statements will cause the following message to be printed beginning at the left margin of the output paper.

ßVALUE OF THE SLOPE

Notice there is no list associated with the WRITE statement. Also notice that the value for w in wH is 18 since allowance must be made for the blank characters as well as for any alphameric characters

Any valid character for the particular computer in use can appear in a Hollerith data field.

To show another form how the H-specification can be used, consider the following:

```
101 FORMAT(1X, 18HVALUE OF THE SLOPE, F8.2)
    WRITE(6,101) TM
```

Assuming -6.34215 is stored for the variable TM, the above two statements will cause the following message to be printed beginning at the left margin of the output paper.

bVALUE OF THE SLOPEbbb-6.34

Notice in this case, the WRITE statement has a list consisting of one variable name and the corresponding FORMAT statement has a F-specification immediately folloiwng the Hollerith data. Since this is the first numeric data specification in the FORMAT statement it is used for the first variable name in the corresponding READ statement.

There are other uses of the H-specification and for some compilers other means of writing out messages, headings, or labels. For further information check the documentation for the particular compiler in use.

Exercises 9.

9.1 Define the READ and FORMAT statements to do the following:

```
Print the word ALPHA in columns 10 through 14.
Print the word BETA in columns 20 through 23.
Print the word GAMMA in columns 30 through 34.
```

9.2 Assume the variable names A, B, and C are defined in a source program. Define the WRITE and FORMAT statements to print the following:

A=xxxx.xxbbbbB=xxxx.xxbbbbC=xxxx.xx

DO Statement

By the use of the GO TO statements and the arithmetic IF Statement it is possible to cause the computer to be placed in a loop. That is, perform over and over again a sequence of statements. Practically every program requires at least one loop to accomplish some task. Although, the use of the above mentioned transfer statements can be used for looping or iteration, there is a statement in FORTRAN that is primarily for this purpose. This statement is called the DO statement.

The purpose of the DO statement is to provide, combined in one statement, four operations necessary for looping or iterating a sequence of statements a specific number of times; namely:

(1) designate the sequence of statements which is to be iterated,

(2) define a variable initially to some specific value,

(3) increment that value by a given amount and

(4) test that value to determine if the required number of iterations has been performed.

The general form of the DO statement is as follows:

$$\text{DO m i} = n_1, n_2, n_3$$

The terms used to describe the parameters in the DO statement are:

m statement number of last statement in range

i index

n_1 initial value of index

n_2 test value

n_3 increment

m is a statement number; i must be an integer variable not yet defined; n_1, n_2, and n_3 are integer constants or defined integer variables.

DO m designates the sequence of statements that is to be iterated, that is, execute all statements that follow the DO statement down to and including the statement numbered m. This set of statements is called the range of the DO statement.

The second part i = n_1, n_2, n_3 of the DO statement gives the number of times the statements in the range are to be iterated. First the index i is set equal to the initial value n_1. The sequence of statements in the range are executed. Then control is transferred back to the DO statement. The value of the index is incremented by the value of the increment, n_3. This new value of the index is then tested to determine if it exceeds the test value n_2. If not, then the sequence of statements in the range are executed again with the new value of the index n_1. This operation is repeated until the new value of the index exceeds the test value n_2, at which time, transfer of control is to the first executable statement after the last statement in the range of the DO statement.

To illustrate, consider the following:

```
                100 FORMAT(F10.0)
                101 FORMAT(1X,2F10.2)
               ┌──→ DO 4 I = 1, 10, 1
loop           │    READ(5,100) X              ┐  range of
 10            │    Y = 3.0 *X**2 + X - 5.0    ┘  the DO loop
times          └─── 4 WRITE(6,101) X, Y
                    STOP
                    END
```

The above program will cause the DO loop to be performed 10 times. Each time the loop is executed, a data card is read with the value of X; Y is calculated for that particular value of X; and X and Y are written out on the printer. Then control is transferred back to the DO statement, the index I is incremented by 1 and the sequence is performed again. After the 10th time, the index I is incremented by 1

giving a value of 11. Since this exceeds the test value of 10, transfer of control is to the first executable statement following the statement numbered 4 which is the STOP statement. Of course, this will cause execution to terminate.

Consider another example:

```
100 FORMAT(I5)
    READ(5,100) L
    LSUM = 0
    DO 10 N = 1, L
 10 LSUM = LSUM + N
```

The above sequence of statements will find the sum of all the integers from 1 to L inclusive. In this example, notice the absence of the increment. When the increment is omitted, the value for the increment will always be taken as 1. Also, note in this example that the index is actually used in the range of the DO statement. The index will always be a defined variable in the range of the DO statement but will be undefined once control has left the range by the normal exit of a DO statement. That is, when the value of the index exceeds the test value.

There are several rules that must be observed when using the DO statement. They are:

(1) The index of a DO statement is initialized only when control is transferred to the DO statement from a statement other than the last in the range of the DO.

(2) The index of a DO statement is incremented only when control returns to a DO statement from the last statement in its range.

(3) A DO loop <u>is always</u> executed once with the index equal to the initial value.

```
Example:      DO 6 I = 5, 3
                    .
                    .
                    .
              6     .....
```

This loop will execute once with I = 5, then transfer control.

(4) The last statement in the range of a DO statement must be executable and not one of the following:

Any GO TO
Arithmetic IF
DO
STOP

(5) No statement in the range of the DO may change the value of i, n_1, n_2, or n_3.

(6) Transfer of control from outside of the range of the DO to a statement within the range is not permitted.

Example:

```
          .
          .
          .
      DO 8 M = K, L, N
          .
          .
          .
      IF(ITEST) 10, 8, 10
          .
          .
          .
8     ........
          .
          .
          .
10 J = M
          .
          .
          .
```

If transfer from the loop is done from the IF statement, the index M is defined and will have its current value at the time of transfer.

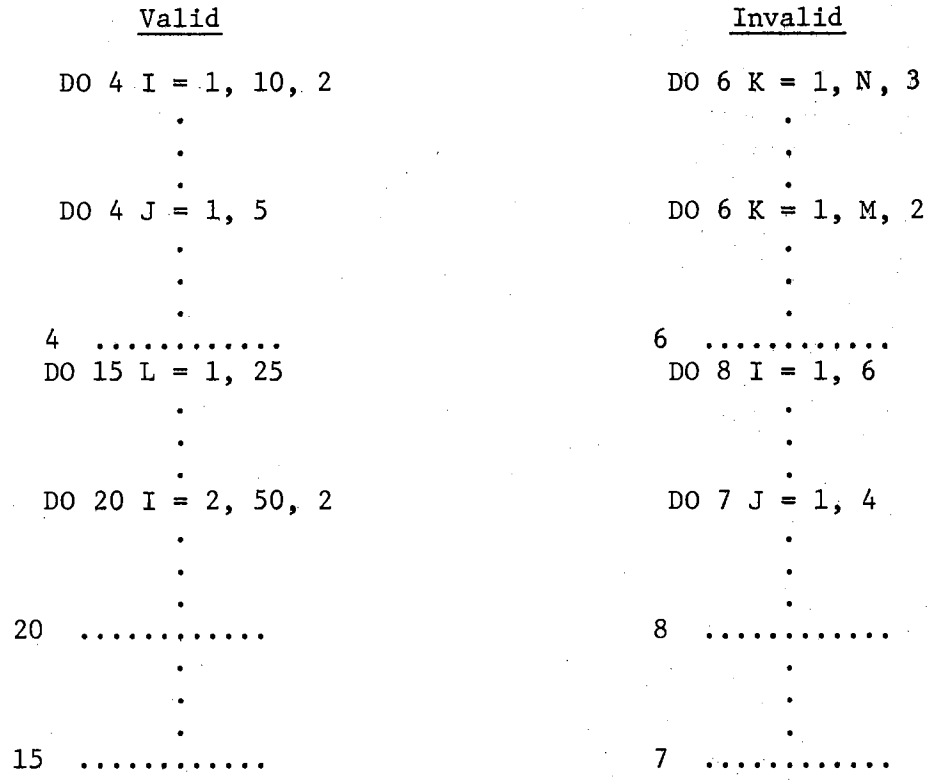
(7) If control is transferred out of a DO loop normally, that is, when the value of the index exceeds the test value, the index variable will be undefined. However, it may be used as any other variable name following the DO loop.

Nested DO Loops

A DO loop may have within its range another DO loop provided the inner loop is completely within the range of the outer loop. Such loops are called nested DO loops. It is permissible for the last

statement of the range of nested DO loops to be the same. However, the index of each DO statement in a nest must have a different variable name so as not to violate rule 5 above.

To illustrate, consider the following:



CONTINUE Statement

Sometimes it happens that for the last statement in the range of a DO must be one of the statements given in rule 4 above. In order for a programmer to comply to the rule, a statement in FORTRAN is provided that will merely pass control to the next statement to be executed. This statement is called the CONTINUE statement. It can be used as the last statement in the range of a DO loop which will refer control back to the DO statement and cause the index to be incremented and compared with the test value.

The general form is

```
n CONTINUE
```

where n is a statement number.

To illustrate, consider the following:

```
          .
          .
          .
     DO 10 I = 1, 15
          .
          .
          .
     GO TO (10, 11), N
10 CONTINUE
11   .........
          .
          .
```

Since the last desired statement in the range of the DO loops is GO TO (10, 11), N which is not an allowable statement by rule 4, the CONTINUE statement is added as the last statement.

Another use of the CONTINUE statement with a DO loop is when, under certain conditions, all the statements in the range should not be executed each time the loop is performed.

To illustrate, consider the following:

Valid

```
          .
          .
          .
    DO 5 I = 1, N
          .
          .
          .
    IF (A - B) 3, 5, 3
          .
          .
          .
3   ........
    X = X + 1.0
5 CONTINUE
          .
          .
          .
```

Invalid

```
          .
          .
          .
4 DO 5 I = 1, n
          .
          .
          .
    IF (A - B), 3, 4, 3
          .
          .
          .
3   ...........
5 X = X + 1.0
          .
          .
          .
```

Sample Problem 5: The arithmetic average or mean of a set of n numbers is given by

$$\text{arithmetic mean} = \frac{\text{sum}}{n}$$

Draw a flow chart and write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card an integer value of n in columns 1-5 right justified. n will be the number of data cards to follow with a real value in column 1-10 with the decimal point punched.

b. If n = 0, write out the message, "N IS ZERO".

c. If n ≠ 0, calculate the sum of all the n real numbers. Then calculate the arithmetic mean of the set of real numbers.

d. Write out and label appropriately, the calculated values of the sum and the arithmetic mean.

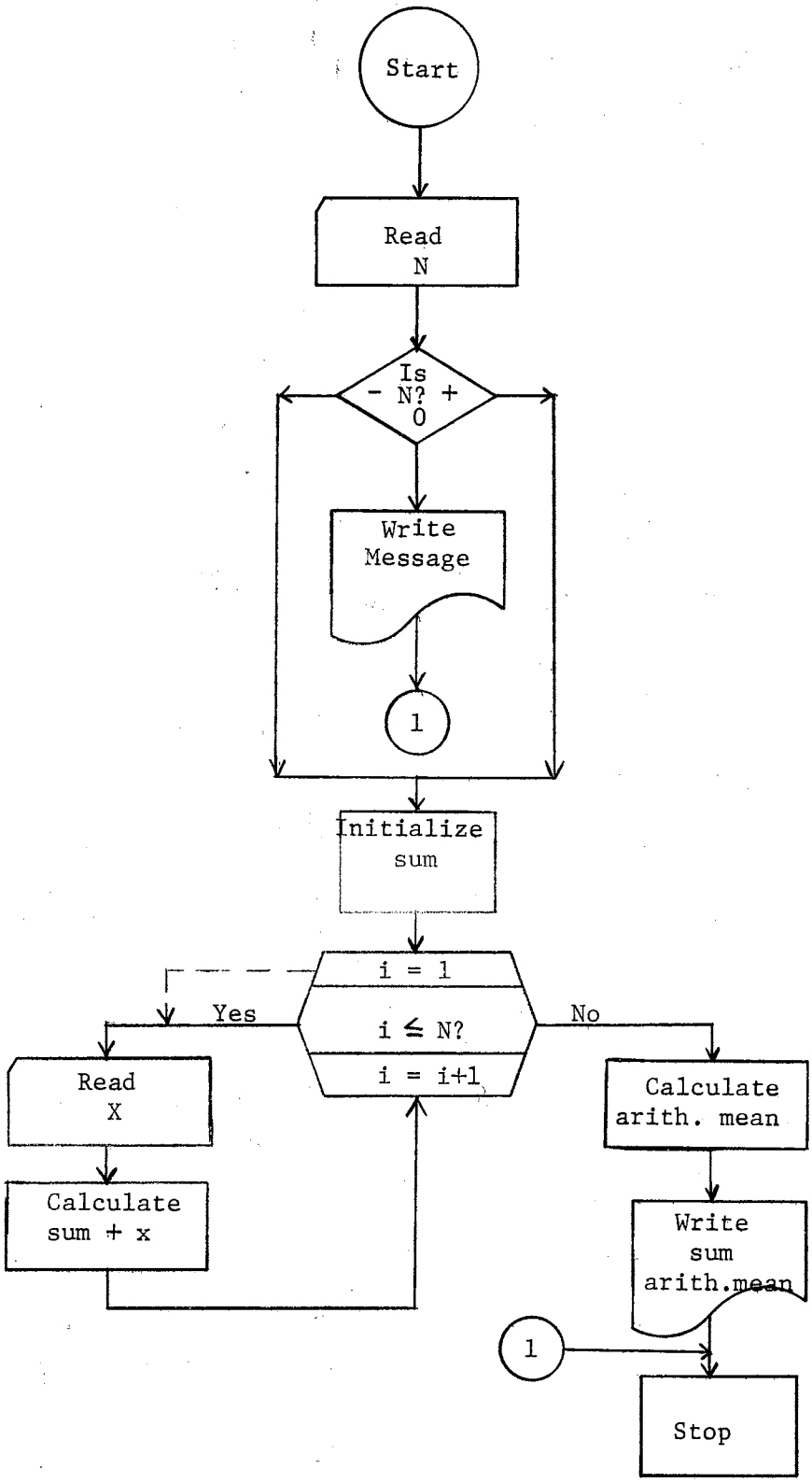
e. Halt execution.

Program for Sample Problem 5

```
1   567
C     NAME
C     ARITHMETIC MEAN OF N REAL NUMBERS
  100 FORMAT (I5)
  101 FORMAT(F10.0)
  102 FORMAT(1X, 9HN IS ZERO)
  103 FORMAT(1X, 6HSUM IS, F10.3, 7HMEAN IS, F10.3)
      READ(5,100) N
      IF(N) 1, 2, 1
    2 WRITE(6, 102)
      GO TO 3
      SUM = 0.0
      DO 4 I = 1, N
      READ(5, 101) X
    4 SUM = SUM + X
      REALN = N
      AMEAN = SUM/REALN
      WRITE(6, 103) SUM, AMEAN
    3 STOP
      END
```

Particular notice should be made of the technique used in finding the sum of the set of numbers by the use of a DO loop. In order to find an accumulative sum using the arithmetic statement, SUM = SUM + X, it

Flow Chart For Sample Problem 5



is necessary to initiate the value of SUM to zero. If this was not done, SUM would be an undefined variable in the arithmetic statement.

Exercises 10.

10.1 What general function does the DO statement perform?

10.2 Describe the effect of the CONTINUE statement.

10.3 Given any two sides of a right triangle, it is possible to determine the third side by use of the Pythagorean Theorem

$$c^2 = a^2 + b^2$$

where c is the hypothenuse or side opposite the right angle and a and b are sides or legs of the triangle.

Draw a flow chart and write a program in FORTRAN IV to perform the following tasks:

a. Read an integer value from columns 1-5 in the first data card which is the number of data cards to follow.

b. On each of the following data cards read real values in the first two data fields of 10 columns each. In column 25 read an integer value of either 0 or 1.

c. If the integer value read in step 2 is 0, find the hypotenuse of a right triangle given as legs the two real values read in step b.
If the integer value read in step b is 1, find a leg of a right triangle given as hypotenuse and other leg the two real values read in step b. (assume larger of two real values read in first data field)

d. Write out and label the values of the three sides of the right triangle correct to hundreths.

e. Go back and read another data card until all data has been processed according to step a.

10.4 Draw a flow chart and write a program in FORTRAN IV to perform the following tasks:

a. Read from each input data card an employee number, his hourly rate, and the number of hours he worked in one week. The input card format will be as follows:

employee's number (integer)	columns 1-10
hourly rate (real)	columns 11-15
number of hours worked per week (real)	columns 16-20

The last card of the input deck is a trailer card and will have a zero for the employee's number.

b. Determine the employee's gross pay for the week, the amount of his Social Security deduction (assume 4 1/2% of gross pay), and his net pay for the week.

c. Calculate the total amount the company must pay to Social Security and the total it must pay to all the employees.

d. Write out in columns labeled appropriately the employee's number, hourly rate, number of hours worked in the week, the amount of Social Security deduction, and the net pay.

e. After the last data card is processed, write out and label the total Social Security and the total paid to all employees by the company.

Subscripts - Arrays

Many times it is necessary to perform the same calculations for many different values stored in the computer's memory unit. The use of subscripts as they are used in mathematics are useful to give a different variable name for a large number of values. In mathematics, a subscript is used to define the position of a variable in a list of variables. The general form is as follows:

$$x_1, x_2, x_3, \cdots, x_n$$

This would describe n different variables. A list of variables or values such as this is sometimes referred to as an array. Arrays are described as one-dimensional, two-dimensional, three-dimensional, and so on. To illustrate, consider the following:

One-dimensional

2
5
6
4
7

Three-dimensional

1	6	4
3	2	1
5	1	6
7	4	2

Rewriting the above one-dimensional array horizontally, we have

2, 5, 6, 4, 7

For convenience, a name is generally given to an array. Suppose the above array is given the name K. Then the first element in the array K is 2, the second element in the array K is 5, and so on. Generally, we write

$K_1 = 2, K_2 = 5, K_3 = 6$, etc.

By using subscripts, it is possible to show the position a particular number has in the array.

The FORTRAN notation for elements in an array is similar to the mathematical notation. To illustrate, consider the following:

K(1) = 2, K(2) = 5, K(3) = 6, etc.

The general form is a variable name followed by parenthesis enclosing a subscript. The name used for a subscripted variable name is selected following the rules for nonsubscripted variable names. The type of elements in the array, real or integer, will determine the type of variable name for the array.

To illustrate, consider the following:

1.1, 2.3, 5.6, -8.6

Since the elements in the array are real numbers, a real type variable name would be required to name the array. For example, the array might be called array A. Then

$A_1 = 1.1, A_2 = 2.3, A_3 = 5.6, A_4 = -8.6$

or in FORTRAN notation,

A(1) = 1.1, A(2) = 2.3, A(3) = 5.6, A(4) = -8.6

All elements in an array must be of the same type. For instance, all of the elements might be integer values or all of them might be real values.

A subscript can be an arithmetic expression written in one of

seven general forms. For most computers the type of the expression must be only integer.

Let v be an unsigned nonsubscripted integer variable; c and c' be unsigned integer constants, then the seven general forms are as follows:

(1) c
(2) v
(3) v + c
(4) v - c
(5) c * v
(6) c * v + c'
(7) c * v - c'

To illustrate, consider the following:

Subscripts	Subscripted Variable Names
3	A(3)
I	A(I)
I + 2	A(I+2)
I - 1	A(I-1)
2*I	A(2*I)
2*I + 3	A(2*I+3)
3*I - 1	A(3*I-1)

NOTE: The above seven general forms of subscripts are the only acceptable ones for some compilers.

If a program is to contain an array, the array dimensions must be communicated to the compiler. This is done by means of a DIMENSION statement. There are other statements that can be used for this purpose but will not be discussed in this study.

The general form is

DIMENSION v(k), v(k), ... , v(k)

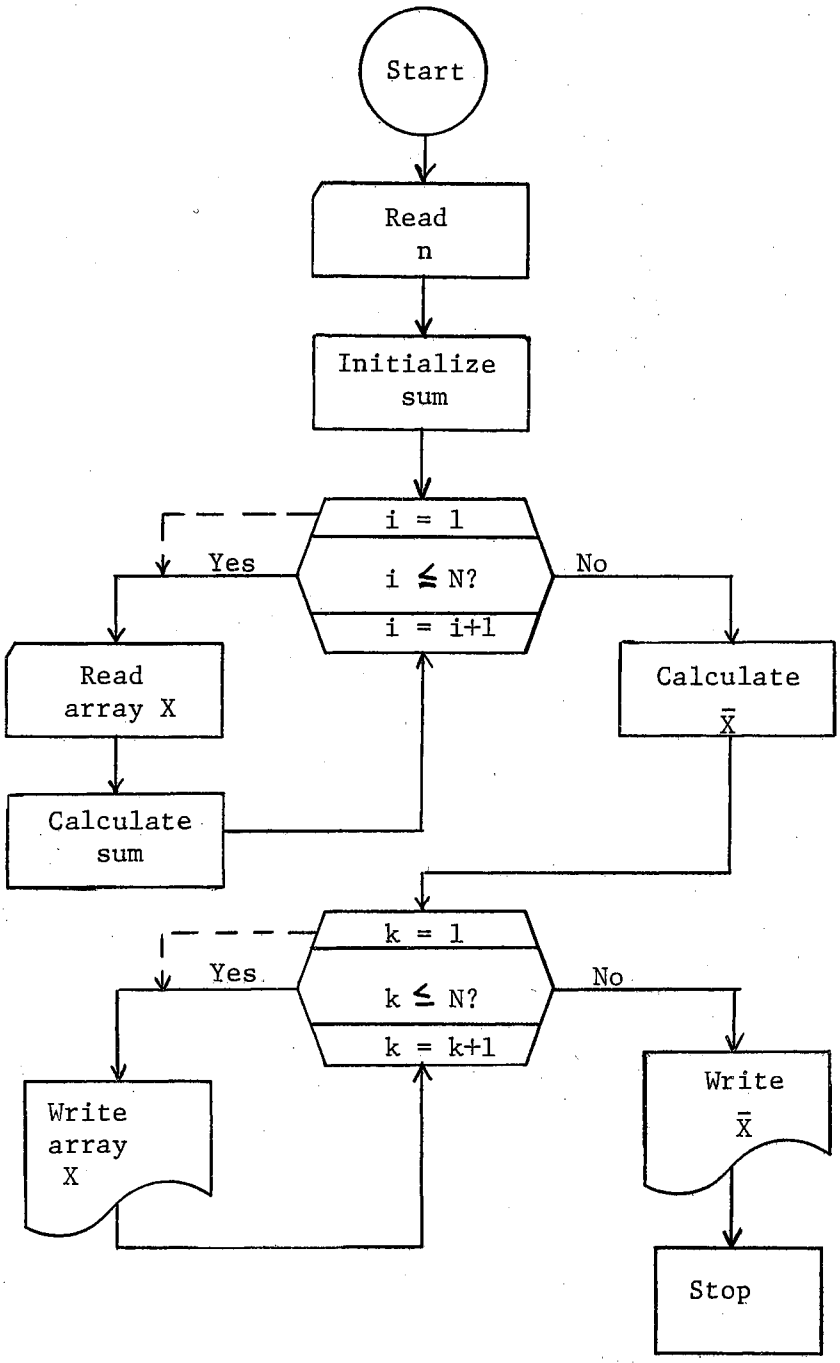
where v is the name of a variable which will be subscripted, and for k, an integer constant is placed which will be the maximum value a subscript for the variable may have.

To illustrate, consider the following:

DIMENSIOM S(10), Y(20), A(50)

This statement informs the compiler that the variables X, Y and A will

Flow Chart For Sample Problem 6



be subscripted variable names in the source program. Also, the maximum value of any subscript for X will be 10; for Y, 20; and for A, 50.

The DIMENSIOM statement must precede the use of any subscripted variable is the source program. Hence, if the DIMENSION statement appears as the first statement, this rule will never be violated.

Sample Problem 6: Write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card an integer value for N (N 100). N will be the number of real values in an array X to be read from the following data cards with one per card in 1-10.

b. Calculate the arithmetic mean

$$\bar{X} = \sum_{i=1}^{N} X_i/N$$

c. Write out the array X placing one element per line.

d. Write out the calculated value for $\bar{X}$.

Program for Sample Problem 6

```
1   567
C     NAME
C     SAMPLE PROBLEM 6 - ARITHMETIC MEAN
      DIMENSION X(100)
  100 FORMAT(I5)
  101 FORMAT(F10.0)
  102 FORMAT(1X, F10.2)
  103 FORMAT(1X, 5HXBAR=, F10.2)
      READ(5,100) N
      SUM = 0.0
      DO 2 I = 1, N
      READ(5,101) X(I)
    2 SUM = SUM + X(I)
      REALN = N
      XBAR = SUM/REALN
      DO 4 K = 1,N
    4 WRITE(6,102) X(K)
      WRITE(6,103) XBAR
      STOP
      END
```

Remember each time a READ statement is executed, a data card is read.

Hence, in the above program, the program segment

```
      DO 2 I = 1,N
    2 READ(5,101) X(I)
```

will cause N data card to be read.

Exercises 11.

11.1 List the different forms the subscript of an array variable can have.

11.2 Is there any difference between the two variable names? Explain.

X3 X(-3)

11.3 Write a program in FORTRAN IV to perform the following tasks:

a. Read in 10 values for arrays X and Y where elements for the array X are placed in columns 1-10 and the elements for the array Y are placed in columns 11-20 of the same card.

b. Calculate the following

$$T = \sum_{i=1}^{10} X_i Y_i + \sum_{i=1}^{10} (X_1 - Y_1)$$

c. Write out the array X and Y with one value for each on a line.

d. Write out the value of T calculated in step b.

Implied DO

In Sample Problem 6, each value of X_i was placed on a separate data card. Suppose we were to place four per card in field widths of 10 columns each. The following program segment might be used to accomplish this task.

```
      DIMENSION X(100)
   10 FORMAT(I5)
   11 FORMAT(4F10.0)
      READ (5,10) N
      DO 2 I = 1,N,4
    2 READ(5,11) X(I), X(I+1), X(I+2), X(I+3)
           .
           .
           .
```

There is a problem that arises from the technqiue used in the above program segment. That is, suppose the value of N is not divisable by 4 (or any other number arbitrarily selected). For example, if N = 6, the READ statement will read X(1), X(2), X(3), X(4) from the first data card. From the second data card the READ statement will read X(5), X(6), X(7), X(8). But N = 6. This technique will work, however allowance may have to be made for extra elements in any array. In this example, X(7), X(8) would be read as zeros if the data fields were blank in the third and fourth fields of the second card.

The FORTRAN language provides a procedure called the implied DO to alleviate this problem. The implied DO is used only with a READ or a WRITE statement. To illustrate, consider the following:

```
  100 FORMAT(4F10.0)
      READ(5,100) (A(I), I=1,100)
```

The part

```
A(I), I= 1,10
```

in the READ statement is similar to the DO statement. Hence, the name implied DO. In this example, I is set equal to 1 and A(1) is read from the first data field according to the corresponding FORMAT statement. Then I is incremented by one (if increment is omitted) and A(2) is read from the second data field in the first data card according to the FORMAT statement, and so on until the list is satisfied. That is, until A(10) is read. The number of data cards required depends

only on the FORMAT statement and the test value in the implied DO. In this example, four numbers are read from the first data card, four numbers from the second data card, and two numbers from the third data card. If the FORMAT statement and READ statement were

```
100 FORMAT(8F10.0)
    READ(5,100) (A(I), I = 1,10)
```

then eight numbers would be read from the first data card with two numbers being read from the second data card.

Using the implied DO in Sample Problem 6, we would have

```
    DIMENSION X(100)
100 FORMAT(I5)
101 FORMAT(8F10.0)
    READ(5,100) N
    READ(5,101) (X(I), I = 1,N)
          .
          .
          .
```

The implied DO can also be used with the WRITE statement in the same way it is used with the READ statement.

Exercises 12.

12.1 The following READ statement and corresponding FORMAT statement will read how many elements into the array B? How many data cards will be required?

```
    DIMENSION B(25)
100 FORMAT(5F15.0)
    READ(5,100) (B(J), J = 1, 16)
```

12.2 Assume the array X has 30 elements stored in memory. Write a WRITE statement and the corresponding FORMAT statement to write the elements of the array X on the printer placing 6 elements per line.

12.3 An efficient method of evaluating a polynomial

$$P_4(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

is to rewrite the polynomial in nested form as

$$P_4(x) = ((((a_4x + a_3)x + a_2)x + a_1)x + a_0$$

Write a program in FORTRAN IV to perform the following tasks:

a. Read ten values of b and one value of X and compute

$$y = b_9x^9 + b_8x^8 + b_7x^7 + b_6x^6 + \ldots + b_1x + b_0$$

b. Write out the values of b 5 values per line. Write out and label the values of X and Y.

c. Halt execution.

Two-Dimensional Arrays

In mathematics two-dimensional arrays are used extensively. Such an array is an arrangement of values in rows and columns. Each of the elements in the array may be referred to by a subscripted variable name having two subscripts which are separated by a comma. The first subscript designates a row and the second subscript designates a column.

To illustrate, consider the following two-dimensional array A written in the conventional mathematical notation:

	column 1	column 2	column 3	column 4
row 1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
row 2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
row 3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$

The above example has three rows and four columns. The element in the second row and third column is given by:

$$a_{2,3}$$

Or given the element

$$a_{3,1}$$

would mean the element is in the third row and first column. In mathematics a two-dimensional array is called a matrix. In this example

the matrix would be called a 3 X 4 matrix.

Using the FORTRAN IV notation for arrays, the above two-dimenational array would be as follows:

A(1,)	A(1,2)	A(1,3)	A(1,4)
A(2,1)	A(2,2)	A(2,3)	A(2,4)
A(3,1)	A(3,2)	A(3,3)	A(3,4)

Consider the two-dimensional array B or 3 X 3 matrix that follows:

3	-4	-2
2	1	0
-3	5	6

Then the element B(2,3) would have a value of 0 and the element B(1,2) would have a value of -4.

There are many applications in mathematics using matrices. In fact, there are courses entirely devoted to the study of matrices and their operations. Further discussion on matrices will be given in the following sections. Also, the use of two-dimensional arrays has many applications in non-mathematical problems.

Sample Problem 7: Write a program in FORTRAN IV to perform the following tasks:

a. Read from a data card a value for M and n. (Assume m, n $\leq$ 50)

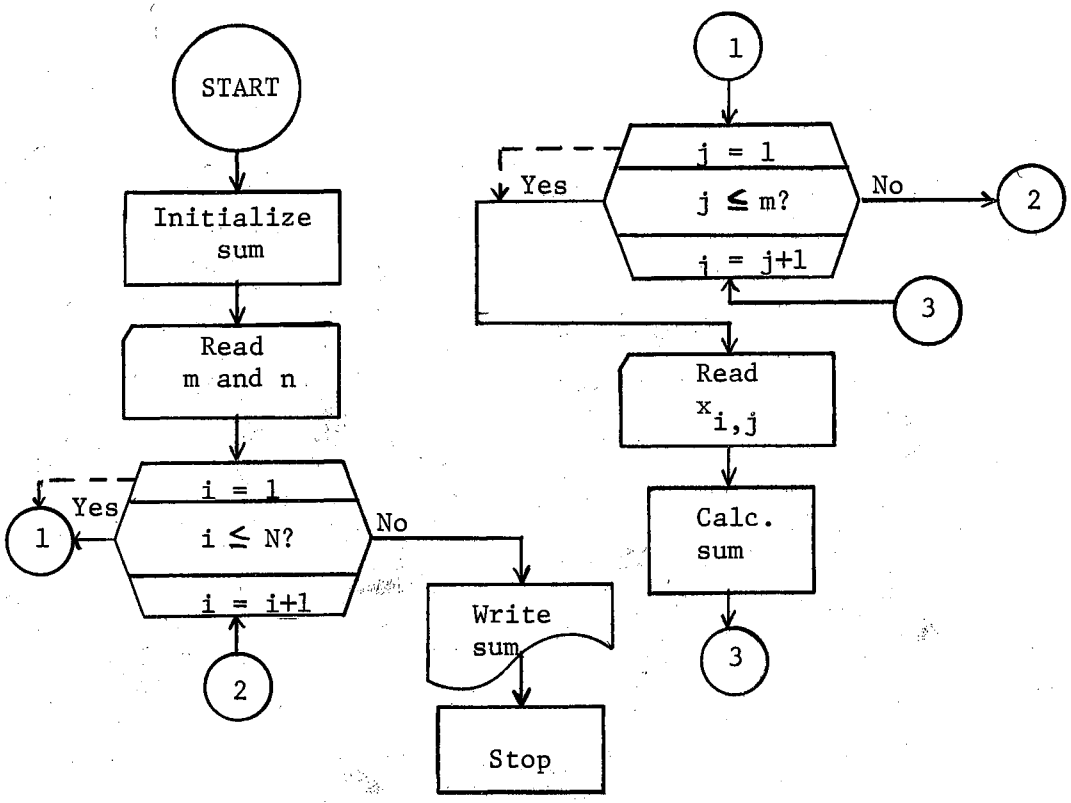
b. Read in a m X n matrix or array named X.

c. Find the sum of all the elements in the array.

d. Print out the value of the sum determined in step c.

e. Halt Execution.

Flow Chart For Sample Problem 7



Program for Sample Problem 7

```
1   567
C      NAME
C      SUM OF THE ELEMENTS OF A M X N MATRIX
       DIMENSION X(50,50)
  100  FORMAT(2I5)
  101  FORMAT(8F10.0)
  102  FORMAT(1X, 15HSUM OF ELEMENTS, F10.2)
       SUM = 0.0
       READ(5,100) M, N
       DO 1 I = 1, N
       DO 1 J = 1, M
       READ(5, 101) X(I,J)
     1 SUM = SUM + X(I,J)
       WRITE(6,102) SUM
       STOP
       END
```

Careful notice should be taken on the nested DO statements in the above program. The above program will read one element of the array from a single data card. Hence, in this program m·n data cards would be needed.

Again, an implied DO statement can be used to read in a two-dimensional array similar to the one-dimensional array.

Second Program for Sample Problem 7

```
1   567
C     NAME
C     SUM OF ELEMENTS OF M X N MATRIX
      DIMENSION X(50,50)
  100 FORMAT(2I5)
  101 FORMAT(8F10.0)
  120 FORMAT(1X, 15HSUM of ELEMENTS, F10.2)
      SUM = 0.0
      READ(5,101) ((X(I,J), I = 1,N),J = 1, M)
      DO 1 I = 1, N
      DO 1 J = 1, M
    1 SUM = SUM + X(I,J)
      WRITE(6,102) SUM
      STOP
      END
```

The above program will read in the elements by rows with 8 elements per data card until the list is satisfied.

Exercises 13.

13.1 The following READ statement will read how many elements in the array A? How many data cards will be required? Elements are read in by rows or columns?

```
      DIMENSION A(100,100)
  100 FORMAT(5F15.0 )
      READ(5,100) ((A(M,N), N = 1, 8), M = 1, 10)
```

13.2 Write a program in FORTRAN IV to perform the following tasks:

a. Read in the values given in the table below which gives the enrollments by classes at a large university for a period of six years.

Year	1962	1963	1964	1965	1966	1967
Freshmen	4,251	4,680	4,842	5,015	5,215	5,356
Sophomore	3,860	4,137	4,456	4,705	4,885	5,092
Juniors	3,420	3,675	3,972	4,310	4,521	4,730
Seniors	2,835	3,305	3,450	3,625	4,206	4,380

b. Print out and label the values in the table in the above form.

c. Calculate the sum of each of the rows.

d. Calculate the sum of each of the columns.

e. Print out the tables of values with the sums found in steps c and d with these sums being printed out as a seventh column of values and fifth row of values. Label this column and row TOTAL.

f. Calculate and print the average enrollment at this school for the years 1962 through 1967

HINT: It will be advantageous to name the various sums found as elements of the array. i.e., A(5,1) is the sum of the first column, etc.

Identification and Development of Advanced Mathematical Concepts

Limit Concept

The concept of a limit plays a fundamental role in higher mathematics. The limit concept allows one to go from algebra and geometry to higher mathematics consisting of the calculus and all of its applications.

To give an informal meaning to this concept consider the following question: As x approaches the number 3, what value, if any, would the expression (3x - 5) approach? One might make a table of values such as in Figure 7.

x	2.8	2.9	2.95	2.98	2.99	2.999
3x - 5	3.4	3.7	3.85	3.94	3.97	3.997

Figure 7.

From observing the values of $(3x - 5)$ as x approaches 3, it appears the value of $(3x - 5)$ is approaching 4. Hence, symbollically we write

$$\lim_{x \to 3} (3x - 5) = 4$$

The fact that $(3x - 5) = 4$ when $x = 3$ is really of no importance in regard to the concept of a limit.

For example, consider the following expression:

$$\lim_{x \to 3} \frac{1/x - 1/3}{x - 3}$$

By substituting the value of x into the expression, the expression would be undefined since division by zero is undefined. However, if a table of values are determined as follows in Figure 8, it appears the limit of the expression is -1/9 or -0.111... .

x	2.8	2.9	2.98	2.99	2.999
$\frac{1/x - 1/3}{x - 3}$	-.119	-.115	-.112	-.111	-.111

Figure 8

As a third example, consider the following:

$$\lim_{x \to 0} (1/2)^{1/x^2}$$

From the table of values given in Figure 9, it appears that $(1/2)^{1/x^2}$ approaches 0 as x is close to 0.

X	1.0	.5	.25	.125	.0625
$(1/2)^{1/x^2}$	.500	.063	.000	.000	.000

Figure 9.

One way of determining the table of values is by the use of the computer. From the examples above, one can see the arithmetic involved becomes a tiresome job.

Although it is not always possible to determine if the limit of an expression exists by forming a table of values, in most instances the table of values will show the values of the expression approaching some fixed number if the limit does exist.

Since the expression is sometimes undefined at $x = a$ in the general form

$$\lim_{x \to a} f(x)$$

and yet will have a limit as x approaches a, some means is needed to signal the computer that the difference between values of f(x) as $x \to a$ are within a certain tolerance, say ε, hence, stop execution. i.e., if we start with $x_1 = a - 1$, and take $x_2 = 1/a(a + x_1)$, $x_3 = 1/2(a + x_2)$, ... , $x_n = 1/2(a + x_{n-1})$ then x will approach a. Now, if for any ε we choose, we can find x_i and x_j such that $|f(x_i) - f(x_j)| < \varepsilon$ then this would imply that f(x) is approaching some fixed value say b. The method chosen above to force x to approach a is arbitrary. However, the above method can be used as an algorithm and will be possible to program for a computer.

Problem Sample 8: Write a program in FORTRAN IV to create a table of values for the following function as x approaches 1.

$$f(x) = \frac{\sqrt{x} - 1}{x - 1}$$

Start with x = 0 and use the method described above to force x to approach 1.

Write out the value of x and the corresponding value of f(x) in a two column array.

Use as the value for ε , .0001. i.e., when $|f(x_i) - f(x_j)| < \varepsilon$, stop execution.

Program For Sample Problem 8

```
1   567
C      NAME
C      CALCULATING TABLE OF FUNCTIONAL VALUES
  100 FORMAT(4X,8HX VALUES,5X,15HFUNCTION VALUES)
  101 FORMAT(1X, E13.6,3X,E13.6)
      WRITE(6, 100)
      KOUNT = 1
      EPSILN = .0001
      X = 0.
    4 FCTN1 = (X**.5 - 1.)/(X - 1.)
      IF(KOUNT - 1) 2,3,2
    2 DIFF = FCTN1 - FCTN2
      IF(DIFF) 5,6,6
    5 DIFF = -DIFF
    6 IF(DIFF - EPSILN) 7,3,3
    3 WRITE(6,101) X, FCTN1
      KOUNT = KOUNT + 1
      IF(KOUNT - 50) 8,8,7
    8 X = .5*(11 + X)
      FCTN2 = FCTN1
      GO TO 4
    7 STOP
      END
```

The output for the above program would appear as follows:

X VALUES	FUNCTION VALUES
0.000000E 00	0.100000E 01
0.500000E 00	0.585786E 00
0.750000E 00	0.535898E 00
0.875000E 00	0.516685E 00
0.937500E 00	0.508067E 00
0.968750E 00	0.503969E 00
0.987375E 00	0.501972E 00
0.992188E 00	0.500984E 00
0.996094E 00	0.500504E 00
0.998047E 00	0.500275E 00

From the table of values it would indicate the

$$\lim_{x \to 1} f(x) = .5$$

There are several techniques used in the program for Sample Program 8 that attention should be given to:

(1) The use of the computer set up by the variable name KOUNT is used for two purposes. (i) forces program to go through the program at least twice so that there will be values for FCTN1 and FCTN2. (ii) to assure that the program will not remain in a loop if no two values for FCTN1 and FCTN2 exist such that $|FCTN1 - FCTN2| < .0001$.

(2) It is required to take the absolute value of the difference of two successive values of the function. The segment that follows assures that this difference is positive.

```
      .
      .
      .
2 DIFF = FCTN1 - FCTN2
  IF(DIFF) 5,6,6
5 DIFF = - DIFF
6 IF(DIFF - EPSILN) 7,3,3
```

(3) The statement X = .5*(1.+ X) will give a new value of X that will get closer and closer to 1.

(4) The number 50 used in the statement IF(KOUNT - 50) 8,8,7 is arbitrary. This prevents the computer from staying in an endless loop if the function or expression fails to have a limit.

Exercises 14.

14.1 Using the above program with the appropriate changes, determine the table of values of the following functions. Give what appears to be the limit of the function if it exists.

a. $\lim_{x \to 2} (x^2 - 3x + 2)$

b. $\lim_{x \to -2} \dfrac{\dfrac{1}{x+1} + 1}{x + 2}$

c. $\lim_{h \to 0} \dfrac{\sqrt{2 + h} - \sqrt{2}}{h}$

14.2 Rewrite the above program to determine a table of values of an expression of the form

$$\lim_{n \to \infty} f(n)$$

where ∞ means infinity or to get larger without bound.

14.3 Use the program written for 14.2 to determine the limit of the following expressions if they exist.

a. $\lim_{n \to \infty} (1 + 1/n)^n$

b. $\lim_{n \to \infty} \frac{n}{2n - 1}$

c. $\lim_{n \to \infty} \frac{3n - 1}{5}$

Application of the Limit Concept

Instantaneous Velocity

Given an object moving along a path in accordance with some definite law, the question might be raised as to the velocity or speed of this object. Generally, when it is given that the speed of an automobile is 60 miles per hour or the speed of an object is traveling at a rate of 100 feet per second, the average speed or velocity is implied. The question then might be what about the speed or velocity at any given instant. This velocity is referred to an instantaneous velocity.

Looking at this problem geometrically, consider Figure 10.

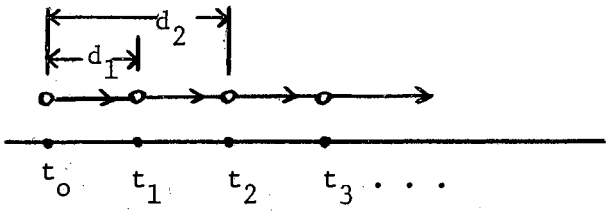


Figure 10.

Let t_o, t_1, t_2 ... be the elapsed time of some initial time t_o and d_1, d_2, ... be the distance traveled in this elapsed time.

The average speed or velocity is given by

$$\text{average speed} = \frac{\text{distance traveled}}{\text{time elapsed}}$$

Hence, the average speeds for the above respective times and distance would be

$$r_1 = d_1/t_1,\ r_2 = d_2/t_2, \text{ etc.}$$

However, it is not so simple to determine the instantaneous velocity of a moving object at a given instant t.

Consider a definite law of motion, so that distance d is a function of time t. Suppose we want the instantaneous velocity at time t_o. If there is to be an instantaneous velocity, it is logical to expect this to be a value that the average velocities would approach for time t as t approaches t_o.

That is

$$\text{instantaneous velocity} = \lim_{t \to t_o} \frac{d - d_o}{t - t_o}$$

In higher mathematics, the instantaneous velocity is defined to be as given above.

To illustrate, consider the following:

Sample Problem 9: Find the instantaneous velocity v at time $t_o = 3$ where the distance d is given as a function of time t

$$d(t) = 128t - 16t^2$$

where t is in seconds and d is in feet.

Solution: From the definition given above

$$v = \lim_{t \to t_o} \frac{d - d_o}{t - t_o}$$

or

$$v = \lim_{t \to 3} \frac{d(t) - d(3)}{t - 3} = \lim_{t \to 3} \frac{128t - 16t^2 - 240}{t - 3}$$

A program is needed similarly to the program of Sample Problem 8 to determine a table of values for the above expression to determine if the expression is approaching some fixed value as $t \to 3$. The program needed is as follows with the corresponding output of values (start with time t = 2 and let t approach 3)

Program for Sample Problem 9.

```
1   567
C      NAME
C      INSTANTANEOUS VELOCITY
  100 FORMAT(4X,8HT VALUES,5X,15HFUNCTION VALUES)
  101 FORMAT(1X,E13.6,3X,E13.6)
      WRITE(6,100)
      KOUNT = 1
      EPSILN = .001
      T = 2
    4 FCTN1 = (128.*T - 16*T**2 - 240.)/(T - 3.)
      IF(KOUNT - 1) 2,3,2
    2 DIFF = FCTN1 - FCTN2
      IF(DIFF) 5,6,6
    5 DIFF = - DIFF
    6 IF(DIFF-EPSILN) 7,3,3
    3 WRITE(6,101) T, FCTN1
      KOUNT = KOUNT + 1
      IF(KOUNT - 50) 8,8,7
    8 T = .5*(T + 3.)
      FCTN2 = FCTN1
```

```
    GO TO 4
  7 STOP
    END
```

T VALUES	FUNCTION VALUES
0.000000E 00	0.800000E 02
0.150000E 01	0.560000E 02
0.225000E 01	0.550000E 02
0.262500E 01	0.380000E 02
0.281250E 01	0.350000E 02
0.290625E 01	0.335000E 02
0.295313E 01	0.327500E 02
0.297656E 01	0.323750E 02
0.298828E 01	0.321875E 02
0.299414E 01	0.320938E 02
0.299707E 01	0.320469E 02
0.299854E 01	0.320208E 02
0.299927E 01	0.320000E 02

It can be seen from the above output that the functional values seem to be approaching a value close to 32. Hence, the instantaneous velocity is approximately 32 feet/second.

Exercises 15.

15.1 Modify the program for Sample Problem 9 to perform the following tasks:

a. Read in on a data card a value for t_o. (In the Sample Problem $t_o = 3$)

b. Use as a starting time $t_o - 1$. For the Sample Problem 9, starting time was used as 2.

c. The program should be written so that control will transfer back to read in another data card with a new value for t_o. This should continue until t_o is read as 0.

15.2 The only required change in the above program to determine instantaneous velocity for a different distance function is to change the arithmetic statement for FCTN1. Given the distance function

$$d(t) = 96t - 1/2\ t^3$$

find the instantaneous velocity at times t = 2, t = 10, t = 30 seconds.

15.3 An oil tank is being emptied. If there are G gallons of oil in the tank at time t, where $G = 67{,}500 - 9000t + 300t^2$ and t is measured in minutes, how many gallons of oil per minute are running out (a) at t = 0? (b) at t = 2?

Tangent to a Curve

It is possible to define the tangent line to a circle at a point P on the circle either as the line perpendicular to the radius at P or as the line intersecting the circle in only one point P. However, neither one of these definitions will apply to a general curve such as the one in Figure 11.

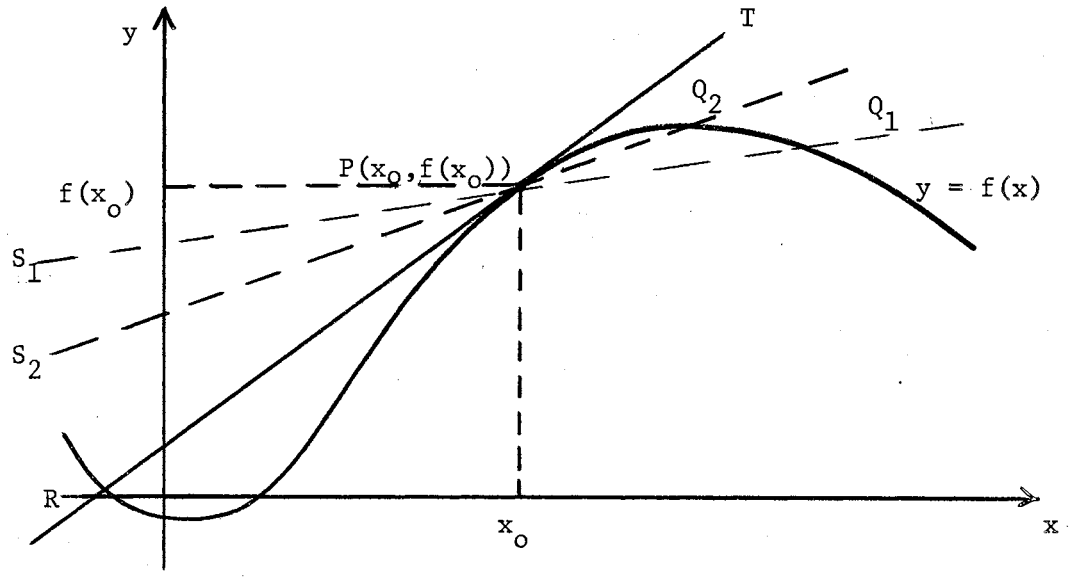


Figure 11.

The line T, which appears to be what we might expect to be tangent at the point P, intersects the curve also at point R. Also, since a general curve doesn't have a radius, the other definition would not be

meaningful.

The tangent line T to the curve $y = f(x)$ at the point P should be the line intersecting the curve at only point P within a small neighborhood of P. By this, it means, if any other line or secant S is taken on point P and some point Q, the secant line should approach the tangent T as the point Q approaches the point P.

The equation of the tangent line at the point $P(x_o, f(x_o))$ to the curve $y = f(x)$, could be determined if the slope m of the line were known by the following equation of a straight line

$$y = f(x_o) = m(x - x_o)$$

Consider the secant S_1 intersecting the curve at point $P(x_o, f(x_o))$ and another point $Q_1(x_1, f(x_1))$.

The slope of the secant S_1 would be

$$m_1 = \frac{f(x_1) - f(x_o)}{x_1 - x_o}$$

If another secant S_2 is taken where x_2 is chosen to x_o, then the slope of this secant is

$$m_2 = \frac{f(x_2) - f(x_o)}{x_2 - x_o}$$

Geometrically, it appears that the slope of the secants as x approaches x_o, should approach what would be the slope of the tangent at the point P. In limit notation, this would be

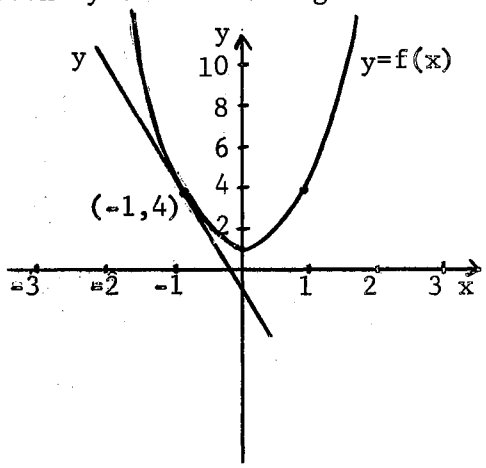
$$m = \lim_{x \to x_o} \frac{f(x) - f(x_o)}{x - x_o}$$

Hence, in higher mathematics, the slope of the tangent to a curve at a point is given by this limit, if it exists.

Sample Problem 12: Find an equation of the tangent line T to the graph of the function f defined by $f(x) = 3x^2 + 1$ at the point (-1,4).

Solution: The slope of the tangent is given by the following limit if it exists.

$$m = \lim_{x \to -1} \frac{f(x) - f(-1)}{x - (-1)}$$



or

$$m = \lim_{x \to -1} \frac{3x^2 + 1 - 4}{x + 1}$$

$$m = \lim_{x \to -1} \frac{3x^2 - 3}{x - 1}$$

If the expression is simplified, it becomes

$$m = \lim_{x \to -1} \frac{3(x^2 - 1)}{x + 1} = \lim_{x \to -1} \frac{3(x + 1)(x - 1)}{x + 1}$$

$$= \lim_{x \to -1} 3(x - 1)$$

$$= -6$$

The above value found could be verified by using the program written for Exercise 15.1.

If the slope of the tangent at the point (-1,4) is -6, then using the equation for a line having a given slope and a given point, the equation is found to be

$$y - 4 = -6(x - (-1))$$

or $$y - 4 = -6(x + 1)$$

or $$6x + y = -2$$

Exercises 16.

16.1 If $h(x) = 2x^2$, find the slope of the tangent line to the curve at each of the points.

a. (-2, h(-2)), (-1, h(-1)), (0,h(0)), (1,h(1)), (2,h(2))

Use program for Exercise 15.1 to determine the slope. Sketch the curve of h and the tangent lines at each of the points.

16.2 Give the equation of the tangent to the curve of 16.1 at the point (-2,h(-2)).

The Derivative of a Function

One of the important uses of the limit concept is to define the derivative of a function. Much of the calculus is concerned with the properties and applications of the derivative.

The derivative is defined as follows: The derivative of the function f at the number x is

$$\lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

if this limit exists.

If this limit doesn't exist, then the function f doesn't have a derivative at x. The notation f'(x) is used to represent the derivative of f at x. This is read as "f prime of x".

Hence

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

To illustrate, consider the following problem.

Sample Problem 13: Find the derivative f' of the function $f(x) = 3x^2$ at any x if it exist.

Solution: By definition,

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

$$= \lim_{h \to 0} \frac{3(x + h)^2 - 3x^2}{h}$$

$$= \lim_{h \to 0} \frac{3x^2 + 6xh + 3h^2 - 3x^2}{h}$$

$$= \lim_{h \to 0} \frac{6xh + 3h^2}{h}$$

$$= \lim_{h \to 0} \frac{h(6x + 3h)}{h}$$

$$= \lim_{h \to 0} (6x + 3h)$$

$$= 6x$$

The $\lim_{h \to 0} (6x + 3h) = 6x$ seems to be a logical answer since the term $3h$ should approach 0 as h approaches 0. A table of values would show this to be true. Hence, we assume the derivative

$$f'(x) = 6x$$

of the function $f(x) = 3x^2$.

If we wanted the derivative of this function at some particular x, say 2, all is necessary is to substitute 2 for x in the derived derivative $f'(x)$. Hence, $f'(2) = 6(2) = 12$. Geometrically, the above problem can be shown as in Figure 12. Notice the expression $\frac{f(x + h) - f(x)}{h}$ is the slope of the secant or line passing through the two points $(x, f(x))$ and $(x + h, f(x + h))$. Now the limit of this expression is the slope of the tangent to the curve at the point $(x, f(x))$ as discussed in the previous section. Thus, the derivative (if it exist) of a function at a given value of x is the slope of the tangent to the curve representing the function.

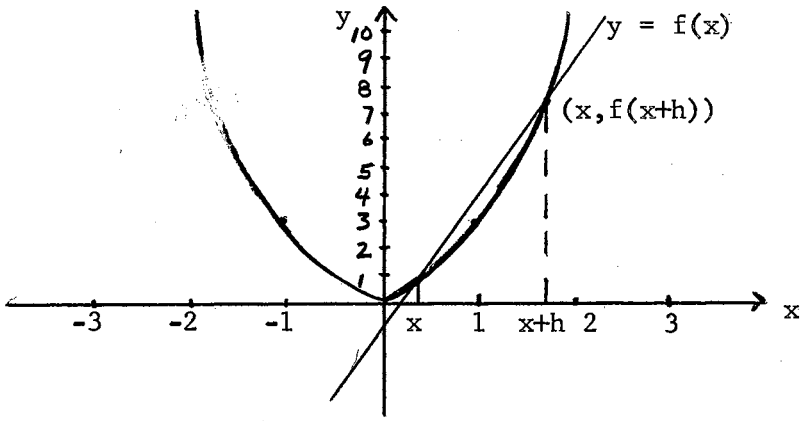


Figure 12

Likewise, the expression

$$\frac{f(x + h) - f(x)}{h}$$

would represent the average velocity of a moving object if f is the distance function and x represents the time. Hence, the limit of this expression, if it exist, is the instantaneous velocity at any given time x. Therefore, the derivative of the distance function at any given time will give the instantaneous velocity of the moving object at this time.

In general, we can say the derivative of a function at any x is the instantaneous rate of change of one quantity with respect to another. This concept has many applications in addition to the two given above and are discussed in undergraduate calculus.

Exercises 17.

17.1 If $f(x) = x^2 - 3x$, use the definition of a derivative to find:

a. $f'(2)$ b. $f'(-3)$

Use program of Exercise 15.1 for determining the necessary limit if needed.

17.2 If $g(t) = t^3 + 4t - 5$, use the definition of a derivative to find:

a. $g(4)$ b. $g(-2)$

17.3 A ball is thrown directly upward from the ground. The distance (feet) the ball is above the ground at any given time t (seconds) is

$$d(t) = -16t^2 + 40t$$

a. Find the instantaneous velocity of the ball 3 seconds after it is thrown.

b. What is the maximum distance the ball will be from the ground? (Hint: when instantaneous velocity is zero)

c. What is the time required after the ball is thrown for the ball to strike the ground?

17.4 Given the function $f(x) = x^3 - 5x + 8$ find the following:

a. Slope of the tangent to the curve of the function at the point $(-1, f(-1))$.

b. The equation of the tangent to the curve at the point $(-1, f(-1))$.

The Definite Integral

Another very important concept which plays a principal part in the calculus is the integral. Before defining this, consider another problem, that of finding the area under a given curve and the x - axis.

Let f be a function that has a continuous smooth curve from some value a to a value b. (Figure 14)

The area under the curve is the region labeled R which is the region under the curve and bounded by the x - axis and the vertical lines at a and b.

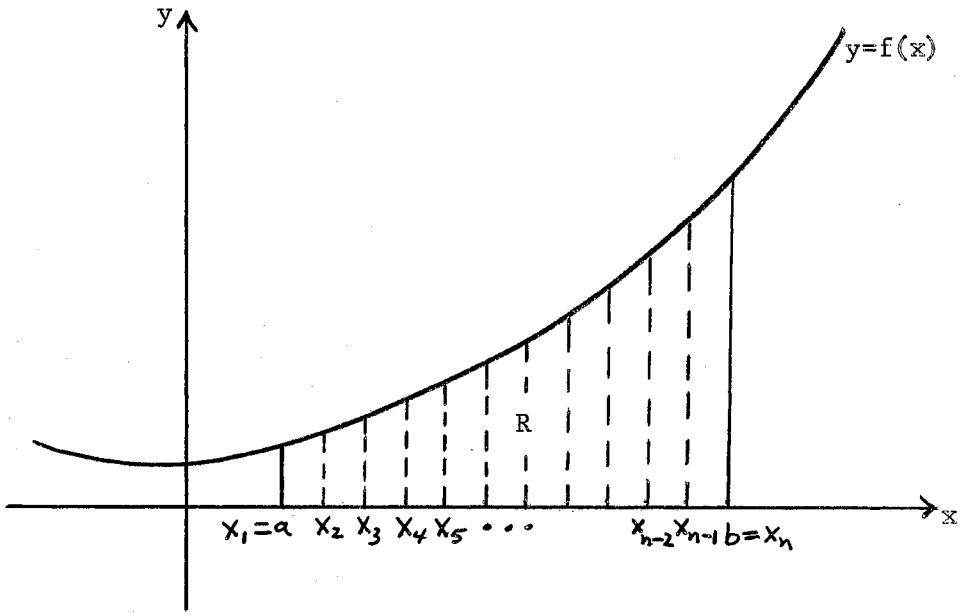


Figure 14.

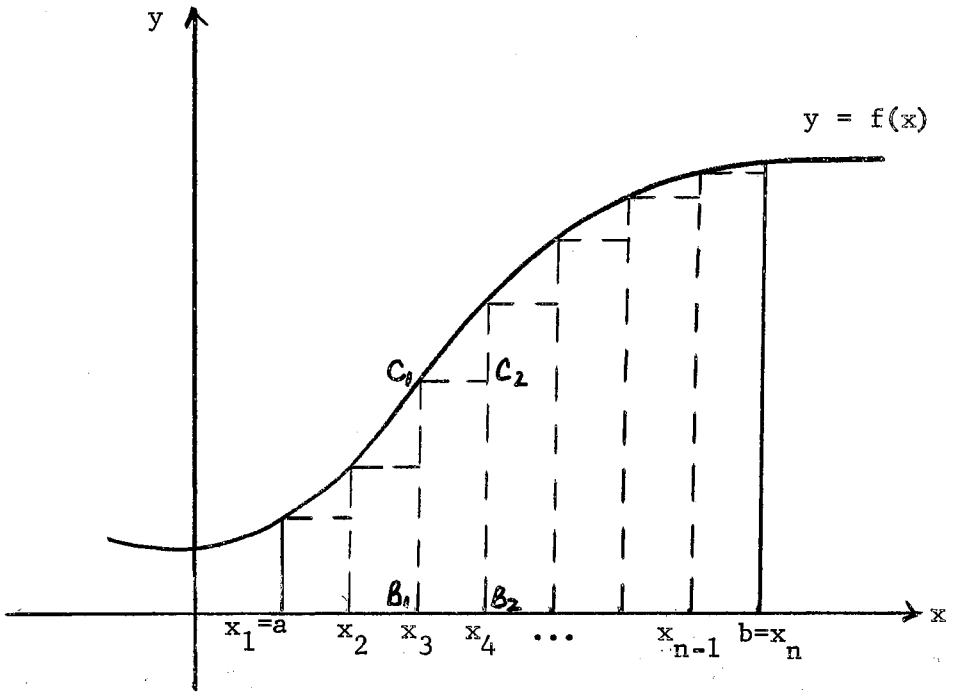


Figure 15.

If n equal spaces are taken on the x-axis from a to b, calling them $x_1, x_2, x_3, \ldots, x_n$ where $x_1 = 2$ and $x_n = b$, then Figure 15 with the width of each interval being $(b - a)/n$.

The area of the region R could be approximated by the inscribed rectangles as shown. Since the area of a rectangle is given by the product of the length and width, the area of one of the rectangles, say $B_1C_1C_2B_2$, would be

$$\text{area } B_1C_1C_2B_2 = \text{width} \cdot \text{length}$$

$$= \frac{b - a}{n} \cdot f(x_3)$$

applying the area of a rectangle to the entire region, the approximation would be

$$\text{area} = \sum_{i=1}^{n} \frac{b - a}{n} \cdot f(x_i)$$

The Greek symbol $\sum$, called sigma, denotes a summation. Hence, the above expression can be shown as follows:

$$\sum_{i=1}^{n} \frac{b - a}{n} f(x_i) = \frac{b - a}{n} f(x_1) + \frac{b - a}{n} f(x_2) + \ldots + \frac{b - a}{n} f(x_n)$$

$$\text{or} \qquad = \frac{b - a}{n}(f(x_1) + f(x_2) + \ldots + f(x_n))$$

Geometrically, it can be seen as the value of n becomes larger, the approximation should be closer to the real area of the region. In fact, it is possible to prove mathematically, that the limit of the summation

$$\sum_{i=1}^{n} \frac{b - a}{n} f(x_i)$$

is the true area bounded by the curve, the vertical lines at a and b, and the x-axis.

Symbolically, this is shown by

$$\text{area} = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{b-a}{n} f(x_i)$$

Suppose, instead of taking the inscribed rectangles, the circumscribed rectangles were chosen (Figure 16).

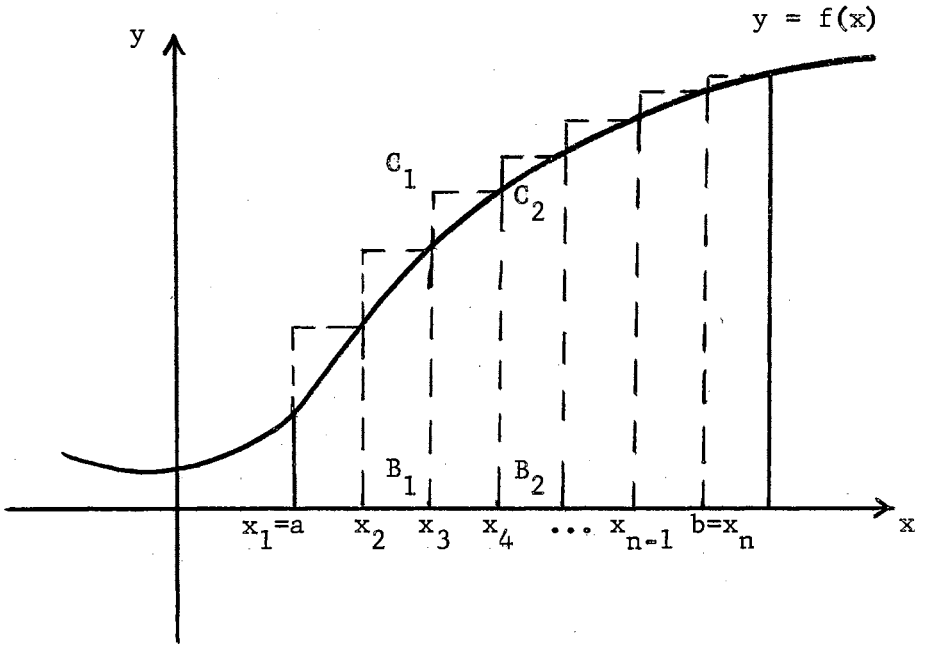


Figure 16.

Then the area of the rectangle $B_1C_1C_2B_2$ would be

$$\text{area } B_1C_1C_2B_2 = \frac{b-a}{n} f(x_4)$$

The approximation of the total region R then would be

$$\sum_{i=2}^{n+1} \frac{b-a}{n} f(x_i)$$

Again, as n gets larger in value, the approximation should approach the true area of the region. As with the inscribed rectangles, it is possible to prove mathematically that the above summation associated with the circumscribed rectangles will have as a limit the ture area of the region. This can be shown as follows:

area of inscribed rectangles | area of region R | area of circumscribed rectangles

or $\sum_{i=1}^{n} \frac{b-a}{n} f(x_i)$ area of R $\sum_{i=2}^{n+1} \frac{b-a}{n} f(x_i)$

Since the limit of either the sum of the inscribed rectangles or the circumscribed rectangles will be the ture area of the region R, it doesn't matter which is used to find the desired approximation. Notice, the only difference in the two summation is for the inscribed rectangles the functional values for the left endpoints of the subintervals were used whereas for the circumscribed rectangles, the right endpoints of the subintervals were used. This is only true when the function f(x) is what is called an increasing function in the interval from a to b. By increasing, we mean

$$f(x_i) \leq f(x_{i+1}) \text{ when } x_i \leq x_{i+1} \text{ for all } x_i \text{ in the interval.}$$

Similarly, a function is decreasing when

$$f(x_i) \geq f(x_{i+1}) \text{ when } x_i \geq x_{i+1} \text{ for all } x_i \text{ in the interval.}$$

Again, mathematically, it can be shown that the function doesn't have to be either an increasing or a decreasing function throughout

the interval, either the left endpoints of the subintervals can be used or the right.

To illustrate, consider the following curve in Figure 17.

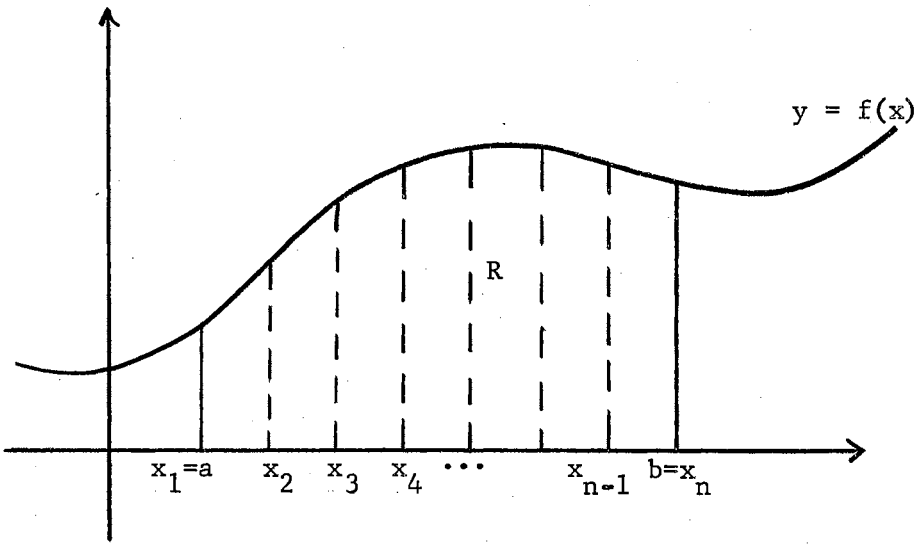


Figure 17.

Then either of the two following expressions would give an approximation:

$$\sum_{i=1}^{n} \frac{b-a}{n} f(x_i)$$

or

$$\sum_{i=2}^{n+1} \frac{b-a}{n} f(x_i)$$

If the limit of the above expression exist, then from the

discussion above

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{b-a}{n} f(x_i) = \text{area of region R} = \lim_{n\to\infty} \sum_{i=2}^{n+1} \frac{b-a}{n} f(x_i)$$

Sample Problem 14: Find an approximation to the area bound between the curve of the function

$$f(x) = x^2$$

from 0 to 2 on the x-axis and the x-axis (Figure 18).

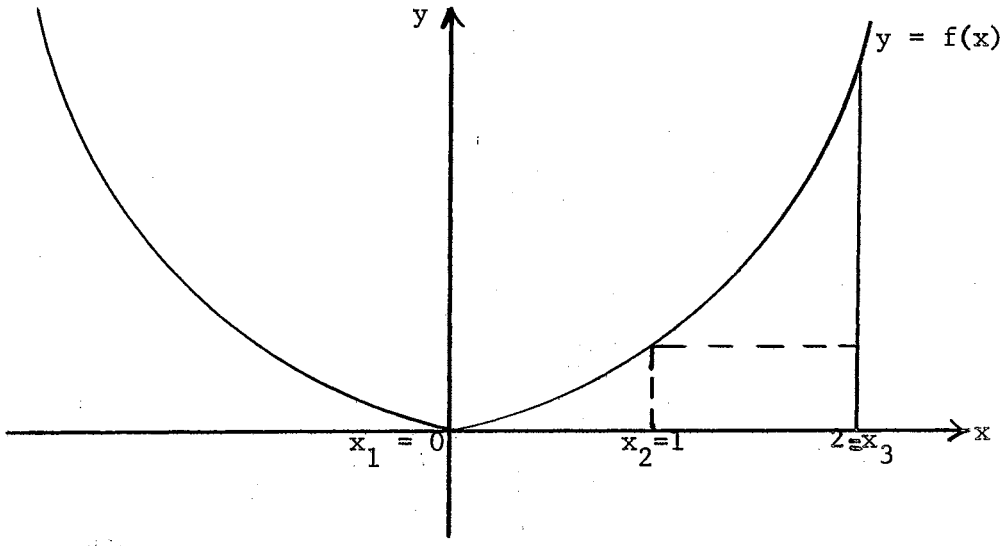


Figure 18.

Solution: First, divide the interval from 0 to 2 into two subintervals as shown in Figure 18. Using the left endpoints of the subintervals, an approximation is given by

$$\sum_{i=1}^{n} \frac{b-a}{n} f(x_i)$$

or, substituting n = 2, a = 0, and b = 2 we have

$$\text{area} = \sum_{i=1}^{n} \frac{2 - 0}{2} x_i^2 = 1 \cdot 0^2 + 1 \cdot 1^2 = 1$$

Hence, 1 square unit would be an approximation to the desired area.

To improve on this approximation, increase the value of h, say to 2n. Then Figure 18 would become as shown in Figure 19.

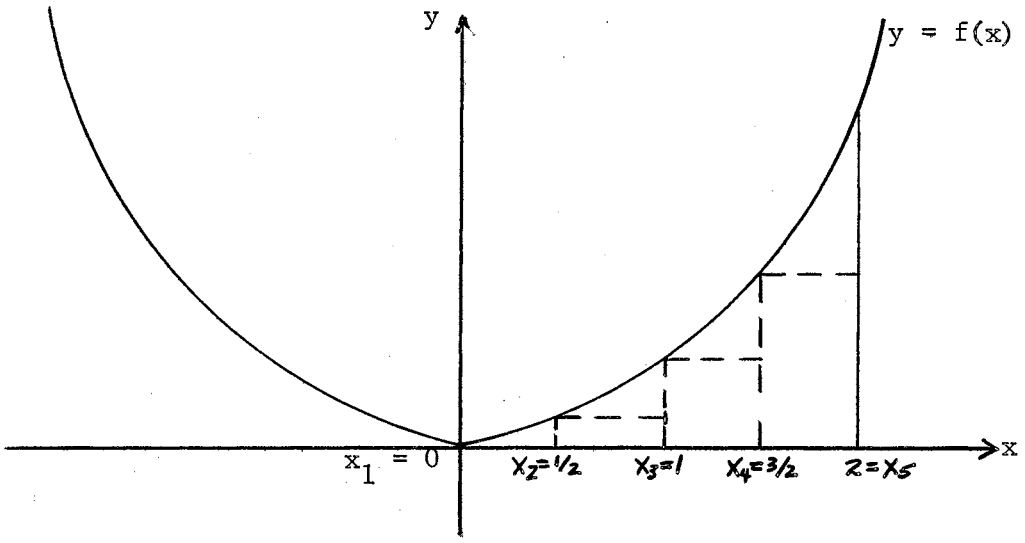


Figure 19.

The improved approximation would then be given by

$$\sum_{i=1}^{n} \frac{b - a}{n} f(x_i = \sum_{i=1}^{4} \frac{2 - 0}{4} x_i^2$$

$$= (1/2)\, x_1^2 + (1/2)\, x_2^2 + (1/2)\, x_3^2 + (1/2)\, x_4^2$$

$$= (1/2)(0)^2 + (1/2)(1/2)^2 + (1/2)(1)^2 + (1/2)(3/2)^2$$

$$= 0 + (1/2)\cdot(1/4) + (1/2)\cdot 1 + (1/2)\cdot(9/4)$$

$$= (1/8) + (1/2) + (9/8)$$

$$= (7/4)$$

Hence, the new approximation is

7/4 or 1.75 square units.

Again, increasing n by doubling this value, we obtain a closer approximation (Figure 20).

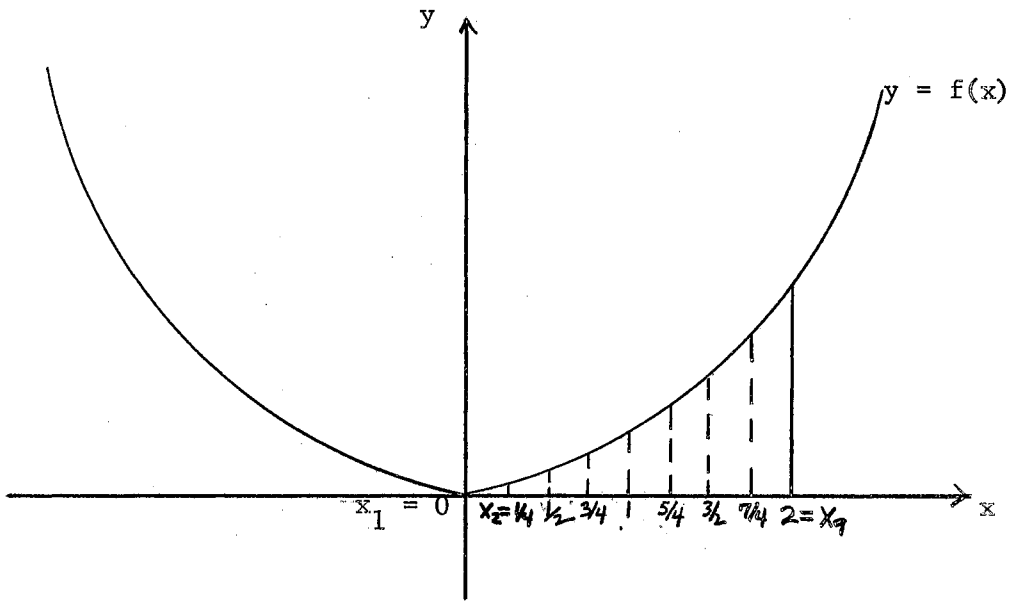


Figure 20.

$$\sum_{i=1}^{n} \frac{b - a}{n} f(x_i = \sum_{i=1}^{n} \frac{2 - 0}{8} x_i^2$$

$$= (1/4)\ x_1^2 + (1/4)\ x_2^2 + (1/4)\ x_2^2 + (1/4)\ x_4^2$$

$$+ (1/4)\ x_5^2 + (1/4)\ x_6^2 + (1/4)\ x_7^2 + (1/4)\ x_8^2$$

$$= (1/4)(0)^2 + (1/4)(1/4)^2 + (1/4)(1/2)^2$$

$$+ (1/4)(3/4)^2 + (1/4)(1)^2 + (1/4)(5/4)^2$$

$$+ (1/4)(3/2)^2 + (1/4)(7/4)^2$$

$$= (1/64) + (1/16) + (9/64) + (1/4) + (25/64)$$

$$+ (9/16) + (49/64)$$

$$= 140/64 \text{ or } 35/16 = 2.1875 \text{ square units}$$

The value of n can be increased until the desired accuracy is obtained. (The true area in this problem is 8/3.)

When using the above technique for finding the area under a curve, the result will be the algebraic value of the area. The reason for this is due to the fact when the curve lies below the x-axis, the area value will be negative since the value of the function is negative.

The limit concept was used to compute an area A. However, the principal concept presented in this section was the <u>limit of a sum</u>. The idea of a limit of a sum occurs in many forms throughout advanced mathematics and is formally defined as "integral calculus." Hence, the calculation of an area under a curve is but one of the many applications of integral calculus.

Exercises 18.

18.1 Find an approximation to the area bounded by the curve of the function $f(x) = x$, the x-axis from $x = 0$ to $x = 3$ by the use of inscribed rectangles discussed in this section. (Compare your approximation when $n = 6$ with the true value)

18.2 Find an approximation to the area bounded by the curve of the function $f(x) = 1 + x^2$, the x-axis from $x = 0$ to $x = 2$ by the use of circumscribed rectangles. (Use $n = 8$)

18.3 Write a program in FORTRAN IV to perform the following tasks:

a. Read in values for a real number a and a real number b ($a < b$) from a data card.

b. Find the area bounded by the curve of the function $f(x) = x^3 - 2x^2 + 1$ from $x = 1$ to $x = 4$ and the x-axis.

c. The program should be so written that n is initially taken as 1, then double n each time and iterate until two successive approximations differ by less than .001.

d. Write out and label approximately on the on-line printer the approximation to the area retaining the maximum number of digits.

e. Write out and label appropriately the value of n upon termination.

Note: The program should be so written that it will be able to approximate the area under any curve by changing only the arithmetic statements containing the particular function.

Topics From Numerical Analysis

Real Roots of an Equation

A very important problem in mathematics is that of finding the real roots of an equation or sometimes called the zeros of a function. That is, given an equation $f(x) = 0$, what value or values of x will make this a true equality. In terms of a function $y = f(x)$, the zeros of the function are those values of x such that the functional values are zero.

There are many numerical methods available to approximate the roots of an equation, but this study will only cover two such methods.

All numerical methods require a rough approximation to a root and then using the particular method, determine a sequence of successive approximations which hopefully will converge to the desired roots. These sequence of successive approximations are found by repeating the

set of operations over and over. Hence, the computer is just the aid needed to take the drudgery out of such a process.

One of the simplest and most common methods for locating a rough approximation for the real root of an equation $f(x) = 0$ is to make a graph of the function $y = f(x)$ and observe where the curve intersects the x-axis. Since the value of the function will be zero at the value of x where the curve intersects the x-axis, this value of x would be a zero of the function or a root of the equation $f(x) = 0$.

The first method to be discussed is that called the "binary chopping" method or sometimes referred to as the "successive bisection" method. This method is perhaps one of the simplest methods, yet will always find a real root if one exist.

Consider the equation $f(x) = 0$. Suppose Figure 21 is the graph of $y = f(x)$.

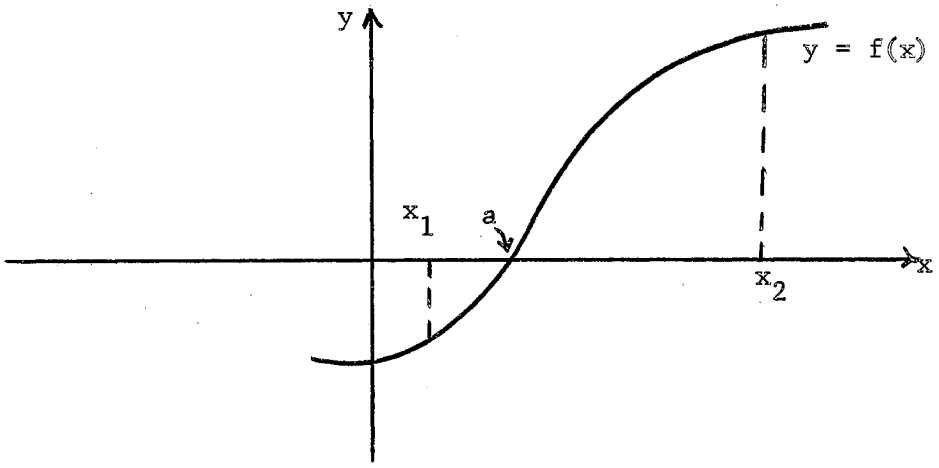


Figure 21.

Suppose x_1 and x_2 are values of x such that $f(x_1) \cdot f(x_2) < 0$. From this, it can be concluded that there is some value of x, say a, where $x_1 \leq a \leq x_2$ such that $f(a) = 0$. Hence, a would then be a root of the equation $f(x) = 0$.

The problem is then to determine the value of a or an approximation to a, say x_i, to some desired accuracy δ. By this it is meant that $|f(x_i) - f(a)| < \delta$, or $|f(x_i)| < \delta$.

From the values of x_1 and x_2 a third value of x is determined, say x_3, where $x_3 = 1/2(x_1 + x_2)$. Geometrically this is the midpoint of the segment x_1x_2, shown by Figure 22.

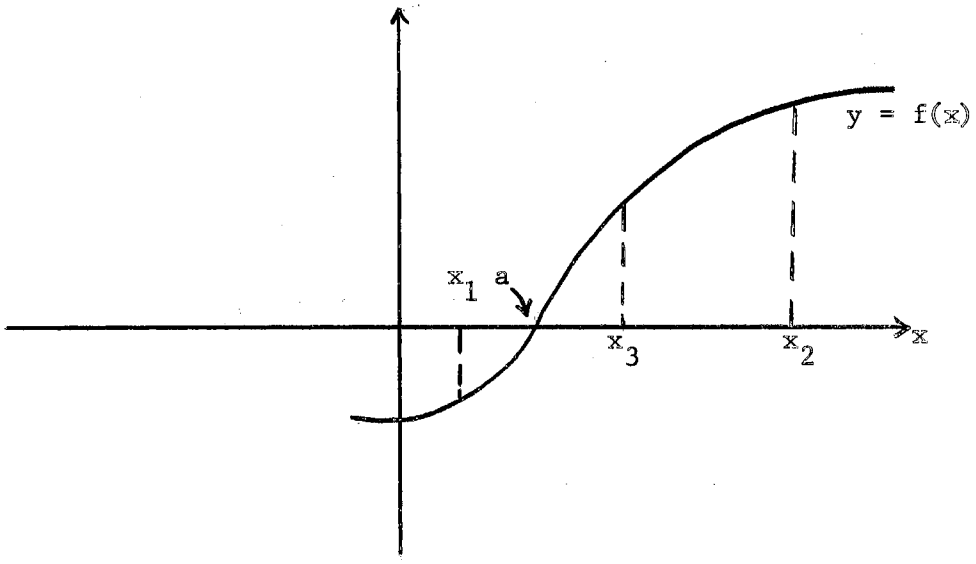


Figure 22.

Then x_3 will replace x_i if $f(x_3)$ and $f(x_i)$ have the same sign for

$i = 1$ or 2. i.e., if $f(x_2) \cdot f(x_3) > 0$ then x_3 replaces x_2. The same procedure is used to determine a new value for x_3. This is continued until x_3 is an approximation to the real root a such that $|f(x_3)| < \delta$ for some predetermined value of δ. Geometrically, it is obvious that if the above method is continued, the new values must converge to the real root a.

Sample Problem: Given the equation $x^3 - 3 = 0$, find an approximation to the real roots correct to 0.1, if any exist, by the succession bisection method. (Note: Use 3 decimal arithmetic)

Solution: Graphing the function $y = x^3 - 3$ we obtain the curve in Figure 23.

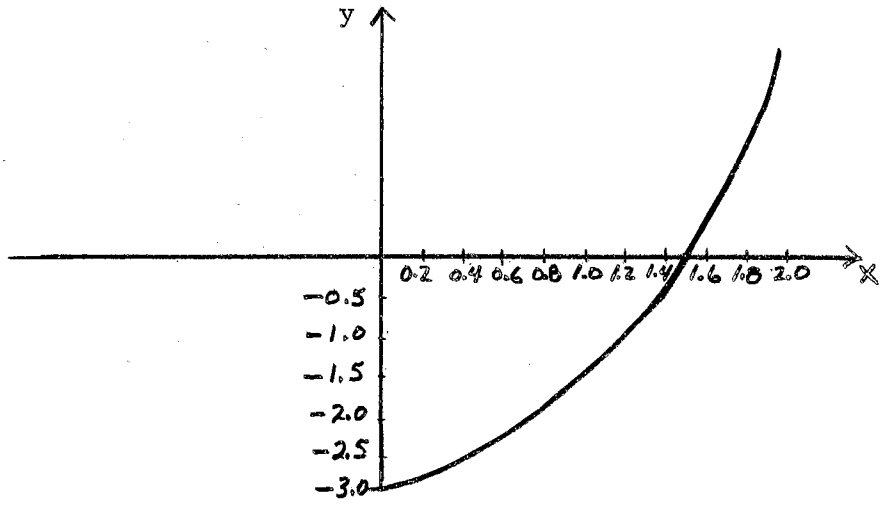


Figure 23.

From the graph, it appears there is a real root between 1.4 and 1.6. Hence, let $x_1 = 1.4$ and $x_2 = 1.6$. Now

$$x_3 = 1/2(x_1 + x_2)$$

or $$= 1/2(1.4 + 1.6)$$

$= 1/2(3.0)$

$= 1.500$

Since $f(1.5) = 1.5^3 - 3$

$= 3.375 - 3$

$= 0.375$

Since $f(1.5) > 0$, x_3 replaces x_2, i.e., $x_2 = 1.5$ which gives

$x_3 = 1/2(x_1 + x_2)$

$= 1/2(1.4 + 1.5)$

$= 1/2(2.9)$

$= 1.450$

$f(1.45) = 1.45^3 - 3$

$= 2.944 - 3$

$= -.056$

Since $|f(1.45)| < 0.1$, the desired accuracy, our approximation to the root is 1.45.

The value of δ used in this Sample Problem is arbitrary. If a closer approximation is needed, the above procedure can be continued until the desired accuracy is obtained.

Notice that the same steps are used in obtaining a new approximation to the root a or x_3. Hence, the computer is again just the tool that is needed to eliminate the trudgery and time in determining the approximation.

Exercises 19.

19.1 Using the method of "successive bisection", find an approximation to the positive real root of the equation $x^2 - 5 = 0$ correct to 0.01. i.e., until $|f(x_3)| < 0.01$.

19.2 Write a program in FORTRAN IV to find an approximation to the positive real roots, if any exist, of the equation

$$x^3 - 2x - 5 = 0$$

by the successive bisection method. The program should be so written that starting with $x_1 = 0$ and $x_2 = 1$, a test is made to determine if $f(x_1)f(x_2) < 0$. If not, use $x_1 = 1$ and $x_2 = 2$ then check again to see if $f(x_1)f(x_2) < 0$. The program should continue until two consecutive integers are found (up to 20) such that the condition $f(x_1)f(x_2) < 0$ exist. The program should then find the approximation to the real root between these two integers by the successive bisection method until a value of x_3 is found such that $|f(x_3)| < .001$. Note: The program should be able to find any positive real root of any equation by replacing the arithmetic statement used to evaluate the particular functional value.

Method of False Position

Another well known method to find a real root of an equation is that known as the method of false position.

Again, suppose there exist values for x_1 and x_2 such that $f(x_1)f(x_2) < 0$. Then there must exist a real root between x_1 and x_2 if f is a continuous function in the interval between x_1 and x_2 such as is shown in Figure 24.

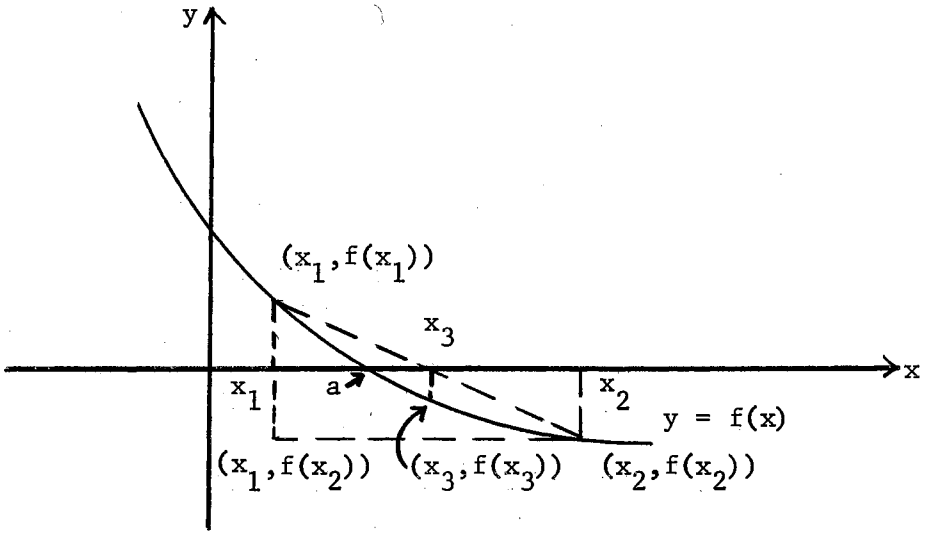


Figure 24.

A third value, x_3, can be found as the x intercept of the line determined by the two points $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Again, as in the previous method, x_3 replaces either x_1 or x_2 which ever function value, $f(x_1)$ or $f(x_2)$ has the same sign as $f(x_3)$. This method can be repeated until an approximation to the root is found to the desired accuracy.

The relationship between x_1, x_2, and x_3 in this method can be determined to be as follows.

Considering the similar triangles of Figure 24, the following is obtained.

$$\frac{x_3 - x_1}{f(x_1)} = \frac{x_2 - x_1}{f(x_1) - f(x_2)}$$

or

$$x_3 = x_1 + \frac{f(x_1)}{f(x_1) - f(x_3)} (x_2 - x_1)$$

Although this expression is a little more difficult to evaluate than the expression used for the successive bisection method, this method generally will converge quicker.

Sample Problem 16: Use the method of false position to approximate the positive root of the equation $x^3 - 3 = 0$, through the second approximation.

Solution: Using the graph of Figure 23, a root can be seen to be between the values $x_1 = 1.4$ and $x_2 = 1.6$. Hence, using the above iterative formula to determine x_3, we have

$$x_3 = 1.4 + \frac{-0.256}{-0.256 - 1.096} (1.6 - 1.4)$$

$$= 1.4 + 0.189\ (0.2)$$

$$= 1.4 + 0.0378$$

$$= 1.4378 \text{ or } 1.438$$

$$f(1.438) = 1.438^3 - 3$$

$$= 2.068$$

Since $f(x_3)$ has same sign as $f(x_2)$, x_3 replaces x_2 so that $x_2 = 1.438$. Another iteration with $x_1 = 1.4$ and $x_2 = 1.438$ gives

$$x_3 = 1.4 + \frac{-0.256}{-0.256 + 0.026}\ (0.032)$$

$$= 1.4 + 1.113\ (0.038)$$

$$= 1.4 + 0.0423$$

$$= 1.4423 \text{ or } 1.442$$

Note: $f(x_3) = f(1.442) = 0.002$. Recall that after two iterations using the successive bisection method, $f(x_3) = 0.056$. Hence, x_3 by the method of false position gives a much closer approximation after two iterations than the successive bisection method.

Exercises 20.

20.1 Using the method of false position, find an approximation to the positive real root of the equation $x^2 - 5 = 0$ correct to 0.01.

20.2 Rewrite the program for Exercise 19.2 using the method of false position instead of successive bisection.

Matrices

In a previous section, the idea of a matrix was introduced. As defined earlier, a matrix is a two-dimensional array or rectangular array of real numbers as shown below:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ \vdots & & & & & \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \cdots & a_{mn} \end{bmatrix}$$

The matrix A above has m rows and n columns. For this reason, the matrix is said to be of order m by n. The numbers a_{ij} which appear in the array are called elements of the matrix, where a_{ij}, indicating

the element in the ith row and jth column. A matrix having only one row or one column such as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \end{bmatrix}$$

or

$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{bmatrix}$$

are called row and column vectors respectively.

Matrix Operations

There are three operations defined on mathrices: addition, scalar multiplication, and multiplication.

Addition of two matrices A and B is defined only for two matrices having the same order. Let a_{ij} be the elements of the matrix A and b_{ij} be the elements of the matrix B, then the sum A + B is the matrix whose elements are $a_{ij} + b_{ij}$ obtained by adding corresponding elements of A and B.

To illustrate, consider the matrices A and B as follows:

$$A = \begin{bmatrix} 2 & -2 & 3 \\ -1 & 4 & 1 \\ 0 & -3 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} -3 & 1 & 4 \\ 2 & -5 & 0 \\ d & -1 & -2 \end{bmatrix}$$

then

$$A + B = \begin{bmatrix} -1 & -1 & 7 \\ 1 & -1 & 1 \\ 3 & -4 & 3 \end{bmatrix}$$

The operation of multiplication of a scalar k and the matrix A is defined to be the matrix obtained from multiplying every element in the matrix A by the scalar k. By a scalar, we mean any real number.

To illustrate, consider the matrix A and scalar k:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & & & \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

then the scalar product is

$$kA = \begin{bmatrix} ka_{11} & ka_{12} & ka_{13} & \cdots & ka_{1n} \\ ka_{21} & ka_{22} & ka_{23} & \cdots & ka_{2n} \\ \vdots & & & & \\ ka_{m1} & ka_{m2} & ka_{m3} & \cdots & ka_{mn} \end{bmatrix}$$

The operation of matrix multiplication is more complicated. For a particular example of matrix multiplication let A be of order 2 by 3 and matrix B of order 3 by 2. Then the product matrix A·B is the matrix

$$A \cdot B = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{31} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

The product matrix is order 2 by 2. In order for the multiplication operation to be defined, the number of columns in the matrix A must be the same number as rows in the matrix B. For some examples, consider the following:

A order of 2 by 3
B order of 3 by 4 — A·B order of 2 by 4.

A order of 3 by 1
B order of 1 by 2 — A·B order of 3 by 2.

A order of 2 by 4
B order of 3 by 2 — A·B undefined.

Notice that matrix multiplication is not always commutative. That is, A·B is not equal to B·A for all A and B.

To illustrate, consider the following:

$$A = \begin{bmatrix} 1 & -2 & 3 \\ 2 & 0 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & -1 \\ 3 & -2 \\ 4 & 1 \end{bmatrix}$$

then

$$A \cdot B = \begin{bmatrix} 8 & 6 \\ 0 & -3 \end{bmatrix}$$

but

$$B \cdot A = \begin{bmatrix} 0 & -4 & 7 \\ -1 & -6 & 11 \\ 6 & -8 & 11 \end{bmatrix}$$

Of course it is possible for the product matrix A·B to be defined and B·A not defined. Such would be the case if A is of order 2 by 3 and B of order 3 by 4. A·B would be of order 2 by 4 and B·A undefined.

Exercises 21.

21.1 Given the two matrices A and B as follows:

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 6 & -2 & -4 \\ 1 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ -5 & 4 \\ 3 & -1 \end{bmatrix} \quad C = \begin{bmatrix} -3 & 0 \\ -2 & 6 \\ 4 & -5 \end{bmatrix}$$

find:

a. kA where scalar k is 2.

b. B + C

c. A·B

d. A + B

e. B·C

21.2 Write a program in FORTRAN IV to perform the following task:

a. Read from a data card a value for m, n, k, l where m and n give the order for a matrix A; k and l give the order for a matrix B. (assume m, n, k, l $\leq$ 10)

b. Read in the elements of the matrix A followed by the elements of the matrix B.

c. Determine A + B if defined. If not defined, print the message "ADDITION NOT DEFINED."

d. Determine A·B if defined. If not defined, print the message "MULTIPLICATION NOT DEFINED."

e. Write out on the on-line printer the matrix A and matrix B, labeled appropriately.

f. Write out the sum A + B (if defined) and label appropriately.

g. Write out the product A·B (if defined) and label appropriately.

h. Halt execution.

Systems of Linear Algebraic Equations

There are many applications to the use of matrices and matrix theory and one application is in the solution of a system of linear algebraic equations. Systems of linear algebraic equations arise in many fields of applied mathematics. While it is theoretically possible to use the well-known direct methods of finding the solution to such a system, it is impractical when the system is of large order. Even with the use of a computer, the problem of round-off error for large systems is a very serious problem due to the large number of operations.

The study will be concerned with one basic method for solving a system of linear algebraic equations, called the Gauss Reduction Method.

Given a set of equations

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n &= b_n
\end{aligned}$$

The Gauss Reduction Method is the process of eliminating one unknown at a time which is the method generally used in introductory

courses in algebra but its application is restricted to a small system. For a large set of equations, the elimination process must be organized into some orderly fashion. This method does primarily that.

The above system could be written in matrix operation as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \cdots a_{1n} \\ a_{21} & a_{22} & a_{23} \cdots a_{2n} \\ \cdot & & \\ \cdot & & \\ \cdot & & \\ a_{n1} & a_{n2} & a_{n3} \cdots a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{bmatrix}$$

Since the coefficients of the variables and the constants are the only values that operations are performed on, to simplify the process of elimination, these values are placed into what is called the augmented matrix of the system. Hence, the augmented matrix of the above system would be

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} \cdots a_{1n} & & b_1 \\ a_{21} & a_{22} & a_{23} \cdots a_{2n} & & b_2 \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ a_{n1} & a_{n2} & a_{n3} \cdots a_{nn} & & b_n \end{array}\right]$$

The Gauss Reduction technique then reduces this augmented matrix to the form

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & a'_{33} & \cdots & a'_{3n} & b'_3 \\ \cdot & & & & & \\ \cdot & & & & & \\ \cdot & & & & & \\ 0 & 0 & 0 & \cdots & a'_{nn} & b'_n \end{array}\right]$$

where all the elements below the diagonal are reduced to zero and the elements shown with a prime are values different from the original augmented matrix.

The first step in the reduction is to replace the entries in the first column by zeros, except for the entry a_{11}. This is accomplished by subtracting the product of the first row and a_{k1}/a_{11} from the kth row for $k = 2, 3, \ldots, n$. The result is that a_{k1} is replaced by 0 for $k = 2, 3, \ldots, n$ while a_{k1} and b_{k2} are replaced by

$$a'_{ki} = a_{ki} - (a_{k1}/a_{11})a_{1i}, \quad b'_k = b_k - (a_{k1}/a_{11})b_1$$

where $k = 2, 3, \ldots, n$ and $i = 1, 2, 3, \ldots, n$.

This is equivalent to the following:

1. Form the ratio a_{21}/a_{11}.
2. Multiply row 1 by the ratio in step 1.
3. Subtract these products from row 2, term by term.
4. Repeat for $a_{31}/a_{11}, \ldots, a_{n1}/a_{11}$ and for rows 3, 4, ... , n in steps 2 and 3, respectively.

After the above steps the augmented matrix is reduced to

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \cdots a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} \cdots a'_{2n} & b'_2 \\ 0 & a'_{32} & a'_{33} \cdots a'_{3n} & b'_3 \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ 0 & a'_{n2} & a'_{n3} \cdots a'_{nn} & b'_n \end{bmatrix}$$

The process described above is carried out for the above excluding the first row and the first column. The formulas used above become

$$a''_{ki} = a'_{ki} - (a'_{k2}/a'_{22})\, a'_{2i}, \quad b''_k = b'_k - (a'_{k2}/a'_{22})\, b'_2.$$

where $k = 3, 4, \ldots, n$ and $i = 2, 3, \ldots, n$.

After the above process, the augmented matrix is reduced to

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \cdots a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} \cdots a'_{2n} & b'_2 \\ 0 & 0 & a''_{33} \cdots a''_{3n} & b''_3 \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ 0 & 0 & a''_{n3} \cdots a''_{nn} & b''_n \end{bmatrix}$$

This process is repeated until all terms below the main diagonal are zero.

Then the set of equations with the reduced augmented matrix written in matrix operation notation becomes

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ 0 & 0 & a'_{33} & \cdots & a'_{3n} \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ 0 & 0 & 0 & \cdots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ \cdot \\ \cdot \\ \cdot \\ b'_n \end{bmatrix}$$

The last step in the process is that called a 'back substitution' process which gives

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b''_2 \\ b''_3 \\ \cdot \\ \cdot \\ \cdot \\ b''_n \end{bmatrix}$$

The formulas to give the last reduction are

$$a_n = b'_n / a'_{nn}$$

and this value is substituted into the next to the last equation in the set giving

$$x_{n-1} = \frac{b'_{n-1} - a'_{n-1,n} x_n}{a'_{n-1,n-1}}$$

or in general

$$x_k = \frac{b'_k - a'_{kn} x_n - a'_{k,n-1} x_{n-1} - \cdots - a'_{k,k+1} x_{k+1}}{a'_{kk}}$$

where $k = n-1, n-2, \ldots, 3, 2, 1$.

A slight variation in this standard form makes the method more straight forward and easier to use. The operations used are primarily

the same as given above by the iterative formulas with the final augmented matrix before the back substitution being as follows:

$$\begin{bmatrix} 1 & a'_{12} & a'_{13} & \cdots & a'_{1n} & b'_1 \\ 0 & 1 & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & 1 & \cdots & a'_{3n} & b'_3 \\ \cdot & & & & & \\ \cdot & & & & & \\ \cdot & & & & & \\ 0 & 0 & 0 & \cdots & 1 & b'_n \end{bmatrix}$$

Then, using back substitution the values of the unknowns can be determined.

The steps to obtain the above augmented matrix is as follows:

1. Divide row one by the element a_{11}.
2. Multiply the first row by a_{21}, subtracting these respective values from the elements in the second row.
3. Continue for rows k = 3, 4, . . . , n.
4. Divide the elements in row two by the diagonal elements a_{22}.
5. Multiply the elements in row two by the element a_{32}, subtracting these respective values from the elements in row three.
6. Continue for rows k = 4, 5, . . . , n.
7. Continue steps 4 and 5 using the respective diagonal elements to create all zeros in the columns below the diagonal elements.
8. Use back substitution to determine the solution.

Note: If the diagonal elements should be zero, search all elements in the same column below the diagonal element to find a non-zero element. Interchange the complete rows so that the diagonal element is non-zero. If a non-zero element cannot be found, then this implies there is no unique solution to the particular system.

Sample Problem 17: Use the Gauss Reduction Method to find an approximation to the solution of the set of algebraic linear equations.

$$2x_1 + x_2 - 3x_3 = 3$$
$$3x_1 + 3x_2 - 2x_3 = 10$$
$$x_1 - 2x_2 + x_3 = -1$$

Solution: Forming the augmented matrix, we obtain

$$\left[\begin{array}{ccc|c} 2 & 1 & -3 & 3 \\ 3 & 3 & -2 & 10 \\ 1 & -2 & 1 & -1 \end{array}\right] \xrightarrow{\text{step 1}} \left[\begin{array}{ccc|c} 1 & 1/2 & -3/2 & 3/2 \\ 3 & 3 & -2 & 10 \\ 1 & -2 & 1 & -1 \end{array}\right]$$

$$\begin{array}{llll} *3 & 3/2 & -9/2 & 9/2 \\ **1 & 1/2 & -3/2 & 3/2 \end{array}$$

*Product of row one by $a_{21} = 3$
**Product of row one by $a_{31} = 1$

$$\xrightarrow{\text{steps 2 and 3}} \left[\begin{array}{ccc|c} 1 & 1/2 & -3/2 & 3/2 \\ 0 & 3/2 & 5/2 & 11/2 \\ 0 & -5/2 & 5/2 & -5/2 \end{array}\right] \xrightarrow{\text{step 4}} \left[\begin{array}{ccc|c} 1 & 1/2 & -3/2 & 3/2 \\ 0 & 1 & 5/3 & 11/3 \\ 0 & -5/2 & 5/2 & -5/2 \end{array}\right]$$

$$*0 \quad -5/2 \quad -25/6 \quad -55/6$$

*Product of row two by $a_{32} = -5/2$

$$\xrightarrow{\text{steps 5 and 6}} \left[\begin{array}{ccc|c} 1 & 1/2 & -3/2 & 3/2 \\ 0 & 1 & 5/2 & 11/2 \\ 0 & 0 & 20/3 & 20/3 \end{array}\right] \xrightarrow{\text{step 7}} \left[\begin{array}{ccc|c} 1 & 1/2 & -3/2 & 3/2 \\ 0 & 1 & 5/3 & 11/3 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

From the last augmented matrix, the value for x_3 can be seen to be 1. Hence $x_3 = 1$. Now substituting this value for x_3 in the second row we have $0 \cdot x_1 + 1 \cdot x_2 + 5/3 \cdot 1 = 11/3$, or solving for x_2 we obtain $x_2 = 2$. Again, substituting the values obtained for x_2 and x_3 into

the first row, we have $1 \cdot x_1 + 1/2 \cdot 2 - 3/2 \cdot 1 = 3/2$ or solving for x_1 we obtain $x_1 = 2$. Hence the solution is $x_1 = 2$, $x_2 = 2$, and $x_3 = 1$. Substituting this into matrix operation form we have

$$\begin{bmatrix} 2 & 1 & -3 \\ 3 & 3 & -2 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ -1 \end{bmatrix}$$

or

$$\begin{bmatrix} 2 & 1 & -3 \\ 3 & 3 & -2 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ -1 \end{bmatrix}$$

Since all arithmetic used in the above solution was in fractional form, no round-off error occurred. If we had used decimal form, retaining one decimal place to the right of the decimal point there would have been some round-off occurring. Hence, if the computer was used in solving a system of algebraic equations, there would be a round-off error occurring. However, for small systems, this round-off error is generally not significant. But for large systems, this can be a very serious problem since the method of finding the solution would require a large number of operations. When using a computer, it would be well to create what is called the error vector or matrix. This is obtained by finding the matrix product of the coefficient matrix and the solution vector and subtracting this from the n by 1 matrix of the constants.

In matrix notation this is given as follows:

Given the system

$$A \cdot X = B$$

the error vector or matrix E would be given by

$$E = B - A \cdot X^*$$

where X^* is an approximation to the solution.

Exercises 22.

22.1 Find the solution, if it exist, of the following systems of algebraic linear equations by the Gauss Reduction Method:

a. $4x_1 + x_2 + 2x_3 = 16$
$x_1 + 3x_2 + x_3 = 10$
$x_1 + 2x_2 + 5x_3 = 12$

b. $x_1 + 2x_2 + 3x_3 = -5$
$3x_1 + 5x_2 - x_3 = 6$
$2x_1 + 4x_2 + 6x_3 = -10$

22.2 Use two decimal arithmetic and the Gauss Reduction Method to find an approximation to the solution of the following system. Determine the error vector or matrix for the solution obtained.

$$6x_1 - x_2 - 3x_3 = 100$$
$$-x_1 + 4x_2 = -50$$
$$3x_1 + 5x_3 = 75$$

22.3 Write a program in FORTRAN IV to solve any nxn system of algebraic linear equations by the Gauss Reduction Method ($n \leq 20$). The program should be written so that the value n is read from a data card followed by the coefficients and constants of each system. The solution should be printed out labeled appropriately along with the error vector or matrix. Assume the diagonal elements will be all non-zero. (Allowance for zero elements in the diagonal can be made, but makes the program much more difficult.)
Hint: It will be advantageous if the constants $b_1, b_2, \ldots, b_n$ are represented as part of the coefficient matrix A. i.e., b_1 can be represented as $a_{1,n+1}$, b_2 as $a_{2,n+1}$, . . . , b_n as $a_{n,n+1}$.

Equipment Alternatives for Program Implementation

Some computer-oriented topics can be taught without the use of a computer. However, it is imperative that students have access to a computer if maximum benefit is to be obtained from a computer-oriented mathematics program such as proposed in this study.

Unknown to many educators, there are computer systems available

at reasonable costs that would provide the minimum requirements for a computer aided instruction program.

There are primarily two areas in the public school's program that could make use of a computer system. That of instruction and in administrative functions. There are certain computer systems that would perhaps serve one of the two areas without providing the necessary hardware to serve the other. This study will consider four basic computer systems, some of which would serve only one or both the above areas. These four systems are:

(1) Courier service to a computer center.

(2) Transportation of students to a computer center.

(3) Remote terminals at the school that permits direct control of a computer located elsewhere.

(4) Acquisition of a small scale computer by the school.

These four types of computer systems will be described in detail and an estimated cost of each system to a school will be given along with the advantages and disadvantages of each.

(1) A courier service provides an efficient and low-cost method of processing student programs. Programs are mailed or sent by messenger to a computer center for execution and the results are returned the same way. Turnaround time would probably range from a few hours to a few days, depending on the distance between the school and the computer center, and the method of transportation used for transport0 ing the programs. If the computer center used was a center serving a large metropolitan school district having several schools, a courier service could be used that should provide a few hours turnaround. A turnaround time of 24 hours would provide an adequate turnaround time for a program of the type proposed by this study.

The availability of a computer center capable of providing computer time should be very good except for schools in remote rural areas. There are many commercial computer centers, industrial companies or universities that are willing to provide such services.

Cost:

Many industrial computer centers and universities now provide free computer services to schools in their areas. If charges are made for computer services, they will be the same as for any user locally. These charges will depend on the particular computer system in use. These charges will generally range from $100 per hour to $200 per hour of Central Processing Unit (CPU) time. The greater charge per hour doesn't always imply the total cost for computing will be the greatest. This is because the time required will depend on the speed of the particular system. i.e., a system having a charge of $200 per hour might be four times faster than the system having a charge of $100 per hour.

A class of 25 students using the computer in a course such as is proposed in this study would use approximately one hour of CPU time a month. Hence, the computer cost for the suggested course of study would be approximately $200 per month plus the cost of transporting the programs to and from the center.

In addition, a keypunch machine would be required at the school which would be at a cost of approximately $60 per month. It would not be absolutely necessary for a school to have a keypunch available to the students since this service could be obtained at the servinc computer center at a cost perhaps less than the cost of the monthly

charge for a keypunch machine.

Disadvantages:

The main disadvantage would be the delay in running the programs. Also, such a method of operation would not serve the administrative needs of a school except for some very limited functions.

Advantages:

The main advantage would be the low cost. There would be no capital investment or fixed obligation required by the school. Another very important, but many times over looked, advantage would be the professional assistance available at the serving computer center.

(2) Transportation of students to a computer center would allow the students the opportunity to physically submit their programs and at some installations be able to have "hands-on" the computer to run their programs. Most computer centers will give a secondary school special turnaround since the time a school could have the students at a computer center would be limited. Again, if a central computer system was used to serve several schools in a large metropolitan area, such a method of operation could provide the schools the computer on a scheduled basis during the school day.

Costs:

The costs of a program where the students are transported to the computer facility would be the same as for the courier service program. The difference would be the cost of transportation of the students over the cost of transporting the programs. Since students would have

access to keypunch machines at the computer center, it would not be necessary to have a keypunch at the school. This would result in a $60 per month savings. Hence, the cost of this program would be just the cost of computer time used plus the cost of transportation of the students.

Disadvantages:

The time required to transport the students to the computer center. Unless the computer center is close to the school, the distance and time could be prohibitive.

Advantages:

Being physically in a computer center and observing the activities would have a high motivating effect on the students.

(3) A remote terminal system provides the remote user with a means of submitting via communications lines of some type, programs that are to be processed on a central computer facility and send the output back to the user.

This study will restrict attention to IBM hardware. However, other manufacturers have similar hardware available at costs comparable to that given in this study.

There are several remote terminal systems available, but this study will describe three types that would support the computer assisted instructional program proposed by this study. The three types are as follows with a description of each:

a. Time-sharing systems using remote teletypewriter terminals such as the IBM 2741. (Figure 25)

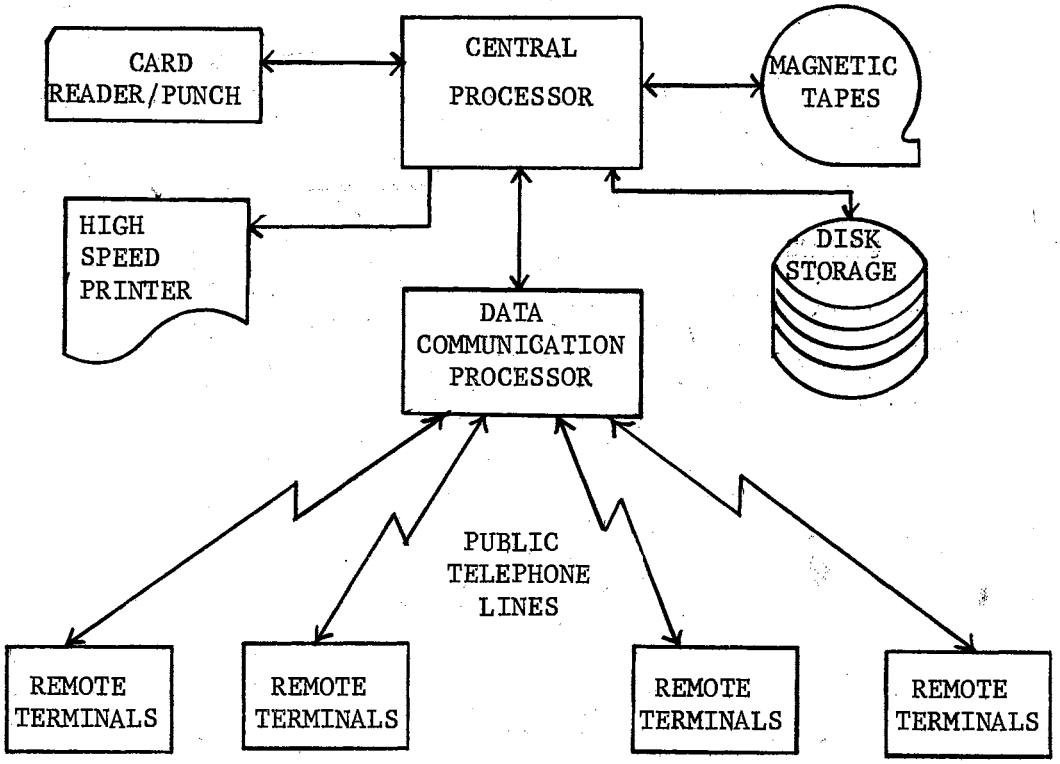


Figure 25. Time-Sharing System

Time-sharing provides a means of extending a computer's capability to a large number of simultaneous users. Such a system shares the computer between all of the users (terminals) of the system at a given time, giving each the impression that he is the only user of the system. A teletypewriter terminal, relatively inexpensive, allows the user to communicate with the central computer system over regular public telephone lines.

To connect a terminal to the time-sharing system, the user dials the telephone number of the system as if making a regular call. The computer will answer the phone and respond with a high-pitched tone indicating that connection has been established. The time-sharing

system checks to see if the calling terminal is valid and the user number is an acceptable one for the particular terminal. This prevents unauthorized users from using the system from an authorized terminal. Most time-sharing systems are conversant in many languages including FORTRAN IV.

Submission of a new program is accomplished by typing it in one line at a time, with each line containing a line number as the first few characters. Consecutive lines are numbered in ascending order with interval of 5 or 10 between line numbers to facilitate the later addition of changes or corrections between statements without having to remember all succeeding lines.

Each line consists of a single statement in whatever language is being used. When the entire program is submitted, the user can give the computer instructions to execute the program. Before actual execution, the program is compiled to produce the actual machine language which is placed in a queue on the direct access disk storage unit to be executed in turn. All users or programs are allotted small blocks of time (few seconds). By this allocation of time, each user experiences minimum delays while waiting for his output. The output is received back on teletypewriter terminal at a rate from 10 to 15 characters per second. This is slow compared to a high speed on-line printer, but for the type programs generally processed from a terminal, the speed of printing the output is not critical.

It is possible to store a program in some type of mass storage device such as a direct access disk unit. Then, the user can recall this program at any time, perhaps change the data, and get the requested output. This would save the time required to type the program, a

line at a time, into the central facility each time the user wanted to use a program.

Costs:

The cost of a time-sharing terminal system will depend on the distance from the terminal to the central computer facility. Also, the charges for various commercial time-sharing systems vary considerable, but below is a representative cost of such a system.

•Terminal (includes maintanance)	$85/month
•Cost of communication equipment (two data sets plus control unit)	100/month
•Monthly cost of telephone transmission	$3/month/air mile
•Central Processing Unit (CPU) $200/hour used. Approximate time for one terminal for one month	200/month

Disadvantages:

- •Cost compared to computing time obtained.
- •Limited to certain types of programs. That is, to programs not requiring a large amount of output.
- •Time required to type in new programs.

Advantages:

- •Faster turnaround than by courier service.
- •Low cost compared to acquisition of a computer.
- •Motivating effect of hands-on control of computer.
- •Experience on the use of remote terminals that are used extensively in colleges and industry.

- Would not require professional staff to support and operate facility.
- Access to large library of programs available on a large central computer system.

(b) Time-sharing system using remote typewriter terminals with attached card reader such as the IBM 1050.

This type system is basically the same as described in (a). The main difference is that the IBM 1050 terminal has a card reader capable of reading 150 cards per minute. This would allow a user to prepare his source deck on a keypunch machine prior to connecting the terminal to the time-sharing system by dialing as described in (a). Although the IBM 1050 card reader is slow compared to an on-line card reader, it would be faster than the keying in of information on an IBM 2741 typewriter terminal.

Costs:

• Terminal (includes maintenance)	$240/month
• Cost of communication equipment (two data sets plus control unit)	100/month
• Monthly cost of telephone transmission.	$3/month/air mile
• Central Processing Unit (CPU) @$200 per hour approximate time for one terminal for one month	200/month
• Keypunch machine	60/month

Disadvantages:

- Cost compared to computing time obtained.
- Limited to certain type of programs as with the 2741 terminal.
- Card reader slow compared to on-line card reader.

Advantages:

- Same as for the IBM 2741 terminal.
- Permits preparation of programs on punched cards prior to submitting.

(c) Remote batch processing terminal such as the IBM 2780 Data Transmission Terminal (Figure 26).

The object of this type of system is to provide a means of handling a large number of programs being submitted at a remote terminal via communications lines. Generally, the remote user will prepare his program on punched cards just as he would if he had a computer of his own. This input deck is read approximately at 250 cards per minute and sent to the central computer by communication lines with necessary control cards. The output can either be sent back by communication lines to the line printer which can print approximately 150 lines per minute or retained at the central facility. A remote batch processing terminal will enter a job or program into the regular batch job stream as if the program had been submitted at the central computer facility. If the central computer facility is a large configuration capable of operating with multipartitions, the remote batch processing terminals can be in a batch partition dedicated to such terminals so that jobs or programs submitted from the terminals will only compete against one another instead of all users submitting programs at the central facility.

Costs:

• 2780 Data Transmission Terminal	$1055/month
• Data Adapter Unit	314/month

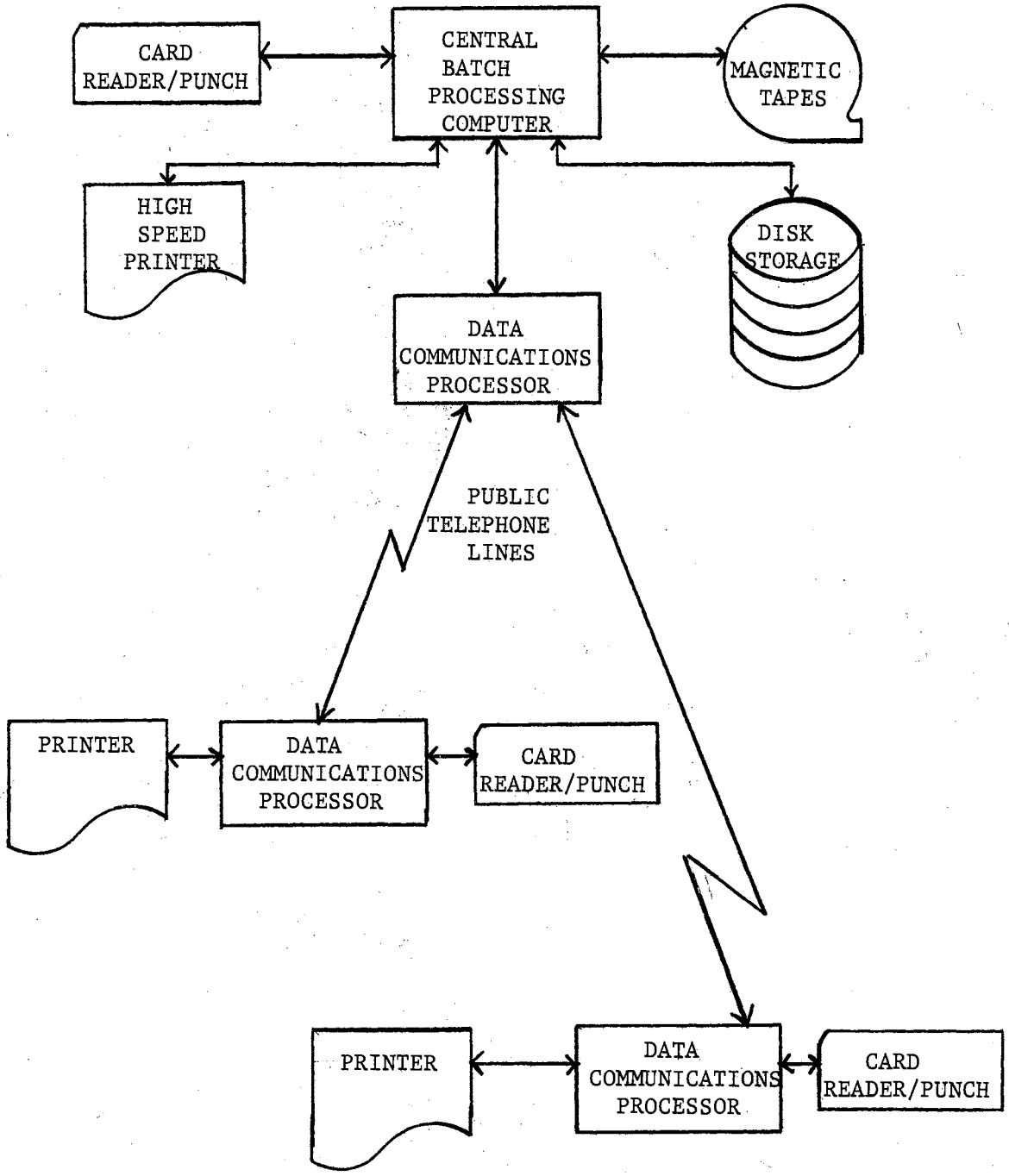


Figure 26. Remote Batch Processing Terminal

•Communication Equipment	344/month
•Monthly cost of telephone transmission	$3.75/month/ air mile
•Central Processing Unit (CPU) @$200 per hour approximate time for one terminal with 50 users	400/month
•Supplies (cards and paper)	100/month
•Keypunch machine	60/month

Disadvantages:

- •Cost of communication equipment.
- •Monthly cost of telephone transmission.
- •Slow reader/punch compared to on-line unit.

Advantages:

- •Faster processing than by teletypewriter terminals.
- •Have access to a large computer system with batch job capabilities at low cost.
- •Would not require professional staff to support and operate system.
- •Motivating effect of hands-on use.

(4) The availability of second-generation computers at a very reasonable cost should make it possible for secondary schools to acquire their own stand-alone computer facility. In the early 1960's many universities installed second-generation computers such as the IBM 1620. These machines proved to be very dependable and reliable which served the needs of the universities until the number of users and the demand for more sophisticated computer equipment required a larger system such as the IBM 7000 series. The IBM 1620 computer proved to be an excellent machine for instruction and in particular

the scientific users. It is easy to learn to use and does not require highly skilled technical personnel to maintain adequate operations such as the more sophisticated systems.

Another small scale computer system available at reasonable cost is the popular IBM 1130 system. This system has many features not available on the IBM 1620, one of which is the capability of being used as a remote terminal such as the IBM 2780 Data Transmission Terminal described in the previous section.

A small scale computer facility at a school would be useful in other ways, particularly in various administrative applications. These might include payroll preparation, accounting, keeping attendance records, keeping records for pupils, and the scheduling of classes.

Although a small scale computer such as the IBM 1620 would not require a highly professional trained person to maintain the system it would be necessary to have a knowledgeable person to direct the computer activities. If the computer is used in connection with the mathematics curriculum as presented in this study, the instructor of the course should be able to provide the necessary direction to develop the computer program. Also, computer manufacturers generally provide assistance to any installation on a regular basis, both with personnel and materials such as manuals and available programs for the particular computer system in use.

A minimum configuration necessary for a computer-oriented course of study such as is proposed in this study would be as follows:

a. Computer Central Processing Unit

b. Console Typewriter Unit

c. Reader/Punch Unit

d. Keypunch Machine

Costs:

•IBM 1620 Computer with card reader/punch and console typewriter (20K core storage)	$633/month
•IBM 1130 Computer with card reader/punch and console typewriter (4K core storage)	900/month
•Keypunch Machine	60/month
•Supplies (cards and console typewriter paper) $1/student per month. Cost for 50 students	50/month
1620 System Total	$743/month
1130 System Total	$1010/month

Disadvantages:

•Space and air conditioning necessary for computer installation.

•Some degree of direction needed.

•Cost per student if used in only one or two courses.

Advantages:

•Computer available at all times at a fixed cost.

•Motivating effect of actual operation of computer.

•Serve both instructional and administrative needs.

CHAPTER V

SUMMARY AND RECOMMENDATIONS

Summary

The advent of the computer and the use of it has had a significant influence on the curriculum at all levels of education. In the past ten years there has been an increasing percentage of secondary school and in particular, college graduates whose careers are intimately connected with high speed computing.

Recent studies show that considerably more than a quarter of the students graduating from college with a major in mathematics are now working in the computing field. Therefore, it is very important that students interested in mathematics or the computing field be introduced to the computer as soon as possible.

In the beginning, computers were generally regarded as a device to solve complicated mathematical problems. Computers are now being used in many nonnumerical tasks such as the simulation of various types of complex systems, storage and retrieval of vast amount of information, and the design of new computers and computer systems.

The potential of the computer in education as a tool or aid in assisting instruction is now being investigated. This particular application is that called "computer-aided instruction." Another very important application of the computer in education is that which might be called "computer-extended instruction." This application consists

of using the computer as an aid in learning various concepts in other subjects such as mathematics. The computer will permit study with more depth, topics that perhaps would be impossible to present without the use of the computer.

This study is intended to provide a suggested computer related course as a part of the mathematics curriculum at the secondary school level that will introduce the computer to potential college bound secondary school students. A subset of the FORTRAN IV language is presented so that the student will have the necessary knowledge to be able to use the computer as a tool or aid in studying some important advanced mathematical topics.

In addition, a description along with the cost, advantages, and disadvantages of various alternate computer systems are given that would enable a secondary school to implement such a course of study as proposed in this study.

Recommendations

The writer makes the following recommendations as a result of this study:

1. The computer should be introduced to secondary school college bound students.

2. The most meaningful method of introducing the computer to secondary school students is in conjunction with related courses, such as mathematics, physics, chemistry, etc.

3. The emphasis in the computer-related courses should be on the use of the computer as a tool or an aid and not attempt to train professional computer programmers.

4. The particular computer system used for implementation of a computer-related course would be determined on the distance the school is from an available computer facility and the available funds for such a program. However, the following systems would be recommended in the following order:

a. Acquisition of a small scale second generation computer system such as the IBM 1130 or IBM 1620 systems.

b. IBM 2780 Data Transmission Terminal capable of handling a batch job stream.

c. IBM 1050 Terminal capable of reading punched cards along with a teletypewriter keyboard.

d. IBM 2741 Teletypewriter Terminal.

e. Transporting students to a computer facility.

f. Courier service where student programs are transported to and from a computer facility.

SELECTED BIBLIOGRAPHY

Anderson, Decima M., Computer Programming-Fortran IV, Appleton-Century-Crofts, New York, N. Y., 1966.

Berkely, Edmund C., The Reference Diary of the Library of Computer and Information Sciences, The Library of Computer and Information Science, 1966.

Carlberg, Mona, Survey of Data Processing Instruction In Six High Schools, The Journal of Business Education, March, 1966.

Carroll, John M., Careers and Opportunities in Computer Science, E. P. Dulton and Co., New York, 1967.

Charp, Sylvia, Computers Solve Math Instruction Problems, Nations Schools, Vol. 78, No. 4, October, 1966.

Dorn, William S., Computers in the High School, Datamation 13, 2 (Feb., 1967).

Filep, Robert T., What We Know, So Far, About Computer-Assisted Instruction, Nations Schools, Vol. 80, No. 4, Oct., 1967.

Ferris, Ruth, "Hands On" Computer Pays Dividend for Altus Schools, Oklahoma Teacher, March, 1966.

Feuche, Michel, Second-Generation Computers Live Again - In The Resale Market, Computers and Automation, Col. 16, No. 9, September, 1967.

Forsythe, Alexander, Math and Computing in High School: A Bethrothal, Mathematics Teacher, Vol. 57, No. 1, January, 1964.

Graham, J. Wesley, Computer, A Revolution in Secondary Education, Computers and Automation 16, 3 (March, 1967).

Gruenberger, Fred, Computing in the Secondary Schools, Datamation, May, 1964.

Harrill, John B., Basic Fortran Programming, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

Hoffman, Albrecht, Atchison, Charp, and Forsythe, Computers for School Mathematics, Mathematics Teacher, Vol. 58, No. 5, May, 1965.

Johnson, Richard and Kiokemeister, Fred, Calculus with Analytic Geometry, Allyn and Bacon, Boston, Massachusetts, 1963.

_____, Computer Yearbook and Directory, American Data Processing, Inc., Detroit, Michigan, 1968.

McCalla, Thomas Richard, Introduction to Numerical Methods and Fortran Programming, Wiley & Sons, New York, N. Y.

Niven, Ivan, Calculus - An Introductory Approach, Nostrand Co., Inc., Princeton, New York, 1966.

Ransom, Charles E., Computers and Education: The IBM Approach, Computers and Automation, March, 1966.

Richardson, Jesse O., Teaching Mathematics Using A Time-Shared Computer System, Computer and Automation, March, 1966.

Schilhab, Elgin, Digital Computing Drops Into The Curriculum Pond, Texas Outlook, Vol. 51, May, 1967.

Taylor, Angus E., Calculus with Analytic Geometry, Prentice Hall, Englewood Cliffs, New Jersey, 1959.

Tonne, Herbert A., Data Processing Instruction In High School, The Journal of Business Education, October, 1964.

Weeg, Gerard P. and Reed, Georgia B., Introduction to Numerical Analysis, Blaisdell Publishing Company, Waltham, Massachusetts, 1966.

Wood, Merle and Espegren, Robert, High School Computer Programming Curriculum, The Journal of Business Education, Dec., 1963.

VITA 3

Robert Duane Gumm

Candidate for the Degree of

Doctor of Education

Thesis: AN ANALYSIS AND DEVELOPMENT OF A COMPUTER SCIENCE PROGRAM FOR USE IN SECONDARY SCHOOL MATHEMATICS

Major Field: Higher Education

Biographical:

Personal Data: Born at Bloom, Kansas, January 11, 1933, the son of Elliott and Lodema Gumm.

Education: Attended elementary school at Kingsdown, Kansas; graduated from Kingsdown High School in 1951; received the Associate of Arts degree from Dodge City Community College, Dodge City, Kansas, 1953; received the Bachelor of Science degree from Fort Hays Kansas State College, Hays, Kansas, 1955, with a major in mathematics; received the Master of Science degree from The University of Kansas, Lawrence, Kansas, 1961, with a major in mathematics; did graduate work in education and mathematics at The University of Colorado, Boulder, Colorado; did graduate work in mathematics and computer science at The University of Missouri at Rolla, Rolla, Missouri; completed requirements for the Doctor of Education degree at Oklahoma State University, Stillwater, Oklahoma, August, 1969.

Professional Experience: Employed by the Lane County Community High School, Dighton, Kansas, as teacher of mathematics from 1955 to 1960; employed by the Jefferson County School District, Lakewood, Colorado, as teacher of mathematics from 1961 to 1963; employed by Fort Hays Kansas State College as Instructor and Assistant Professor of Mathematics from 1963 to 1967; presently employed as Associate Director of the University Computer Center, Oklahoma State University.

Professional Organizations: Association for Computing Machinery and Mathematical Association of America.