digital

edusystem handbook









ADDITIONAL COPIES

Additional copies of this handbook may be purchased for $5.00 per copy. Please send your order to the address below. DEC offers special discounts on quantity orders.

Digital Equipment Corporation
Communications Services, Parker Street
Maynard, Massachusetts 01754

digital

edusystem handbook

prepared
by
small systems technical writing group
programming department
digital equipment corporation

pdp-8 handbook series

FIRST PRINTING, JANUARY 1973

The description and availability of the software products described in this manual are subject to change without notice. The availability or performance of some features of the software products may depend on a specific configuration of equipment. Consequently, DEC makes no claim and shall not be liable for the accuracy of the software products. Distribution of software products shall be in accordance with the then standard policy for each such software product.

The following are registered trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	FOCAL
DECtape	OS/8
Digital	PDP
EduSystem	RSTS

ERROR REPORTING

If you find any errors in this handbook, or if you have any questions or comments concerning the clarity or completeness of this handbook, please direct your remarks to:

Digital Equipment Corporation
Software Information Service, Building 3-5
Maynard, Massachusetts 01754

ADDITIONAL COPIES

Additional copies of this handbook may be purchased for $5.00 per copy. Please send your order to the address below. DEC offers special discounts on quantity orders.

Digital Equipment Corporation
Communications Services, Parker Street
Maynard, Massachusetts 01754

foreword

The computer is an exciting contemporary piece of equipment. This fact alone explains some of the fascination that a computer invariably stirs up among students, but it doesn't explain all of it. The powerful motivational capabilities which the computer demonstrates in every school that installs one can only be explained by the way it is used.

The computer is tireless. Unlike a student, it loves to do complex calculations. Separating theory from calculation in school assignments has always been a problem. Some of the most interesting and challenging concepts of math and science involve, unfortunately, an overwhelming amount of dull calculations. The student gets bogged down in the arithmetic and never gets excited about the idea. Or, worse still, a less important concept is taught because its calculations come out even. The computer, by taking on all calculations and doing them quickly and accurately, opens up new possibilities for classroom study and student interest.

The computer is immediate and unfailingly accurate. It does in seconds what people take minutes or hours to do. The speed of its responses make for powerful reinforcement. It challenges the student to think through the concepts as fast as it grinds through the calculations.

The computer is anonymous. Real learning occurs when a student has an idea and tries it out. At the start, he isn't sure whether the idea is valid. Some students have no fear of being wrong and will tell the class their idea. Others are less willing to risk the ridicule of being wrong. The computer lets all students try out their ideas and gain confidence in them. It treats all students alike; it has no favorites. It waits for the slower user and bounds ahead quickly for the brighter student. With a computer, a student competes with his past achievements, not with other students.

Above all else, the computer is challenging. Why is it that a student who quits on a homework problem after trying it once will work tenaciously to get a program running right? Partly, it's

the immediacy of the computer's response. Partly, it's the ease with which the student can change his program and try again. Partly, it's the fun of talking to a machine and having it respond. But beyond all that, there is something about the close interaction between a user and a fast, willing, logical machine which is tremendously challenging.

In all these ways and more, the computer stimulates the student, stretches his thinking, provides an immediate and pertinent application for skills learned in class. The impact on the teacher can be just as great as the impact on the student. Given motivated students, the dedicated teacher becomes even more dedicated. It is not unusual for a teacher to stay at school until five o'clock to give students more time on the computer.

Computers have other uses around the school than instruction. For example, a computer can easily be programmed to grade tests, thereby saving precious teacher time, as well as providing more immediate feedback to students. Other tedious administrative chores such as attendance reporting, grade reporting, transcripts, and payroll can be performed on the computer.

What will the computer mean in your school? Almost certainly it will mean that students learn and improve at a faster rate. Tests have shown that students who use the computer as part of their math class improve at four times the rate of those who are taught in the traditional way—not just honor students, not using a futuristic curriculum. A cross-section of students—black, white, Chicano, disadvantaged, rich, middle-class—in regular algebra classes improved four times as fast with the computer. The computer also means more thorough understanding coupled with tremendous student motivation. It means more dedicated teachers and an erasing of tedious administrative chores. It means contemporary education for today's world.

The computer does not replace the teacher. Nor does the simple existence of the computer make a poor student into a superior student. But in school after school, the computer is turning the bored, lethargic student into an involved, eager student. We at Digital believe that this is the vital, first step to better education. And we firmly believe that computers are for all kids, not for a few geniuses.

introduction

EDUSYSTEMS—SCHOOL COMPUTERS THAT MEET THE CHALLENGE

It takes more than just hardware to make an effective school computer system. It takes a thoroughly tested combination of system components and instructional materials designed specifically for classroom use. Recognizing this fact, Digital Equipment Corporation has designed EduSystems—a complete line of computer systems tailored to the needs of schools and colleges.

The basis of each EduSystem is a PDP-8/E computer,[1] a terminal, and a BASIC language processor. All EduSystems utilize the well-known computer language BASIC. BASIC programs are simple combinations of English words and decimal numbers. Students with no previous computer experience can be writing meaningful programs after as little as an hour of instruction.

EduSystems are compact, trouble-free, and engineered for use in the busy school environment. Even the largest EduSystem can be installed and used right in the classroom. All systems can run completely unattended (i.e., no operator is necessary). Each EduSystem is designed to handle a large number of student users. Time-sharing allows up to 16 students and teachers to work at the computer simultaneously. Batch processing allows hundreds of student runs per day.

EduSystems are expandable; as school and student needs increase, the configuration of the system may also increase. Starter systems (EduSystems 5, 10, 15, and 20) grow to be Intermediate Systems (EduSystems 25, 30, and 40); Intermediate Systems grow

---

[1] Certain EduSystems are also available on the PDP-11 computer. Write to the Educational Products Group, Building 5-5, Digital Equipment Corporation, Maynard, Mass. 01754, for more information.

to be the Total System (EduSystem 50). The expansion modules can be installed right at the school. Expandable EduSystems are always the right size to meet a school's present demands. There is never the need to start out with too much computer or end up with too little.

As the EduSystem computer expands, so does the BASIC language. Digital's Total Systems[2] offer the most powerful BASIC language processors of any computers in their class. In addition, EduSystem 50 provides time-shared FOCAL and FORTRAN-D language processors, a time-shared Assembly Language package, and system utility programs.

The effectiveness of EduSystems as classroom tools is well proven. Hundreds of schools starting out with EduSystem 10 or EduSystem 20 have since expanded to an Intermediate System, or even a Total System, while many others are continuing to support excellent programs of computer education without expanding their facilities at all.

USING THE EDUSYSTEM HANDBOOK

The *EduSystem Handbook* provides a complete user's guide for each individual EduSystem and a self-instruction course in the use of the BASIC language in general. Most EduSystem users will need to read only two chapters of this handbook: Chapter 1, and the chapter concerning the EduSystem being used.

Chapter 1 is a primer on the BASIC language,[3] allowing the user to teach himself the fundamentals of BASIC and to familiarize himself with the EduSystem terminal keyboard. Many examples and exercises are included to aid the user in discovering the elements of the BASIC language. A user familiar with Chapter 1 can write simple BASIC programs and run them on any EduSystem.

Once the user knows the fundamentals of BASIC, he refers to the chapter (chapters 2 through 9) concerning his particular EduSystem. The individual EduSystem chapters describe the features and extended capabilities of BASIC as it is used on the specific EduSystem. Each chapter also contains detailed operating instructions and error messages for the EduSystem. In addition,

---

[2] EduSystem 80 is a Total System available only on the PDP-11.

[3] Chapter 1 is derived from *Teach Yourself BASIC,* Volumes I and II published by Technica Education Corporation.

each chapter contains a table summarizing the BASIC language capabilities of the EduSystem being described.

Chapter 9 provides a detailed description of EduSystem 50 capabilities, not only of the BASIC language but also of FOCAL, FORTRAN-D, PAL-D (the assembly language), the EduSystem 50 Monitor, and all the system utility programs. The EduSystem 50 user will also find much helpful information in *Introduction to Programming* and *Programming Languages,* Volumes 1 and 2 of the PDP-8 handbook series.

The *EduSystem Handbook* is designed to serve as the primary guide for users of all EduSystems. Users of each EduSystem will find many programming examples to facilitate their understanding of the system. A summary of the BASIC language capabilities of all Digital EduSystems is provided at the end of Chapter 1.

COMMON PROGRAMMING TERMS

Such words as loop, jump, nesting, and array have special meanings to computer programmers. Familiarity with these terms is a prerequisite to learning the more advanced programming languages. The Index/Glossary at the end of this handbook defines many of the commonly used computer programming terms.

contents

CHAPTER 1 TEACH YOURSELF BASIC

CHAPTER 2 EDUSYSTEM 5

CHAPTER 3 EDUSYSTEM 10

CHAPTER 4 EDUSYSTEM 15

CHAPTER 5 EDUSYSTEM 20

CHAPTER 6 EDUSYSTEM 25

CHAPTER 7 EDUSYSTEM 30

CHAPTER 8 EDUSYSTEM 40

CHAPTER 9 EDUSYSTEM 50

APPENDICES

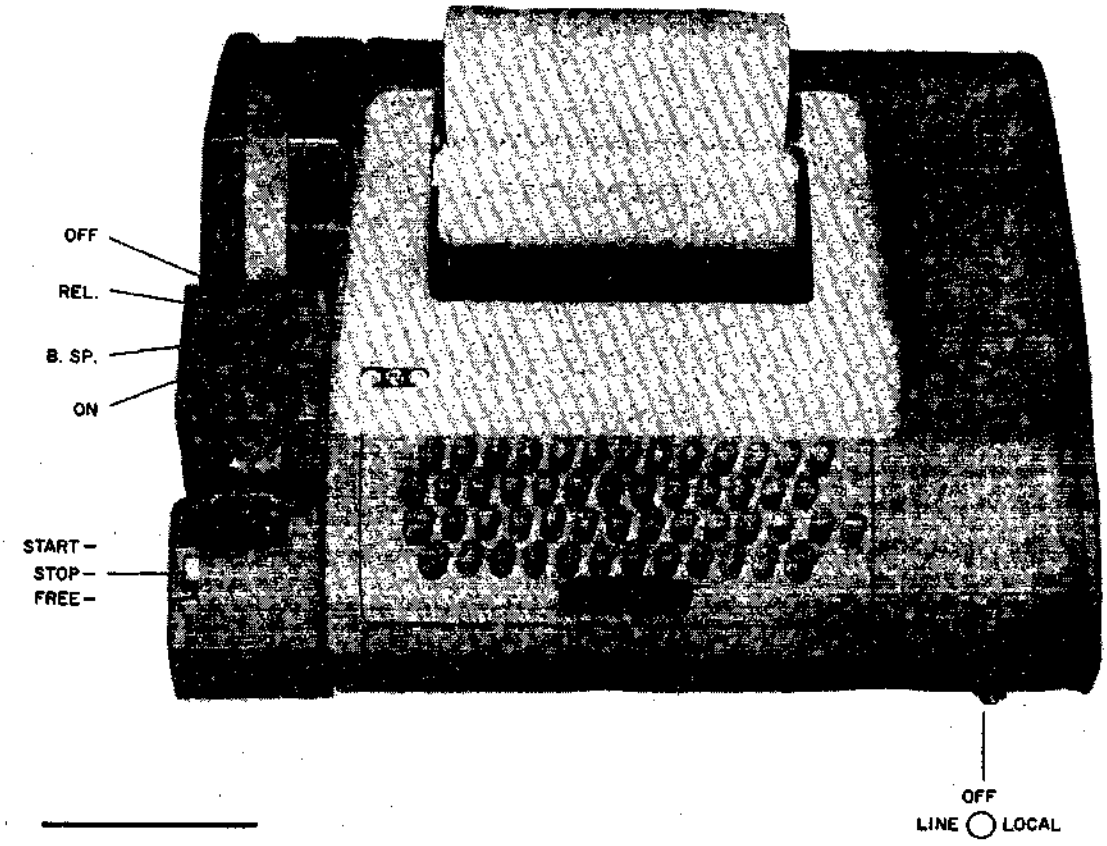
LIST OF TABLES

LIST OF ILLUSTRATIONS

chapter 1
teach yourself basic

INTRODUCTION

BASIC[1] is a conversational computer language which enables a human to carry on a "dialog" with a computer. We will "talk" to the computer by using a Teletype[2] like the one shown below. Your terminal may be an alphanumeric cathode ray tube (CRT) DECterminal or a high-speed DECwriter. Operations with these terminals are, in most cases, identical with the Teletype, so we will only discuss the Teletype here.



[1] BASIC (Beginner's All-purpose Symbolic Instruction Code) is a trademark registered by the Trustees of Dartmouth College.
[2] Teletype is a registered trademark of the Teletype Corporation.

Using the Teletype, we type messages to the computer, requesting it to carry out operations. The computer performs the required operations and prints the results on the same Teletype. If we make certain mistakes or if we ask the computer to do something it cannot do, it may print an error message. For example, if we type

```
DO THE HOMEWORK ON PAGE 257
```

(and press the RETURN key)

```
WHAT?
```

the computer may respond by printing a message such as

WHAT?

The actual response depends on the EduSystem that you are using.

If you wish to use the computer, you must:

Learn what the computer can do and what the computer cannot do.

Learn a language, such as BASIC, so that you can instruct the computer to do things within its capability.

Communcate with the computer by means of the Teletype.

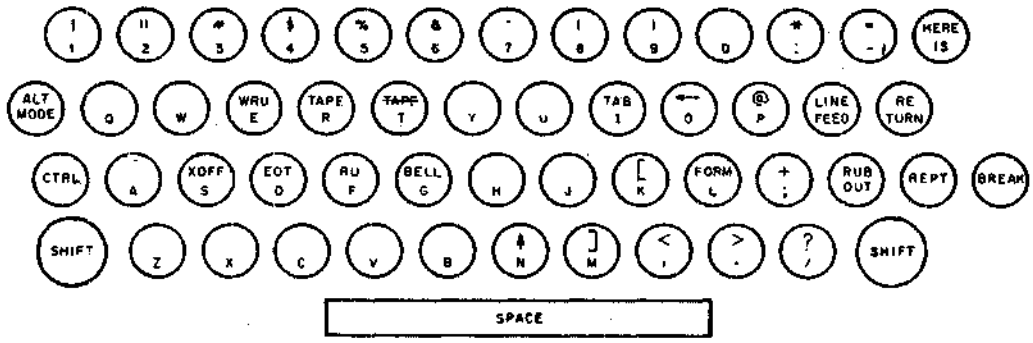
We will begin by assuming that you know little or nothing about computers and will try to lead you through the following four levels of "know-how."

1. You know nothing about computers. If you wish to use a computer to help you solve a problem, you describe the problem to Susan. Susan uses the computer and returns the answer to you.
2. You can operate the computer (Teletype), using a program supplied by another person. The program, however, is gibberish to you—incomprehensible!
3. You can read and understand programs written by others but are unable to write original programs of your own.
4. Computerland is yours! You can invent your own original problem-solving procedures, write them in the BASIC language, check them out on the computer, correct them ("debug" them) if necessary, and obtain the desired results.

This chapter is designed to help you learn the fundamentals of BASIC. Many examples and exercises are included to aid you in discovering the elements of the BASIC language.

Teletype Keyboard

BASIC programs must be written using the symbols that appear on the Teletype keyboard. A diagram of the Teletype keyboard is shown below:



No other symbols may be used. For example, the following symbols are commonly used in mathematics, but may *not* appear in a BASIC program because they are not on the keyboard.

$\alpha \quad \beta \quad \Sigma \quad \int \quad \div \quad \pi \quad \neq \quad \varepsilon \quad \equiv$

On the keyboard diagram, locate the keys with the following symbols:

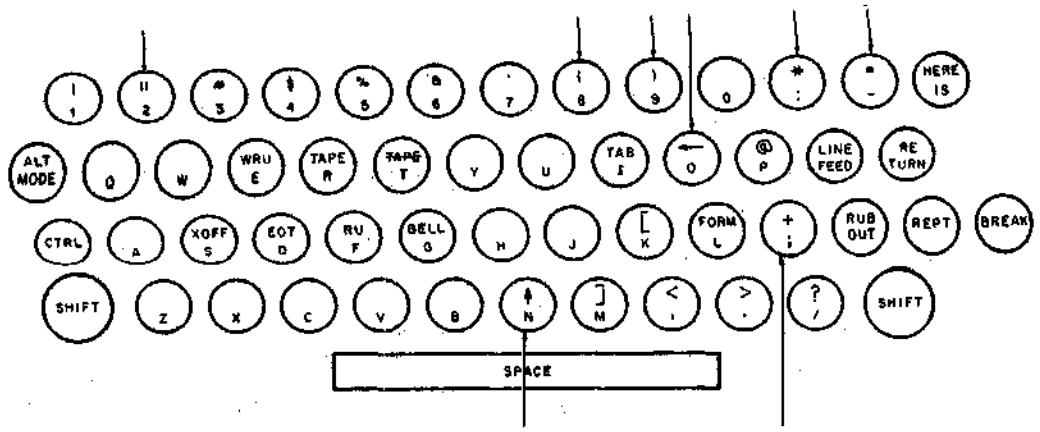
Letters: A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z

Digits: 1 2 3 4 5 6 7 8 9 0

Special: , . / -

To type any of the above symbols, simply press the appropriate key. For example, to type the letter S, press the key on which S appears.

Here is another copy of the keyboard chart so you won't have to turn the page back.



Locate the SHIFT keys. There are two of them, located at the left and right ends of the bottom row of keys. When we refer to a character that is typed while the SHIFT key is held down, we will show it as SHIFT/Character. For example, if we want you to type a ← (back arrow) we will tell you to type SHIFT/O.

We have drawn arrows to call your attention to the keys with the following special characters.

" ( ) * = ← ↑ +

To type any of these characters, you must hold either SHIFT key down and press the key that has the desired character.

The space bar looks like this: 

Use it to insert spaces as you type.

Locate the RETURN key: 

Now you are ready to start "talking" to the computer.

GETTING ACQUAINTED WITH BASIC

Imagine that we are seated at the teletype and that (perhaps with some help) we have attracted the attention of the computer. We will begin with some absurdly simple programs.

> If you try these programs, remember to press the RETURN key at the end of each line that you type. If you make a mistake, the computer may print an error message. Ignore it—retype the line.

Here we go!

```
SCR
```

First, we type SCR and press the RETURN key. The computer SCRatches (erases) any old program in its *memory*.

```
10 PRINT 7
99 END
```

Then we enter our BASIC program, consisting of two *statements*. Each statement is on a separate line. The program is in the computer's memory.

```
RUN
 7
```

We tell the computer to RUN the program. It does and prints the result, 7.

The BASIC *program* is shown again below.

```
10 PRINT 7
99 END
```

It consists of two *statements*, a PRINT statement and an END statement. Each statement begins with a *line number*. Read on—it gets better.

Let's do something a little more exciting.

```
SCR
```

First, we SCRatch the preceding program.

```
10 PRINT 3+4
99 END
```

And then enter a new program. This program also has two statements.

```
RUN
```

We tell the computer to RUN the program.

```
7
```

It does. The result is 7 since 3 + 4 = 7.

```
20 PRINT 3-4
30 PRINT 3*4
40 PRINT 3/4
40 PRINT 3/4
```

Let's *add* three more statements. We use 20, 30 and 40 as line numbers.

```
LIST
```

Then we type LIST and press the RETURN key.

```
10 PRINT 3+4
20 PRINT 3-4
30 PRINT 3*4
40 PRINT 3/4
99 END
```

The computer LISTs the program in its memory. Note that there are *five* statements and that they are listed in *line number* order.

```
RUN
```

O.K., let's RUN the program.

```
 7
-1
 12
 .75
```

Here are the four results, one for each of the first four statements in the program.

Exercise 1. Examine the preceding program, then write the symbol that is used to tell the computer to perform each of the following arithmetic operations.

OPERATION	SYMBOL
Addition	_____
Subtraction	_____
Multiplication	_____
Division	_____

Remember these things.

WE TYPE	TO TELL THE COMPUTER TO
`SCR`	SCRatch (erase) the program in memory.
`RUN`	RUN the program in memory.
`LIST`	LIST the program in memory.

The statement

```
10 PRINT 3+4
```

tells the computer to evaluate the numerical expression 3 + 4 and print the result. In this case, the result is 7.

The statement

```
30 PRINT 3*4
```

tells the computer to evaluate the numerical expression 3*4 (3 times 4) and print the result. This time, the result is 12.

The statement

```
99 END
```

simply marks the end of the program. Every BASIC program must have an END statement.

Each statement begins with a *line number*. A line number may be any counting number in the range:

$$1 \leqslant \text{line number} \leqslant 2046$$

Larger EduSystems permit line numbers to 9999, but 2046 is plenty for most programs.

Instead of numbering statements with consecutive counting numbers (1, 2, 3, etc.) we use 10, 20, 30 and so on. This gives us room to insert a new statement between two old statements. For example, if we had already entered a program using 10, 20, 30, 40 and 99 as line numbers, we could insert a statement between statement 20 and statement 30 by using 25 as the line number of the new statement.

Exercise 2. Do *not* use the computer to do this exercise. Instead, pretend that *you* are the computer. We have entered the following program into your memory and told you to RUN the program. Do it.

```
10 PRINT 12+3
20 PRINT 12-3
30 PRINT 12*3
40 PRINT 12/3
99 END
RUN
```

We entered this program, consisting of five statements, into your memory.

O.K., computer, RUN the program. Indicate your answer by filling in the blanks.

_____

_____

_____

_____

Exercise 3. You are still the compuer. We will add a statement to the program we entered in the preceding exercise.

```
50 PRINT 2*3+4
```

We add this statement.

```
LIST
```

LIST the program in your memory.

_____

_____

_____

_____

_____

_____

```
RUN
```

Now RUN the program.

_____

_____

_____

_____

_____

Numerical Expressions

The computer prints the value of a numerical expression as a decimal numeral. The following table shows examples of computer-printed values of numerical expressions.

EXPRESSION	VALUE	REMARKS	
3.14	3.14	A decimal numeral is a	
−123	−123	decimal numeral	
2 + 3 + 4	9	2 + 3 + 4 = 5 + 4 = 9	
5 − 3 + 4	6	5 − 3 + 4 = 2 + 4 = 6	NOTE 1.
2*3*4	24	2*3*4 = 6*4 = 24	
2*3/4	1.5	2*3/4 = 6/4 = 1.5	
2*3 + 4	10	2*3 + 4 = 6 + 4 = 10	
2 + 3*4	14	2 + 3*4 = 2 + 12 = 14	
35 − 2*3	29	35 − 2*3 = 35 − 6 = 29	NOTE 2.
1/2 + 3	3.5	1/2 + 3 = .5 + 3 = 3.5	
2 + 3/4	2.75	2 + 3/4 = 2 + .75 = 2.75	
24/2*3	36	24/2*3 = 12*3 = 36	
1/(2 + 3)	.2	1/(2 + 3) = 1/5 = .2	
(2 + 3)/4	1.25	(2 + 3)/4 = 5/4 = 1.25	NOTE 3.
24/(2*3)	4	24/(2*3) = 24/6 =4	
1/3	.3333333		
2/3	.6666667	Value rounded to seven	
100/3	33.33333	significant digits.	
200/3	66.66667		

NOTES

1. The operations are done in left to right order.

2. All multiplications and/or divisions are done before *any* additions and/or subtractions. To evaluate 24/2*3 the computer first divides 24 by 2, obtaining 12. Then it multiplies 12 by 3, obtaining 36.

3. We use parentheses to modify the order in which operations are done.

Your turn—but let someone else use the computer while you work the following exercises.

Exercise 4. You are the guest star on the television program "Computer for a Day." To win the grand prize (7 microbucks) you must evaluate each of the following numerical expressions. Go!

(1) 3*4 + 5 _____ (2) 3 + 4*5 _____

(3) 3/4 + 5 _____ (4) 3 + 4/5 _____

(5) 3*4/5 _____ (6) 3/4*5 _____

(7) 3*(4 + 5) _____ (8) 3/(4*5) _____

(9) 3/(4 + 5) _____ (10) 3/4/5 _____

Use your computer to check the answers. Did you win the prize? If not you can still win by finding a computer that evaluates numerical expressions the same way you do. Good luck!

Exercise 5. Complete the following table showing BASIC expressions that correspond to given mathematical expressions.

MATHEMATICAL EXPRESSION	BASIC EXPRESSION
(1) $2 \times 3 + 4 \div 5$	2*3 + 4/5
(2) $37(43 - 19)$	37*(43 – 19)
(3) $3.14 \times 5 \times 5$	_____
(4) $\frac{2}{3 + 4}$	_____
(5) $\frac{73 - 25}{29 + 53}$	_____

Back to the computer to check your answers.

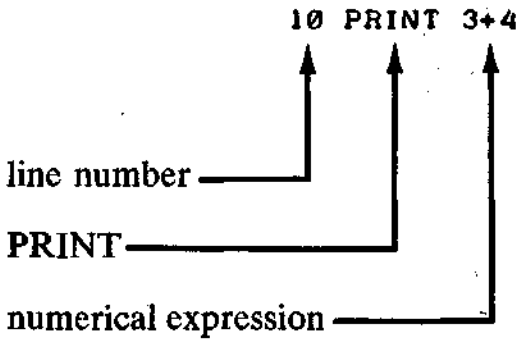
The PRINT Statement

Previously, we used PRINT statements of the following form:

n PRINT *e*

where *n* = line number
e = numerical expression

For example:



A PRINT statement of this form directs the computer to compute the value (simplest form) of the numerical expression *e* and to print the result on the Teletype.

The following program illustrates a PRINT statement that has more than one numerical expression.

```
SCR
```
As usual, we first SCRatch any left-over program.

```
10 PRINT 3+4, 3-4, 3*4, 3/4
99 END
```
Next, we enter the program. The PRINT statement includes *four* expressions.

```
RUN
```
Let's RUN the program.

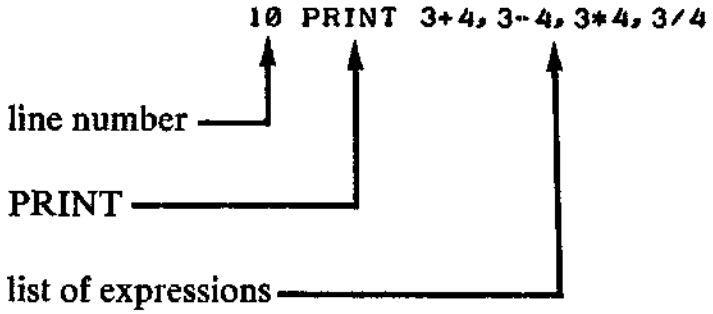
```
 7            -1            12            .75
```

Since the PRINT statement has four expressions, the computer prints four results.

A more general form of the PRINT statement is shown below.

n PRINT - list of expressions

For example,



Remember these things:

- A PRINT statement can contain more than one expression.
- One result is printed for *each* expression in a PRINT statement.
- If a PRINT statement contains more than one expression, then the expressions must be separated by commas.
- Up to five (5) results per line are printed. If there are more than five expressions in the PRINT statement, additional results are automatically printed on the next line.

  For example, the statement

```
10 PRINT 3+4, 3-4, 3*4, 3/4, 3*4*5, 3*4/5, 3/4*5
```

will cause the computer to print the following results.

```
RUN
 7            -1            12            .75           60
 2.4           3.75
```

Exercise 6. Do not use the computer for this exercise. Instead, pretend that you are the computer and RUN each of the following programs.

```
10 PRINT 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
99 END

RUN
```

_____ _____ _____ _____

_____ _____ _____ _____

_____ _____ _____ _____

```
10 PRINT 1*2, 2*3, 3*4, 4*5, 5*6, 6*7, 7*8, 8*9
99 END
RUN
```

_____ _____ _____ _____

_____ _____ _____ _____

Exercise 7. Each new PRINT statement causes a new line to be printed. RUN this program.

```
10 PRINT 1, 1*1
20 PRINT 2, 2*2
30 PRINT 3, 3*3
40 PRINT 4, 4*4
50 PRINT 5, 5*5
60 PRINT 6, 6*6
70 PRINT 7, 7*7
99 END
RUN
```

Once again . . .
Each new PRINT statement causes a new line to be printed.

If we use a semicolon (;) instead of a comma to separate expressions, the results will be packed more closely together. For example, try this one on your computer.

```
SCR
```

Goodbye, old program!

```
10 PRINT 3+4;3-4;3*4;3/4
99 END
```

Note the semicolons (;).

Watch the spacing in the results below.

```
RUN
 7 -1  12  .75
```

The results are "packed" more closely together than if we had used commas.

When we use semicolons to separate expressions, the computer will print up to 17 results per line. The actual number, however, depends on the number of digits that it must print. For example,

```
10 PRINT 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19
99 END
RUN

 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17
 18  19
```

The first 17 results were printed on the first line — the 18th result on the second line.

Let's see what happens as the numbers get larger.

```
10 PRINT 1;12;123;1234;12345;123456;1234567
99 END
RUN
 1  12  123  1234  12345  123456  1.234567E+06
```

This is a *floating point numeral.* We describe floating point numerals in the next section.

Remember these things:

- If a PRINT expression contains more than one expression, then the expressions must be separated by commas (,) or semicolons (;).
- If commas are used for spacing, up to five results per line are printed. If semicolons are used, the results are "packed" more closely together. The actual spacing depends on the size of the numbers involved.
- If you want to find out more about spacing — experiment!

EXPERIMENT!

Floating Point Numerals

Floating point notation is similar to scientific notation. The computer does it this way.

```
10 PRINT 10
20 PRINT 100
30 PRINT 1000
40 PRINT 10000
50 PRINT 100000
60 PRINT 1000000
70 PRINT 10000000
99 END
```

In the *program* each number is expressed in "standard" or "common" notation.

```
RUN
 10
 100
 1000
 10000
 100000
 1.000000E+06
 1.000000E+07
```

The numerals are printed in standard notation, exactly as they are written in the PRINT statements.

But these are printed as *floating point numerals.*

The following examples show the same number expressed in "standard" notation, scientific notation and floating point notation. If a number is larger than 6 digits, its numeral will be printed in floating point notation.

STANDARD NOTATION	SCIENTIFIC NOTATION	FLOATING POINT
1000000	1×10^6	1.000000E + 06
10000000	1×10^7	1.000000E + 07
100000000	1×10^8	1.000000E + 08
1000000000000	1×10^{12}	1.000000E + 12

Let's see how BASIC handles small numbers.

```
10 PRINT .1
20 PRINT .001
30 PRINT .000000001
99 END
RUN
 .1
 1.000000E-03
 1.000000E-09
```

If a number has more than 2 decimal places, its numeral is printed in floating point notation.

Exercise 8. Complete the following table showing the numerals in scientific notation and standard notation corresponding to numerals given in floating point notation as they might be printed by the computer.

FLOATING POINT	SCIENTIFIC NOTATION	STANDARD NOTATION
1.00000E + 09	1×10^9	1000000000
1.00000E − 09	1×10^{-9}	.000000001
2.00000E + 09	2×10^9	2000000000
2.00000E − 08	2×10^{-8}	
3.00000E + 12		
6.02000E + 23	6.02×10^{23}	
1.23456E − 16		.00000000000000123456

Printing Messages

The PRINT statement in the following program directs the computer to print a message.

The message is enclosed in quotation marks.

```
10 PRINT "I LIKE PEOPLE"
99 END
RUN
```

Let's RUN the program.

```
I LIKE PEOPLE
```

The computer types the message.

Here is another example.

```
10 PRINT "GOOD MORNING"
99 END
RUN
GOOD MORNING
```

Unfortunately, if you RUN this program in the afternoon, the computer will still print

```
GOOD MORNING
```

The next example illustrates the difference between a numerical expression and the value of a numerical expression.

```
10 PRINT "3+4=", 3+4
99 END
RUN
3+4=            7

READY.
```

This program directs the computer to print the *message* "3 + 4 =" followed by the *value* of 3 + 4.

If you didn't like the spacing in the printed results, you can use a semicolon instead of a comma to control the spacing.

```
10 PRINT "3+4=";3+4
99 END
RUN

3+4=7
```

Semicolon spacing.

Exercise 9. You be the computer. RUN the following program without using the computer—you do the work and fill in the blanks.

```
10 PRINT "IF I WERE A COMPUTER,"
20 PRINT "I'D DO ARITHMETIC LIKE THIS"
30 PRINT "3+4=";3+4
40 PRINT "3-4=";3-4
50 PRINT "3*4=";3*4
60 PRINT "3/4=";3/4
99 END
RUN
IF I WERE A COMPUTER,
I'D DO ARITHMETIC LIKE THIS
```

_____

_____

Your work is here.

_____

_____

REMEMBER THIS: Anything enclosed in quotation marks in a PRINT statement is printed exactly as it appears. No arithmetic is performed.

Exponents—Computing a Power of a Number

The following program illustrates a new idea — computing a power of a number.

```
10 PRINT "5*5=5↑2=";5↑2
20 PRINT "2*2*2=2↑3=";2↑3
30 PRINT "3*3*3=3↑4=";3↑4
99 END
RUN
5*5=5↑2= 25
2*2*2=2↑3= 8
3*3*3=3↑4= 81
```

The ↑ is on the bottom row of keys. Hold the SHIFT down when you wish to type ↑.

We use the ↑ key when we want to tell the computer to compute a *power* of a number.

In math, we write 2^3, but in BASIC we write 2 ↑ 3. Remember —BASIC notation is *not* math notation even though there are similarities.

Here are some examples showing the values of expressions in which the ↑ is used.

EXPRESSION	VALUE	REMARKS
2↑5	32	2↑5 = 2*2*2*2*2 = 32
3↑2 + 4↑2	25	3↑2 + 4↑2 = 9 + 16 = 25
(2 + 3)↑4	625	(2 + 3)↑4 = 5↑4 = 5*5*5*5 = 625

Exercise 10. Write the value of each expression.

EXPRESSION	VALUE	YOUR REMARKS
1↑3		
7↑2		
3↑3		
4↑5		

Unless parentheses are used to change the order, the computer does powers first, then multiplications and divisions, then additions and subtractions. (See Note 2, page 1-9.)

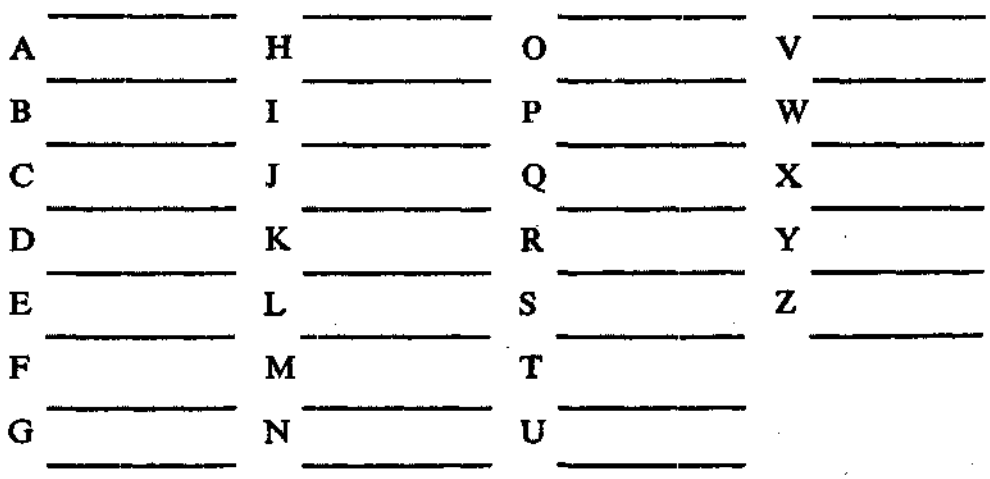
GATHERING SPEED

Variables

In mathematics, we have great freedom in selecting symbols to use as variables. We use the letters A — Z of our alphabet, the letters of the Greek alphabet and, in fact, any symbol that we may "invent" for this purpose. In BASIC, however, we must restrict you in your choice of symbols. For now, we impose this rule:

> A BASIC variable may be any letter of the alphabet. That is, any of the following may be used as a variable:
>
> A B C D E F G H I J K L M
>
> N O P Q R S T U V W X Y Z

In BASIC, each variable refers to a distinct *location* in the computer's memory. It may help you to think of the computer's memory as a set of 26 boxes, labeled A through Z, like this:



We call these boxes *locations.* Each location can hold one number at any one time. This number is the *value* of the variable corresponding to the location.

Exercise 11. LET A = 3. In other words, take pencil in hand and write the numeral "3" in the box labeled "A." Then do the following in similar fashion:

(1) LET B = 4 (2) LET P = 3.14 (3) LET Z = −1

The following example shows how we assign a value to a variable in a BASIC program.

```
10 LET A=3
20 PRINT A
99 END
RUN
 3
```

Assign the value 3 to the variable *A*. Print the value of *A*.

And here it is (the value of *A*, that is).

A more general form of the LET statement is shown below.

n LET *v* = *e*

where *n* = line number
v = BASIC variable
e = numerical expression

For example,

```
10 LET S=2*3+4*5
```

Here is some additional evidence. You may wish to RUN the following program.

```
SCR
```

Don't forget to SCRatch!

```
10 LET A=3
20 LET B=4
30 LET C=3+4
40 LET D=3-4
50 LET E=3*4
60 LET F=3/4
70 LET G=3↑4
80 PRINT A;B;C;D;E;F;G
99 END
RUN
 3   4   7  -1   12   .75   81
```

(Note the semicolons.)

Let's see what happens—blow by blow—as the computer RUNs the program. Below is a trace of the program from the preceding page. The trace shows the value of each variable *after* the statement on the same line has been carried out by the computer.

STATEMENT	A	B	C	D	E	F	G
10 LET A = 3	3						
20 LET B = 4	3	4					
30 LET C = 3 + 4	3	4	7				
40 LET D = 3 − 4	3	4	7	−1			
50 LET E = 3*4	3	4	7	−1	12		
60 LET F =3/4	3	4	7	−1	12	.75	
70 LET G = 3 ↑ 4	3	4	7	−1	12	.75	81
80 PRINT A;B;C;D;E;F;G	3	4	7	−1	12	.75	81
99 END	3	4	7	−1	12	.75	81

The *trace* is an important idea—from now on, we will depend on it. Therefore, you had better learn how to (1) read a trace and (2) do a trace.

Exercise 12. Trace the following program.

STATEMENT	P	Q	R	S
10 LET P = 5				
20 LET Q = −123				
30 LET R = 57.3				
40 LET S = 2*3 + 4*5				
50 PRINT P, Q, R, S				
99 END				

The LET statement directs the computer to compute the value of the expression to the right of the "=" symbol and assign this value to the variable that appears to the left of the "=" symbol. This value *replaces* any previous value of the variable. For example,

STATEMENT	A	REMARKS
10 LET A = 1	1	Assign the value 1 to *A*.
15 PRINT A	1	Print the current value of *A*.
20 LET A = 2	2	Assign the value 2 to *A*.
25 PRINT A	2	Print the current value of *A*.
30 LET A = 3	3	Assign the value 3 to *A*.
35 PRINT A	3	Print the current value of *A*.
99 END	3	

If we RUN the program above, we obtain the following results.

```
RUN
1
2
3
```

Exercise 13. Without using the computer, RUN each of the following programs. (Fill in the blanks.)

```
10 LET X=3
20 LET X=5
30 LET X=7
40 PRINT X
99 END
RUN
```

_____

```
10 LET X=3
20 LET Y=5
30 LET Z=7
40 PRINT X;Y;Z
99 END
RUN
```

_____ _____ _____

P.S.

On EduSystems, you don't have to type the word LET in an expression. In other words,

10 A = 1 is equivalent to 10 LET A = 1

To be consistent with Dartmouth BASIC, we'll use LET in this manual but you don't have to on your EduSystem.

Variable Expressions

A variable expression is an expression that contains a variable. For example, the following are variable expressions:

A A − B 2*X P/Q −C A*(B + C)

A/B + C/D 3.14*R ↑ 2

We *evaluate* a variable expression by assigning values to its variable or variables and carrying out the indicated operations.

For example, A*B is a variable expression with variables A and B. If A = 3 and B = 4, then the value of A*B is 12. But if A = −7 and B = 5, then the value of A*B is −35.

VARIABLE EXPRESSION	VALUE(S) OF VARIABLE(S)	VALUE OF EXPRESSION
A	$A = 3$	3
	$A = -123$	−123
A − B	$A = 12$ and $B = 7$	5
	$A = 3$ and $B = 4$	−1
2*X	$X = 3.14$	6.28
	$X = -6$	−12
P/Q	$P = 35$ and $Q = 5$	7
	$P = 2$ and $Q = 3$	.666667
−C	$C = 8$	−8
	$C = 0$	0
	$C = -12$	12
A*(B + C)	$A = 3, B = 4, C = 5$	27
3.14*R ↑ 2	$R = 3$	28.26

Each of the following programs directs the computer to evaluate one or more variable expressions and print the result or results. We use LET statements to assign values to variables.

```
10 LET A=3
20 LET B=4
30 PRINT A+B
99 END
RUN
 7
```

```
10 LET A=3
20 LET B=4
30 PRINT A*B
99 END
RUN
 12
```

Exercise 14. Without using the computer, complete each RUN by filling the blank with the result.

```
10 LET A=3
20 LET B=4
30 PRINT A-B
99 END
RUN
```

_____

```
10 LET A=3
20 LET B=4
30 PRINT A/B
99 END
RUN
```

_____

Exercise 15. The following program illustrates the use of variable expressions. Trace the program by filling in the blanks under the headings "A" through "G."

PROGRAM	A	B	C	D	E	F	G
10 LET A = 3							
20 LET B = 4							
30 LET C = A + B							
40 LET D = A − B							
50 LET E = A*B							
60 LET F = A/B							
70 LET G = A ↑ B							
80 PRINT A;B;C;D;E;F;G							
99 END							

FEEDING THE BEAST

The INPUT Statement

In this section, we introduce a statement called the INPUT statement. But first, let's solve a problem that may point up the need for the INPUT statement.

Problem. The area A of a circle of radius R is given by the formula:

$$A - \pi R^2 \qquad \text{where } \pi = 3.14$$

We want to use the computer to compute the areas of three different circles. These circles have radii

$$R = 2,\ R = 3, \text{ and } R = 8.$$

Here is a step-by-step description of how we could use a computer to solve the problem.

```
SCR
```
First, SCRatch any old program.

```
10 LET R=2
20 PRINT R,3.14*R↑2
99 END
```
Here is our program. It will work for $R = 2$.

```
RUN
```
Let's RUN it.

```
2             12.56
```
For $R = 2$, the area is 12.56.

```
10 LET R=3
```
Do NOT type SCR. Instead, enter a *new* Statement 10.

```
RUN
 3             28.26
```
For $R = 3$, $A = 28.26$.

```
10 LET R=8
```
Again—change Statement 10.

```
RUN
```
And RUN the program.

```
8             200.96
```
For $R = 8$, $A = 200.96$.

We can reduce the amount of work required to solve the problem on the preceding page by using the INPUT statement. Here is a program that uses an INPUT statement to permit input of a value of R.

```
10 INPUT R
20 PRINT R,3.14*R↑2
99 END
RUN
?
```

We enter this program.

And tell the computer to RUN it.

The computer types a question mark and stops.

For a second, we sit and contemplate that question mark. What does the computer want? Of course! It wants a value for R. So we enter 2 as the value of R and press the RETURN key. The computer then prints

```
2         12.56
```

O.K., here is a RUN for $R = 2$, $R = 3$ and $R = 8$. Try this one on your computer.

```
SCR
```

SCRatch any previous program.

```
10 INPUT R
20 PRINT R,3.14*R↑2
99 END
```

Enter the program.

```
RUN
```

RUN the program.

```
?2
 2      12.56
```

Enter 2 and press RETURN.
For $R = 2$, $A = 12.56$.

```
RUN
```

RUN the program again.

```
?3
 3      28.26
```

Enter 3 and press RETURN.
For $R = 3$, $A = 28.26$.

```
RUN
```

RUN the program again.

```
?8
 8      200.96
```

Enter 8 and press RETURN.
For $R = 8$, $A = 200.96$.

The general form of the INPUT statement is

line number INPUT list of variables

For example,

```
10 INPUT A,B,C
```

line number

INPUT

list of variables

Note that only the variables in the list are separated by commas. There is *no* comma following the word "INPUT" and there is *no* comma after the last variable in the list.

The INPUT statement directs the computer to type a question mark and then stop and wait. Now you must understand that computers are very patient—if you don't cooperate, the computer will simply wait—and wait—and wait. To prevent this from happening, all you have to do is feed the computer—it's hungry—it wants data.

Remember these things:

- The INPUT statement causes the computer to type a question mark.
- When the question mark appears, the operator must enter one value for *each* variable in the INPUT statement. The values are entered in the same left to right order as the variables appear in the INPUT statement.
- Don't forget to type commas *between* values.
- After entering the last number, press the *RETURN* key. If you have done everything correctly, the computer will proceed.

Here is another example.

```
17 INPUT A,B,C
23 PRINT A*(B+C)
99 END
RUN
?3,4,5
 27
```

Since there are *three* variables, we must enter three values.

If $A = 3, B = 4, C = 5$, then $A*(B + C) = 27$.

Exercise 16. There is something wrong with each INPUT statement shown below. For each one, circle the mistake and write the reason.

INCORRECT STATEMENT	REASON
10 INPUT ,A,B,C,	
20 INPUT X,Y,	
30 IMPUT P,Q,R,S,T	
40 INPUT A+B	
50 INPUT I;J;K	
60 INPUT AA,BB,	
70 INPUT A B C	

Exercise 17. We ran two simple programs. Here they are but some things are missing. Complete each RUN by filling in the blanks.

```
10 INPUT A,B,C
20 PRINT A
30 PRINT B,C
99 END

RUN
?-2,-3,-4

_________

_________ _________
```

```
12 INPUT U,V,W,X
25 PRINT U,U+V
36 PRINT W,W*X
99 END

RUN
?____ ____

7       12
3       45
```

The GO TO Statement

The following program appeared on page 1-27.

```
10 INPUT R
20 PRINT R,3.14*R↑2
99 END
```

When we used it, we had to type RUN for *each* value of *R*. (See page 1-27.) To eliminate the need to type RUN for each new value of *R*, we add the following GO TO statement.

```
30 GO TO 10
```

(This directs the computer to "GO TO Statement 10.")

Here is a RUN of the modified program. Try it on your computer.

```
SCR
```

First, let's SCRatch.

```
10 INPUT R
20 PRINT R,3.14*R↑2
```

Then enter the program.

```
30 GO TO 10
99 END
```

Here is our GO TO statement.

```
RUN
```

Now let's RUN the program.

```
?2
 2      12.56
?3
 3      28.26
?8
 8      200.96
?
```

Each time after printing the results the computer does a GO TO 10 and automatically restarts at the INPUT statement.

How do we tell the computer we are finished? Hold CTRL down, press C, and release. The computer will stop.

The GO TO statement has the general form

line number GO TO line number

The GO TO statement directs the computer to GO TO the statement that has this

For example: 30 GO TO 10

line number

GO TO

line number

Exercise 18. Do not use the computer to answer this. If you were a computer and you came to the following statement, what would you do?

```
45 GO TO 45
```

Exercise 19. Complete the following program to convert from degrees Centigrade to degrees Fahrenheit.

In math notation, the formula is $F = \frac{9}{5} C + 32$

```
10 INPUT C
20 LET F=                          (You write the formula—in
30 PRINT C,F                       BASIC.)
40 PRINT
50 GO TO                           (GO TO where?)
99 END

RUN
?0
 0              32                 If C = 0, then F = 32.

?100
 100            212                If C = 100, then F = 212.

?37
 37                                (Hint: Body temperature)

?
                77                 Give F, what is C?
```

READ and DATA Statements

Whenever possible, we prefer providing data (values of variables) by means of the READ and DATA statements. The following program is a modification of our "Area of a Circle" friend on page 1-26.

```
10 READ R
20 PRINT R,3.14*R↑2

30 DATA 2,3,8

40 GO TO 10
99 END
RUN
 2                12.56
 3                28.26
 8                200.96

DATA ERROR AT LINE 10
```

This is a READ statement.

This is a DATA statement.

Here are the results. On each line, the value of *R* is on the left and the value of the area is on the right.

This message may be different or even omitted. It simply means that the computer has READ all the DATA.

The statement

```
10 READ R
```

tells the computer to read *one* value of *R* from the list of values in the DATA statement. Each time the READ statement is executed, the computer reads the *next* value from the DATA statement. In other words, the computer remembers what values have already been read.

If there is no more data to be read in the DATA statement, the computer stops automatically.

Here is another example using the READ and DATA statements.

Four students named Frodo, Sam, Gandalf, and Strider have each taken three quizzes. Their scores are:

STUDENT	FIRST SCORE	SECOND SCORE	THIRD SCORE
Frodo	66	81	75
Sam	91	88	95
Gandalf	78	78	62
Strider	80	83	86

We have written a program to compute the arithmetic mean (average) of three scores and have run it for the above data.

```
10 READ X,Y,Z
20 LET M=(X+Y+Z)/3
30 PRINT X,Y,Z,M
40 GO TO 10
90 DATA 66,81,75
91 DATA 91,88,95
92 DATA 78,78,62
93 DATA 80,83,86
99 END
```

We use *X*, *Y*, *Z* to denote the first, second and third scores.

Frodo's scores.
Sam's scores.
Gandalf's scores.
Strider's scores.

```
RUN
 66            81            75            74
 91            88            95            91.33333
 78            78            62            72.66667
 80            83            86            83

DATA ERROR AT LINE 10
```

The averages of the three scores are in this column.

DATA statements may be placed anywhere in the program. They must, however, have line numbers smaller than the line number of the END statement.

The general form of the READ statement is

line number READ list of variables

For example: 10 READ X, Y, Z

line number

READ

list of variables

The variables are separated by commas.

Exercise 20. There is probably[3] something wrong with each of the following READ statements. For each one, circle the mistake (if possible) and write the reason.

INCORRECT STATEMENT	REASON
10 READ, A,B,C	
20 READ X,Y	
30 REED P,Q,R,S,T	
40 READ A+B	
50 READ I;J;K	
60 READ AA,BB	
70 READ ABC	
80 READ 3.14	

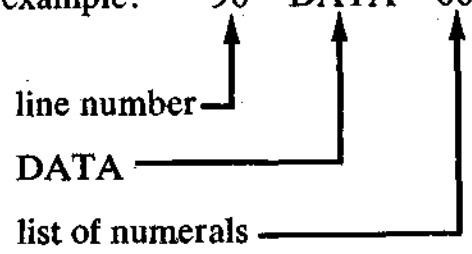
The READ statement directs the computer to read one value from the DATA statement for *each* variable in the READ statement. If there are two or more DATA statements in a program, the values in the statement with the smallest line number are used first, then the data in the statement with the next smallest line number and so on.

[3] Depends on the system . . . EXPERIMENT!

The general form of the DATA statement is:

line number DATA list of numerals

For example: 90 DATA 66, 81, 75



Expressions such as 2 + 3 are not allowed in the list.

Exercise 21. There is something wrong with *each* of the following DATA statements. For each one, circle the mistake and write the reason.

INCORRECT STATEMENT	REASON
10 DATE 1, 2, 3, 4	
20 DATA 1/2, 2/3, 3/4	
30 DATA A, B, C, D, E	
40 DATA, 3.7, 2.9	
50 DATA 3.7, 2.9,	

The following three sets of DATA statements are equivalent.

```
90 DATA 2, 3, 6, 8, 12, 15, 19, 27, 33, 26, 47, 59
```

```
90 DATA 2, 3, 6
91 DATA 8, 12, 15, 19, 27
92 DATA 33, 26, 47, 59
```

```
90 DATA 2, 3, 6, 8, 12, 15
91 DATA 19, 27, 33, 26, 47, 59
```

That's right, the numerals in the list are separated by commas.

The RESTORE Statement

The RESTORE statement allows you to reuse DATA statements, beginning with the lowest numbered DATA statement in the program. An example of the use of the RESTORE statement is shown below:

```
10 DATA 2,3,6
20 DATA 8,12,15
30 READ A,B,C,D
40 PRINT A,B,C,D
50 RESTORE
60 READ E,F
70 PRINT E,F
99 END
```

The RESTORE statement at line 50 allows the READ statement at line 60 to obtain values from the DATA statement at line 10

```
RUN
 2             3             6             8
 2             3
```

Without the RESTORE statement, an error message would have occurred, indicating a lack of data for the READ statement at line 60.

Exercise 22. Without using the computer, RUN each of the following programs. (Fill in the blanks.) If you wish, check your answers with the computer.

```
10 DATA 1,2
20 READ A,B
30 PRINT A,B
40 RESTORE
50 DATA 3,4
60 READ C,D
70 PRINT C,D
99 END
RUN
```

```
10 READ X
20 PRINT X
30 RESTORE
40 READ Z
50 PRINT Z
60 DATA 4,1,2
70 DATA 3,5,7
99 END
RUN
```

More Messages

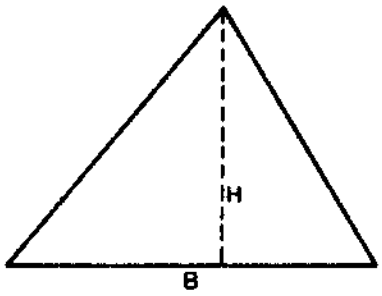
We can make the results more readable by including a statement that causes the computer to print a heading. For example

```
5  PRINT "RADIUS","AREA"                    Print a heading.
10 READ R
15 DATA 2,3,8
20 PRINT R,3.14*R↑2
25 GO TO 10
99 END
RUN
RADIUS         AREA
 2              12.56
 3              28.26
 8              200.96

DATA ERROR AT LINE 10
```

Exercise 23. Time to play computer. Without using the computer, RUN the following program.

```
10 PRINT "L","R","L*10+R","L*8+R"              Headings
13 READ L,R
21 PRINT L,R,L*10+R,L*8+R
27 GO TO 13
85 DATA 0,7,1,0,1,1,1,7,2,0,7,7
99 END

RUN

_____
_____
_____
_____                                  Numerical results
_____
_____
_____
```

Exercise 24. We have written a program to compute the area of a triangle, given the base B and the height H as data.



AREA = $\frac{1}{2}$BH = BH/2

We want to complete the following table.

B	*H*	*AREA*
7	6	
8	12	
5	9	
23	17	

Here is our program. Complete it by filling in the DATA statement. Then, if you can get on the computer, RUN it and complete the above table.

```
10 PRINT "B","H","AREA"
20 READ B,H
30 PRINT B,H,B*H/2
40 GOTO 20
90 DATA
99 END
```

YOU CAN COUNT ON IT

Loops

Let's teach the computer to count. That is, let's develop a program to direct the computer to generate and print consecutive *counting numbers*. The counting numbers are:

1, 2, 3, 4, 5, 6, 7, 8, . . .

Here is our program and a RUN as evidence that it does what we claim.

Beware! If you RUN the following program you may have trouble stopping the computer. Check with someone who knows how to interrupt your computer.

```
10 LET K=1
20 PRINT K
30 LET K=K+1
40 GOTO 20
99 END
```

Assign the value 1 to *K*.
Print the current value of *K*.
Increase the value of *K* by 1.
Go around again.

```
RUN
```

Let's RUN the program.

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

If we don't interrupt the computer, it will go on and on—printing counting numbers. We can interrupt the computer by holding the CTRL key down and pressing C.

and so on.

The above program contains a loop. The loop is described below.

```
10 LET K=1
20 PRINT K
30 LET K=K+1
40 GO TO 20
99 END
```

This is a loop. The statements in the loop are repeated indefinitely. Each time through the loop, the current value of *K* is increased by 1 and the loop is repeated.

If you are confused by the statement

30 LET K = K + 1

BEFORE		STATEMENT	AFTER	
K	1	30 LET K = K + 1	K	2
K	2	30 LET K = K + 1	K	3
K	3	30 LET K = K + 1	K	4

Remember the general form of the LET statement.

line number LET variable = expression

The expression may be any BASIC expression. The LET statement directs the computer to evaluate the expression and then assign the value to the variable. If the expression is a variable expression, it is evaluated using the current values of its variable or variables.

Therefore, the statement LET K = K + 1 directs the computer to evaluate the expression $K + 1$ using the current value of K and then assign the *new* value to K.

Exercise 25. Show the value of the variable after the statement has been executed.

BEFORE		STATEMENT	AFTER
K	25	30 LET K = 1	K
E	6	40 LET E = E + 2	E

Loops Exposed

In order to clarify what happens as the computer executes the program, we will "unwrap" the loop and trace it. The following trace shows the value of *K following* the execution of each statement in the program. Under the heading "OUTPUT" we also show results printed by the computer. We have traced the program seven times through the loop.

Study this trace carefully. We will ask you to do several such traces.

STATEMENT	K	OUTPUT	REMARKS
10 LET K = 1	1		
20 PRINT K	1	1	First time through the loop.
30 LET K = K + 1	2		
40 GO TO 20	2		
20 PRINT K	2	2	Second time through the loop.
30 LET K = K + 1	3		
40 GO TO 20	3		
20 PRINT K	3	3	Third time through the loop.
30 LET K = 1 +1	4		
40 GO TO 20	4		
20 PRINT K	4	4	Fourth time through the loop.
30 LET K = K + 1	5		
40 GO TO 20	5		
20 PRINT K	5	5	Fifth time through the loop.
30 LET K = K + 1	6		
40 GO TO 20	6		
20 PRINT K	6	6	Sixth time through the loop.
30 LET K = K + 1	7		
40 GO TO 20	7		
20 PRINT K	7	7	Seventh time through the loop.
30 LET K = K + 1	8		
40 GO TO 20	8		

and so on!

Exercise 26. Without using the computer, show the first five results printed by the computer under control of each of the following programs. (Fill in the blanks.)

```
10 LET X=1
20 PRINT X
30 LET X=X+2
40 GO TO 20
99 END

RUN
```

_____
_____
_____
_____
_____

and so on.

```
10 LET E=2
20 PRINT E
30 LET E=E+2
40 GO TO 20
99 END
RUN
```

_____
_____
_____
_____
_____

and so on.

Exercise 27. Complete each program (fill in the blanks) so that when we run the program, the computer will produce the results shown.

```
10 LET J=
20 PRINT J
30 LET J=
40 GO TO 20
99 END

RUN

0
1
2
3
4
```

and so on.

```
12 LET P=
25 PRINT P
33 LET P=
41 GO TO
99 END

RUN

1
2
4
8
16
```

and so on.

Exercise 28. This is a trace of one of the programs shown in Exercise 26. Complete the trace for three times through the loop.

STATEMENT	X	OUTPUT	REMARKS
10 LET X = 1	1		
20 PRINT X	1	1	First time through loop.
30 LET X = X + 2	3		
40 GO TO 20	3		
20 PRINT X	—	—	Second time.
30 LET X = X + 2	—		
40 GO TO 20	—		
20 PRINT X	—	—	Third time.
30 LET X = X + 2	—		
40 GO TO 20	—		
20 PRINT X	—	—	Fourth time and we quit!
and so on.			

Exercise 29. Here are two variations on previously written programs. For each one, show the first four results.

```
10 LET K=0
20 LET K=K+1
30 PRINT K
40 GO TO 20
99 END

RUN
```

_____

_____

_____

_____

and so on.

```
10 LET K=1
20 PRINT 2*K
30 LET K=K+1
40 GO TO 20
99 END

RUN
```

_____

_____

_____

_____

and so on

Exercise 30. Trace the following program four times through the loop.

```
10 LET A=1
17 LET B=1
25 LET C=A+B
30 PRINT A
36 LET A=B
43 LET B=C
50 GO TO 25
99 END
```

STATEMENT	A	B	C	REMARKS
10 LET A = 1				These statements are done once.
17 LET B = 1				
25 LET C = A + B				First time through loop.
30 PRINT A				
36 LET A = B				
43 LET B = C				
50 GO TO 25				
25 LET C = A + B				Second time through loop.
30 PRINT A				
36 LET A = B				
43 LET B = C				
50 GO TO 25				
25 LET C = A + B				Third time through loop.
30 PRINT A				
36 LET A = B				
43 LET B = C				
50 GO TO 25				
25 LET C = A + B				Fourth time through loop.
30 PRINT A				
36 LET A = B				
43 LET B = C				
50 GO TO 25				

FOR-NEXT Loops

The loops we have used so far do not terminate by themselves. They go on, and on, and on . . . until someone manually interrupts. Now let's look at a loop that terminates automatically. This loop makes use of two new statements called the FOR statement and NEXT statement.

```
10 FOR K=1 TO 5
20 PRINT K
30 NEXT K
99 END
RUN
 1
 2
 3
 4
 5
```

We call this a FOR-NEXT loop.

Let's Run it and see what happens.

We did *not* interrupt.

The computer stopped *automatically*.

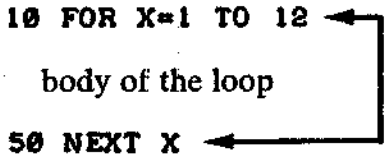
Perhaps the following trace will help you understand how a FOR-NEXT loop works.

STATEMENT	*K*	OUTPUT	REMARKS
10 FOR K = 1 TO 5	1		*K* starts at 1.
20 PRINT K	1	1	First time through loop.
30 NEXT K	2		$K \leqslant 5$. Do it again.
20 PRINT K	2	2	Second time through loop.
30 NEXT K	3		$K \leqslant 5$. D it again.
20 PRINT K	3	3	Third time through loop.
30 NEXT K	4		$K \leqslant 5$. Do it again.
20 PRINT K	4	4	Fourth time through loop.
30 NEXT K	5		$K \leqslant 5$. Do it again.
20 PRINT K	5	5	Fifth time through loop.
30 NEXT K	6		$K > 5$. Stop the loop!
99 END	6		Everything stops.

A FOR-NEXT loop consists of three things.

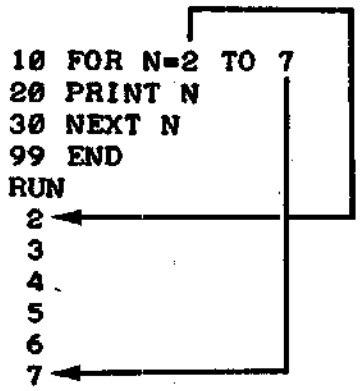
1. A FOR statement
2. A NEXT statement
3. A set of statements between the FOR statement and the NEXT statement.

A FOR-NEXT loop begins with a FOR statement and ends with a NEXT statement. The set of statements between FOR and NEXT is called the *body of the loop*. Here is a "pictorial" representation for a FOR-NEXT loop.



The same variable must be used in both places.

Here is another example.



This FOR statement defines a set of values for *N*. The set is:

[2, 3, 4, 5, 6, 7]

The body of the loop is executed repeatedly, once for each value of *N* defined by the FOR statement.

Every FOR statement must have a NEXT statement and every NEXT statement must have a FOR statement.

Has it occurred to you that the value of the variable increases by *one* each time through the loop?

Exercise 31. In each program, circle the *body* of the FOR-NEXT loop.

```
10 PRINT "RADIUS","AREA"
20 FOR R=2 TO 4
30 PRINT R,3.14*R↑2
40 NEXT R
99 END
RUN
RADIUS          AREA
 2               12.56
 3               28.26
 4               50.24
```

```
10 PRINT "RADIUS","AREA"
20 FOR R=2 TO 4
25 LET A=3.14*R↑2
30 PRINT R,A
40 NEXT R
99 END

RUN
RADIUS          AREA
 2               12.56
 3               28.26
 4               50.24
```

Remember these things.

- The FOR statement defines a *set* of values for the variable.
- The body of the loop is executed for each member of the set.
- The NEXT statement causes the body of the loop to be executed again, using the next member of the set. However, if all members of the set have already been used, then the NEXT statement directs the computer to move on to the statement following the NEXT statement.

The following shows the set of values defined for the variable in each example of a FOR statement. We have omitted line numbers.

FOR Statement	Variable	Set of Values for the Variable
FOR J = 0 TO 3	J	[0, 1, 2, 3]
FOR I = 1 TO 1	I	[1]
FOR A = 3 TO 5	A	[3, 4, 5]
FOR X = −2 TO 2	X	[−2, −1, 0, 1, 2]
FOR B = 1 TO 0	B	Empty — the loop is skipped.

Do you see a pattern? Try the following exercises.

Exercise 32. Complete the following table.

FOR Statement	Variable	Set of Values for the Variable
FOR N = 1 TO 6	N	
FOR C = 0 TO 5	C	
FOR W = −3 TO 0		
FOR E = 12 TO 12		
FOR T = 7 TO 5		

The next three are tricky. Be brave—guess!

FOR X = .5 TO 2.5		
FOR Y = 1 TO 2.5		
FOR Z = .5 TO 3		

Exercise 33. Time to play computer. RUN each program as if you were the computer.

```
10 FOR X=2 TO 4
20 PRINT X,X*X
30 NEXT X
99 END

RUN
```

_____ _____

_____ _____

_____ _____

```
12 FOR K=1 TO 3
23 LET P=2↑K
30 PRINT K,P
37 NEXT K
99 END

RUN
```

_____ _____

_____ _____

_____ _____

Instead of using the computer to grind out numbers, let's use it to print patterns.

```
10 FOR K=1 TO 8
20 PRINT "**********"
30 NEXT K
99 END
RUN
**********
**********
**********
**********
**********
**********
**********
**********
```

```
10 FOR K=1 TO 7
20 PRINT "XOXOXOXOX"
30 NEXT K
99 END
RUN
XOXOXOXOX
XOXOXOXOX
XOXOXOXOX
XOXOXOXOX
XOXOXOXOX
XOXOXOXOX
XOXOXOXOX
```

```
10 FOR K=1 TO 3
20 PRINT "+-+-+-+-+-+-+"
30 PRINT "-+-+-+-+-+-+-"
40 NEXT K
99 END
RUN
+-+-+-+-+-+-+
-+-+-+-+-+-+-
+-+-+-+-+-+-+
-+-+-+-+-+-+-
+-+-+-+-+-+-+
-+-+-+-+-+-+-
```

```
10 PRINT "XXXXXXXXX"
20 FOR K=1 TO 5
30 PRINT "XOOOOOOOX"
40 NEXT K
50 PRINT "XXXXXXXXX"
99 END
RUN
XXXXXXXXX
XOOOOOOOX
XOOOOOOOX
XOOOOOOOX
XOOOOOOOX
XOOOOOOOX
XXXXXXXXX
```

Exercise 34. Write a program to generate each pattern.

```
+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-
+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-
+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-
+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-
+-+-+-+-+-+-+-+
```

```
*******
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*******
```

The STEP Clause

A variation of the FOR statement is shown in the following program.

```
10 FOR K=1 TO 9 STEP 2
20 PRINT K
30 NEXT K
99 END
RUN
 1
 3
 5
 7
 9
```

Note the STEP clause.

The STEP 2 clause causes the value of *K* to increase by 2 each time. You can verify this by examining the printed results.

The following shows the set of values defined for the variable in each FOR statement. We have omitted line numbers.

FOR Statement	Values of the Variable
FOR E = 0 TO 10 STEP 2	E = 0, 2, 4, 6, 8, 10
FOR E = 0 TO 11 STEP 2	E = 0, 2, 4, 6, 8, 10
FOR X = 1 TO 3 STEP .5	X = 1, 1.5, 2, 2.5, 3
FOR W = 1 TO 7 STEP 0	W = 1, 1, 1, . . .

Exercise 35. Complete the following table.

FOR Statement	Values of the Variable
FOR T = 0 TO 6 STEP 3	T =
FOR N = 1 TO 5 STEP 1	N =
FOR K = 100 TO 130 STEP 10	K =
FOR X = 0 TO 1 STEP .25	X =
Now—be careful on the next two!	
FOR E = 0 TO 0 STEP 2	E =
FOR B = 3 TO 0 STEP −1	B =

In the following example, we use a space-saving trick.

```
12 FOR X=0 TO 12 STEP 2
23 PRINT X,                                   Aha! Note the comma.
35 NEXT X
99 END
RUN


 0            2            4            6            8
 10           12
```

Exercise 36. If the PRINT statement ends with a comma, results are printed across the page, with up to five results per line. Here is another example—we call it program COUNTDOWN.

```
10 FOR N=10 TO 1 STEP -1
20 PRINT N,
30 NEXT N
40 PRINT "BLAST-OFF!!!"
50 FOR K=1 TO 55
60 PRINT ">";
70 NEXT K
99 END
                                   A semicolon to the rescue!
RUN
 10           9            8            7            6
 5            4            3            2            1
BLAST-OFF!!!
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Be sure to RUN this program on your computer.

Variable FOR Statements

By using variables instead of numerals, we obtain variable FOR statements such as the one in the following program.

```
10 INPUT N
20 FOR K=1 TO N
30 PRINT K
40 NEXT K
50 GOTO 10
99 END

RUN
?3
1
2
3
?5
1
2
3
4
5
?0
```

See it!

Variable FOR-NEXT loop.

We enter 3 as the value of N.

For $N = 3$, $K = 1, 2, 3$.

We enter 5 as the value of N.

For $N = 5$, $K = 1, 2, 3, 4, 5$

We enter 0 as the value of N. The FOR loop is skipped because $1 > N$.

and so on.

Change one statement in the above program.

```
10 INPUT N
20 FOR K=1 TO N
30 PRINT "*"
40 NEXT K
45 PRINT
50 GOTO 10
99 END

RUN
?3
***

?7
*******
```

and so on.

Here is the change.

And we added this statement. If you want to know why, RUN the program without it.

We enter 3 as the value of N. And the computer prints 3 asterisks.

We enter 7 as the value of N. And the computer prints 7 asterisks.

Carry on—*you* pick the value of N.

Another example. Study it carefully.

```
10 INPUT A,B
20 FOR X=A TO B
30 PRINT X;
40 NEXT X
50 PRINT
60 GO TO 10
99 END

RUN
?3,8
 3   4   5   6   7   8
?1,13
 1   2   3   4   5   6   7   8   9   10   11   12   13
```

The semicolon packs 'em in.

Let's trace the above program for $A = 3$, $B = 8$.

STATEMENT	A	B	X	OUTPUT	REMARKS
10 INPUT A, B	3	8			
20 FOR X = A TO B	3	8	3		
30 PRINT X;	3	8	3	3	First time
40 NEXT X	3	8	4		$X \leqslant B$
30 PRINT X;	3	8	4	4	Second time
40 NEXT X	3	8	5		$X \leqslant B$
30 PRINT X;	3	8	5	5	Third time
40 NEXT X	3	8	6		$X \leqslant B$
30 PRINT X;	3	8	6	6	Fourth time
40 NEXT X	3	8	7		$X \leqslant B$
30 PRINT X;	3	8	7	7	Fifth time
40 NEXT X	3	8	8		$X \leqslant B$
30 PRINT X;	3	8	8	8	Sixth time
40 NEXT X	3	8	9		$X > B$. Stop the loop!
50 PRINT	3	8	9		A carriage return
60 GO TO 10	3	8	9		and a line feed.
10 INPUT A, B	and so on,				

Exercise 37. The following program directs the computer to compute and print the sum and the arithmetic mean of N numbers. Trace it.

```
10 READ N
20 LET S=0
30 FOR K=1 TO N
40 READ X
50 LET S=S+X
60 NEXT K
70 PRINT N,S,S/N
90 DATA 3
91 DATA 87,73,95
99 END
```

The value of *N* is in Line 90.

The values of *X* are in line 91.

Here is the value of *N*. (See Line 10.) And here are the values of *X*. Note that there are *N* values.

STATEMENT	*N*	*S*	*K*	*X*	OUTPUT
10 READ N					
20 LET S = 0					
30 FOR K = 1 TO N					
40 READ X					
50 LET S = S + X					
60 NEXT K					
40 READ X					
50 LET S = S + X					
60 NEXT K					
40 READ X					
50 LET S = S + X					
60 NEXT K					
70 PRINT N, S, S/N					
99 END					

Exercise 38. How would you modify the two DATA statements to compute the mean of the numbers: 75, 66, 83, 75, 98?

```
90 DATA _____
91 DATA _____
```

Extra for Experts

The examples on this and the following page may help you discover more things about the FOR statement and how it is used. We also encourage you to *experiment*—try out your own ideas on the computer. You may wish to guess how something works and then try it out.

```
10 INPUT A,B,H
20 PRINT "RADIUS","AREA"
30 FOR R=A TO B STEP H
40 PRINT R,3.14*R↑2
50 NEXT R
55 PRINT
60 GO TO 10
99 END

RUN
?2,8,3
RADIUS          AREA
 2               12.56
 5               78.5
 8               200.96
```

Compare!

($A = 2$, $B = 8$ and $H = 3$)

The value of R is stepped from 2 to 8 in steps of 3.

Another example—another idea.

```
10 READ N
20 FOR K=N*N TO N*(N+1)
30 PRINT K;
40 NEXT K
50 PRINT
60 GO TO 10
90 DATA 3,7
99 END

RUN
 9  10  11  12
 49  50  51  52  53  54  55  56

DATA ERROR AT LINE 10
```

We can use any BASIC expression.

Note the semicolon.

Causes a line space. That is, prints a blank line since there is no list following the word PRINT.

Results for $N = 3$

Results for $N = 7$

We can have a loop *within* a loop.

```
10 PRINT "I","J"
15 PRINT
20 FOR I=1 TO 2
30 FOR J=1 TO 3
40 PRINT I,J
50 NEXT J
60 NEXT I
99 END

RUN
I             J

 1             1
 1             2
 1             3
 2             1
 2             2
 2             3
```

Inside loop

Outside loop

We can use a loop within a loop to print a pattern of *M* rows with *N* asterisks in each row.

```
10 INPUT M,N
20 FOR I=1 TO M
30 FOR J=1 TO N
40 PRINT "*";
50 NEXT J
60 PRINT
70 NEXT I
80 PRINT
90 GO TO 10
99 END
RUN
```

N asterisks per row

M rows

Try this program without the PRINTS in Lines 60 and 80.

(7 rows, 12 asterisks per row)

```
?7,12
************
************
************
************
************
************
************

?
```

Here they are. Seven rows of asterisks with 12 asterisks in each row.

FUNCTION JUNCTION

The Integer (INT) Function

The integer (INT) function returns the value of the nearest integer not greater than x. Let's see how it works.

```
10 PRINT INT(0),INT(1),INT(2),INT(3.14),INT(7.99)
99 END

RUN

 0            1            2            3            7
```

From the results, we see that:

INT(0) = 0 INT(1) = 1 INT(2) = 2
INT(3.14) = 3 INT (7.99) = 7

Here are some rules you can count on.

- If X is a whole number, then INT(X) = X.

  For example,

  INT(0) = 0 INT(1) = 1
  INT(2) = 2 INT(3) = 3

- If X is a positive number or zero, then INT(X) is the whole number part of X.

  For example,

  INT(2.99) = 2 INT(123.45) = 123
  INT(0.75) = 0 INT(.75) = 0

Exercise 39. Complete the following.

(1) INT(4) = _____ (4) INT(27.01) = _____
(2) INT(12345) = _____ (5) INT(12/4) = _____ [4]
(3) INT(0.999999) = _____ (6) INT(13/5) = _____ [4]

[4] Evaluate the numerical expression first, then apply the INT function to the result.

The INT function is useful in solving problems such as the following.

> Let a be a whole number and let b be a natural number. Divide b into a to obtain a whole number partial quotient, q, and a whole number remainder, r.

For example, let $a = 28$ and $b = 7$.

```
   4          Partial quotient, q = 4.
7 ) 28
    28
    0         Remainder, r = 0.
```

Next let's try $a = 97$ and $b = 5$.

```
   19         Partial quotient, q = 19.
5 ) 97
    5
    47
    45
    2         Remainder, r = 2.
```

Once more – this time $a = 29$ and $b = 32$.

```
   0          Partial quotient, q = 0.
32 ) 29
    0
    29        Remainder, r = 29.
```

We can write a mathematical sentence relating a, b, q and r, as follows.

$$a = bq + r \qquad \text{where } 0 \leqslant r < b.$$

For example, $97 = 5 \times 19 + 2$

$a = b \times q + r$ Note that $r < b$.

Let's solve for r: $r = a - bq$

And let's remember that q is the whole number part of a/b. Here is a program to compute q and r given input values of a and b.

```
10 PRINT "A","B","Q","R"
20 READ A,B
30 LET Q=INT(A/B)
40 LET R=A-B*Q
50 PRINT A,B,Q,R
60 GO TO 20
70 DATA 28,7,97,5,29,32
99 END
```

Note how we compute q and r.

```
RUN

A             B             Q             R
 28            7             4             0
 97            5             19            2
 29            32            0             29

DATA ERROR AT LINE 20
```

Exercise 40. Complete the following trace of the RUN of the preceding program.

STATEMENT	A	B	Q	R	OUTPUT			
10 PRINT "A","B","Q","R"					A	B	Q	R
20 READ A,B	28	7						
30 LET Q = INT (A/B)	28	7	4					
40 LET R = A − B*Q	28	7	4	0				
50 PRINT A,B,Q,R	28	7	4	0	28	7	4	0
60 GO TO 10	28	7	4	0				
20 READ A,B								
30 LET Q = INT(A/B)								
40 LET R = A − B*Q								
50 PRINT A,B,Q,R								
60 GO TO 10								
20 READ A,B								
30 LET Q = INT(A/B)								
40 LET R = A − B*Q								
50 PRINT A,B,Q,R								
60 GO TO 10								
and so on.								

Exercise 41. Let x be a 2-digit whole number. That is, x is a whole number such that:

$$10 \leqslant x \leqslant 99$$

We define a number y as follows.

$$y = \text{sum of the digits of } x.$$

For example, if $x = 10$, then $y = 1 + 0 = 1$

if $x = 25$, then $y = 2 + 5 = 7$

if $x = 99$, then $y = 9 + 9 = 18$

Complete the following program to compute y for a given value of x. RUN it for the DATA shown.

```
10 READ X
20
30
40
50
70 PRINT X,Y
80 GO TO 10
90 DATA 10,15,23,37,40,99
99 END
```

Exercise 42. Let z be the number obtained by reversing the digits of x. For example:

if $x = 10$ then $z = 01 = 1$
if $x = 37$ then $z = 73$
if $x = 99$ then $z = 99$

Modify your program of Exercise 41 so that the computer computes the value of z instead of the value of y.

EXPLORE!

Try the INT function on negative numbers (non-integers as well as integers). Can you give a general definition of the INT function?

The Square Root (SQR) Function

We assume that you know how to compute square roots of numbers. Let's find out.

Exercise 43. Complete each of the following:

1. A square root of 4 is _____

2. Another square root of 4 is _____

Here is how we instruct the computer to compute square roots.

```
10 PRINT SQR(4),SQR(25),SQR(100),SQR(0)
99 END

RUN

 2             5             10            0
```

If X is any *non-negative* number (positive or zero), *then* SQR(X) is the non-negative square root of X. If you try to take the square root of a negative number, you may obtain an error message.

```
10 PRINT SQR(-4)
99 END

RUN
```

Note the error message.

```
ARGUMENT ERROR AT LINE 10
```

The symbol SQR( ) which we use in BASIC corresponds to the symbol $\sqrt{\ }$ which we use in mathematics. Perhaps you recall that,

$\sqrt{a}$ is used to mean the non-negative square root of a.

$-\sqrt{a}$ is used to mean the negative square root of a when $a > 0$.

Here is a program to compute the two square roots of a.

```
10 INPUT A
20 PRINT SQR(A),-SQR(A)
30 GO TO 10
99 END

RUN
?4
 2             -2
?4096
 64            -64
?0
 0             0
?2
 1.414214      -1.414214
```

If $a = 0$, there is no negative square root.

These answers are *approximations* to the square roots of 2.

By using the FOR-NEXT loop, you can build your own square root table.

```
10 FOR X=1 TO 10
20 PRINT X,SQR(X)
30 NEXT X
99 END

RUN

 1             1
 2             1.414214
 3             1.732051
 4             2
 5             2.236068
 6             2.44949
 7             2.645751
 8             2.828427
 9             3
 10            3.162278
```

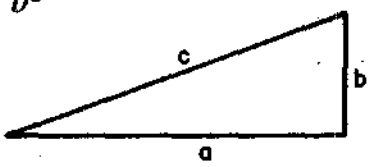
To obtain the square root table of your choice, modify the FOR statement.

Exercise 44. How would you write the FOR statement to obtain a square root table with values of X running from 0 to 1 in STEPS of .2?

If we know the lengths of two sides of a right triangle, we can compute the length of the third side by applying the Pythagorean theorem. For example, suppose c is the length of the hypotenuse and a and b are the lengths of the other two sides, as indicated in the diagram.

Given a and b, we want to compute c.

$$c = \sqrt{a^2 + b^2}$$



Let's use the computer.

```
10 READ A,B
20 LET C=SQR(A↑2+B↑2)
30 PRINT A,B,C
40 GO TO 10
90 DATA 3,4,12,5,1,1                    (Data for 3 triangles)
99 END

RUN

 3            4            5
 12           5            13
 1            1            1.414214

DATA ERROR AT LINE 10
```

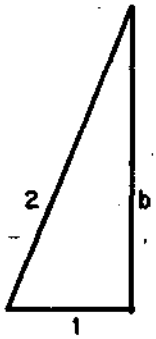
From the results we see that

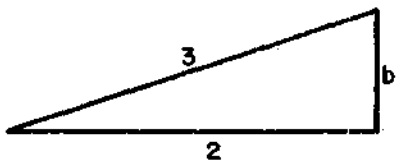
If $a = 3$, $b = 4$ then $c = 5$

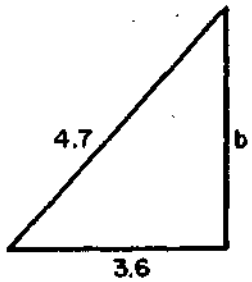
If $a = 12$, $b = 5$ then $c = 13$

If $a = 1$, $b = 1$ then $c = 1.41421$

Exercise 45. Suppose we know the values of c and a. You write the program to compute and print the value of b. Then use your program to obtain the value of b for each of the following.







LOOK BOTH WAYS

LOOK BOTH WAYS

Look back—where have we been?

Loops SCR – LIST INT =

FOR NEXT floating point + RUN

SQR *expressions* ↑ * READ

LET INPUT *messages* PRINT

variables DATA END RESTORE

Look ahead—more BASIC—where will it lead?

IF RND STOP TAN

< >

ABS COS ATN DIM GOSUB

RETURN REM *Arrays* SIN

ON . . . GO TO SGN flow charts EXP

Subscripted variables DEF LOG

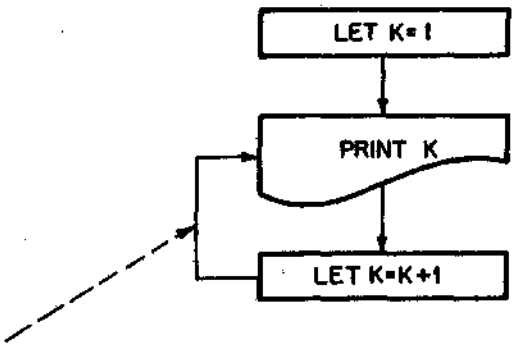
FINDING YOUR WAY

Flowcharts

A flowchart is like a road map. A road map helps you find your way through unfamiliar terrain—a flowchart helps you find your way through a computer program. If you want to use a road map, you must learn how to read it—true also of a flowchart. We will start with easy flowcharts and show you how to read them. Here is a flowchart of a program to generate counting numbers.

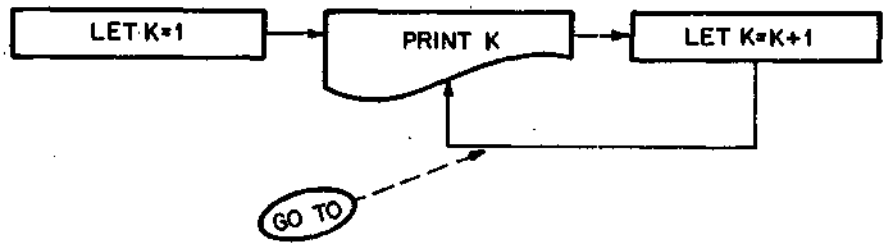
When you read a *program* follow the *line numbers.*	When you read a flowchart, follow the *arrows.*

```
10 LET K=1
20 PRINT K
30 LET K=K+1
40 GO TO 20
99 END
```

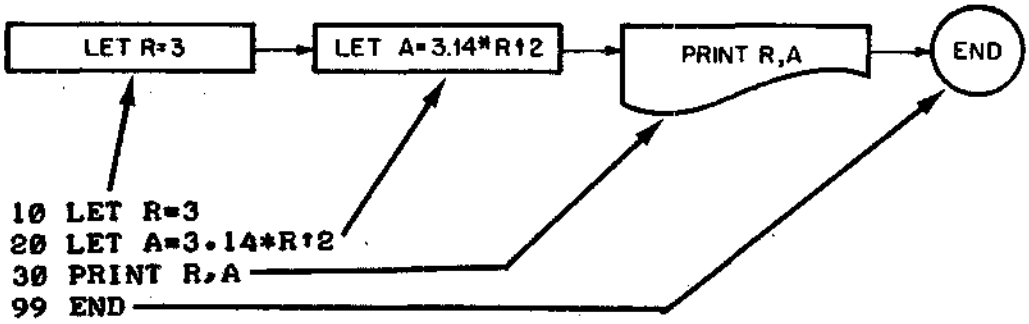


Note that this arrow corresponds to the GO TO statement. In this flowchart there is no box corresponding to the END statement.

If you prefer, you can draw horizontal flowcharts instead of vertical flowcharts.

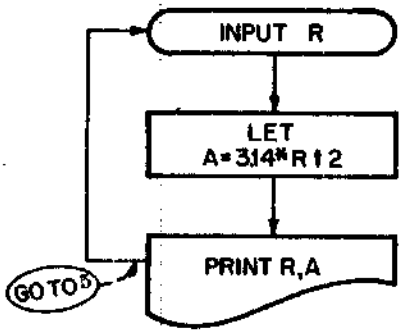


Here is a flowchart with an END box.



In the above flowchart, we showed the END box because it is actually executed in the program, causing the computer to stop.

What next? An INPUT box.




```
10 INPUT R
20 LET A=3.14*R↑2
30 PRINT R,A
40 GO TO 10
99 END
```

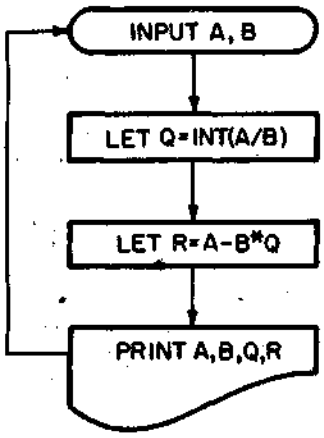
This END is never actually *executed* by the computer; so we don't show it in the flowchart.

Note the *shapes* of the boxes used in the flowchart so far.

---

[5] From now on, we will usually omit this reminder.

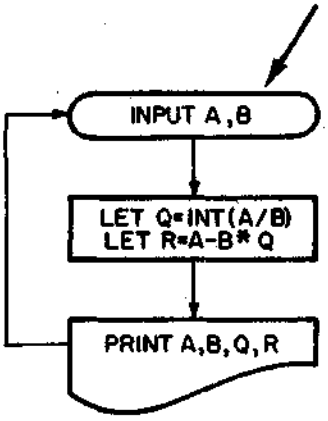
Confused? More examples may help.



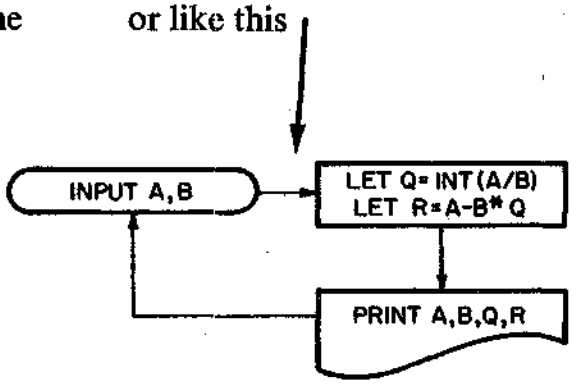

```
13 INPUT A,B
25 LET Q=INT(A/B)
37 LET R=A-B*Q
43 PRINT A,B,Q,R
52 GO TO 13
99 END

RUN

?79,12
 79          12          6           7
?
```

? and so on.

To save space, we might do the above flowchart like this



or like this



We simply put two LET statements in one LET box in order to save space. If there are two or more statements in a box, execute them in order from *top* to *bottom*.

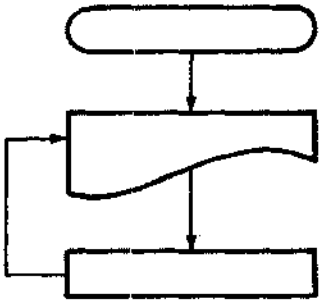
Exercise 46. Below are flowchart boxes in the order in which we introduced them. Note the shape and write the type of box (INPUT, END, LET, etc.) beneath each shape.

_____ _____ _____ _____

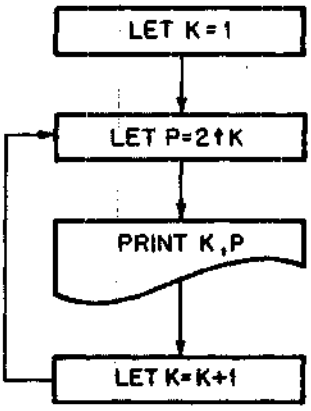
How do we indicate a GO TO statement in a flowchart? _____

Exercise 47. Complete the flowchart of the following program.

```
10 INPUT A,D
20 PRINT A
30 LET A=A+D
40 GO TO 20
99 END
```



Exercise 48. Write the program that corresponds to the following flowchart. Use our line numbers.



12 _____

24 _____

36 _____

48 _____

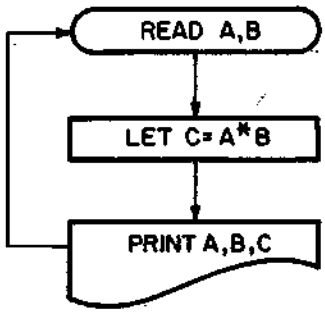
60 _____

99 _____

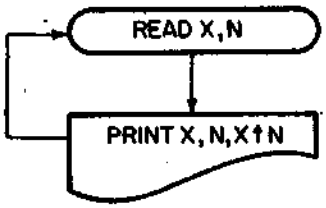
What does the program do? That is, what does it direct the computer to do? Predict the results, then RUN the program on the computer.

We use the same box shape for READ as we do for INPUT. But there is *no* flowchart box corresponding to a DATA statement.

```
10 READ A,B
20 LET C=A*B
30 PRINT A,B,C
40 GO TO 10
90 DATA 3,4,5,-7
99 END

RUN

 3             4             12
 5            -7            -35

DATA ERROR AT LINE 10
```



Let's put the computation directly into the PRINT statement (program) or the PRINT box (flowchart).

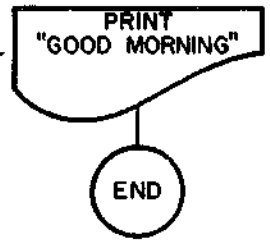



```
10 READ X,N
20 PRINT X,N,X+N
30 GO TO 10
60 DATA 2,3,10,10
99 END

RUN

 2             3             5
 10            10            20

DATA ERROR AT LINE 10
```

Remember: Use  for either INPUT or READ.

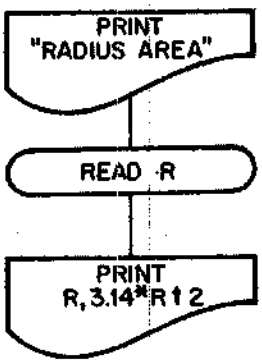
Messages? How do we show them in a flowchart? Easy . . .

```
10 PRINT "GOOD MORNING"
99 END
```



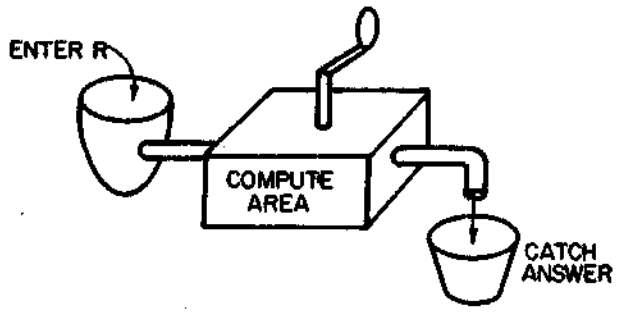
Note the quotation marks.

Another example—




```
10 PRINT "RADIUS","AREA"
20 READ R
30 PRINT R,3.14*R↑2
40 GO TO 20
50 DATA 2,3,8
99 END

RUN

RADIUS          AREA
 2               12.56
 3               28.26
 8               200.96

DATA ERROR AT LINE 20
```

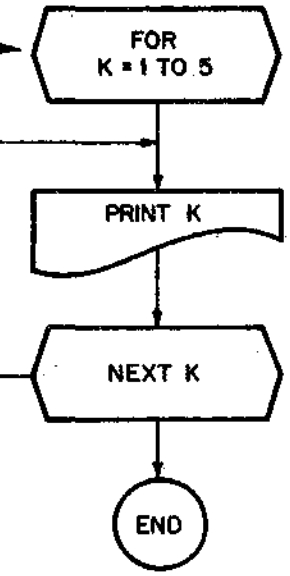
If you don't like our style of flowchart, invent your own. Like . . .



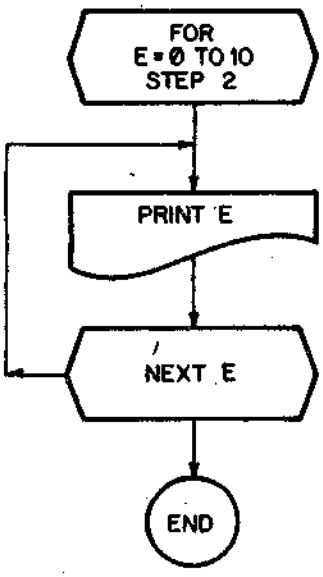
FOR-NEXT Loops in a Flowchart

Remember FOR-NEXT loops? We show them in a special way in a flowchart.

```
10 FOR K=1 TO 5
20 PRINT K;
30 NEXT K
99 END

RUN

 1  2  3  4  5
```



Another one—this time with a STEP clause.

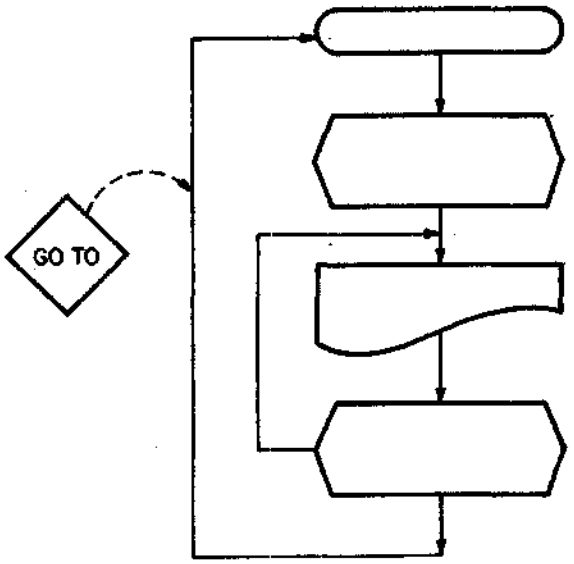



```
10 FOR E=0 TO 10 STEP 2
26 PRINT E
39 NEXT E
99 END

RUN

 0
 2
 4
 6
 8
 10
```

We include the END box in the flowchart above because it is actually executed—stopping the computer after all numbers specified by the FOR-NEXT loop have been printed.

Exercise 49. Complete the flowchart of the following program.

```
10 INPUT A,B,H
20 FOR X=A TO B STEP H
30 PRINT X
40 NEXT X
50 GO TO 10
99 END
```



Exercise 50. Draw a flowchart of the following program. Draw your flowchart here.

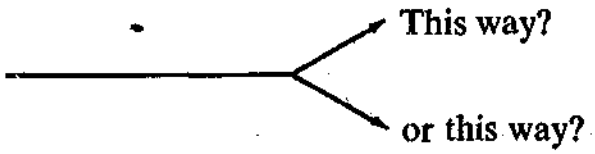
```
10 INPUT N
20 FOR K=1 TO N
30 LET P=K↑K
49 PRINT K,P
50 NEXT K
99 END
RUN

?5
 1            1
 2            4
 3            27
 4            256
 5            3125
```

K ↑ K increases rapidly,
doesn't it?!

MAKING DECISIONS

A Fork in the Road

The road divides into two paths . . . which way?

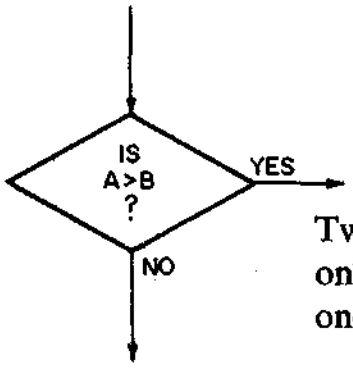


You must decide.

In a flowchart we use a diamond shaped box to indicate a *decision point* based on a *condition.*

One arrow into the box

We write the condition inside the box.



Two arrows out of the box– one labelled "YES" and one labelled "NO"

The condition in the box must have exactly two possible outcomes.

- If the condition in the box is TRUE, follow the arrow labelled YES.
- If the condition is FALSE, follow the arrow labelled NO.

Examine the condition in the decision box above and complete the following table showing which path to follow for given values of *A* and *B*.

Value of *A*	Value of *B*	Which Path? (YES or NO)
3	4	
5	2	
7	7	
0	1	
0	−1	
−2	−1	

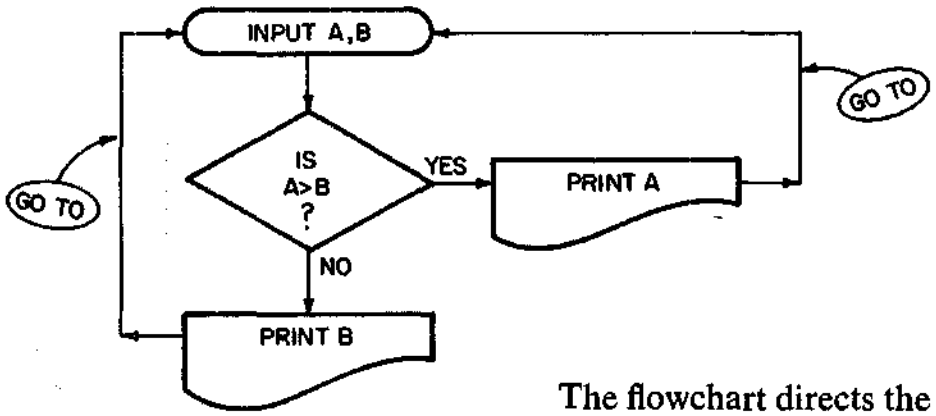
Which is larger, 3 or 4?

Which is larger, 5 or 2?

If $A = 3$ and $B = 4$, which is larger, A or B? _____

If $A = 5$ and $B = 2$ which is larger, A or B? _____

In the flowchart below, trace the path for $A = 3$, $B = 4$ and the path for $A = 5$, $B = 2$. You may wish to mark these paths in two different colors.



The flowchart directs the computer to print the *larger* of two numbers, A and B.

Suppose you mark the path for $A = 3$, $B = 4$ with a red pencil and the path for $A = 5$, $B = 2$ with a green pencil. Which path would we follow (red or green) if

$A = 1, B = 0$? _____ $A = 0, B = 2$? _____

$A = 7, B = 7$? _____ $A = -1, B = -2$? _____

Let's translate the flowchart into a BASIC program.

```
10 INPUT A,B
20 IF A>B THEN 50
30 PRINT B
40 GO TO 10
50 PRINT A
60 GO TO 10
99 END
RUN
?3,4
 4
?5,2
 5
```

This is an IF statement. It corresponds to a decision box in a flowchart.

. . . Follow me if and *only* if A>B.

Prove it! RUN the program.

$A = 3, B = 4.$
B is larger than A.
$A = 5, B = 2.$
A is larger than B.

If $A = 3$ and $B = 4$ the computer executes statements 10, 20, 30 and 40.

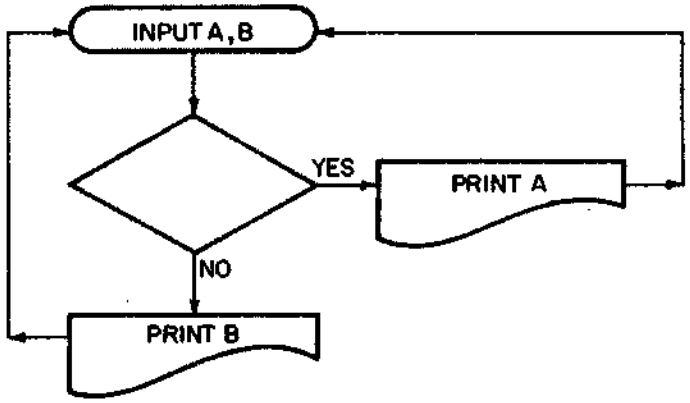
If $A = 5$ and $B = 2$ the computer executes statements 10, 20, 50 and 60.

Let's trace the program. We follow the computer as it executes each statement and show the values of A and B after each statement is carried out. For a PRINT statement, we also show what is printed under the heading "OUTPUT."

STATEMENT	A	B	OUTPUT	REMARKS
10 INPUT A,B	3	4		First case: $A = 3$, $B = 4$.
20 IF A>B THEN 50	3	4		$A>B$ is FALSE.
30 PRINT B	3	4	4	Continue with Statement 30.
40 GO TO 10				Go around again.
10 INPUT A,B	5	2		Second case: $A = 5$, $B = 2$.
20 IF A>B THEN 50	5	2		$A>B$ is TRUE.
50 PRINT A	5	2	5	Go to Statement 50 and proceed.
60 GO TO 10				Go around again.
10 INPUT A,B	7	7		Third case: $A = 7$, $B = 7$.
20 IF A>B THEN 50	7	7		$A>B$ is FALSE.
30 PRINT B	7	7	7	Continue with Statement 30.
40 GO TO 10				
and so on.				

Exercise 51. Complete the following program and flowchart directing the computer to print the *smaller* of two numbers, *A* and *B*.

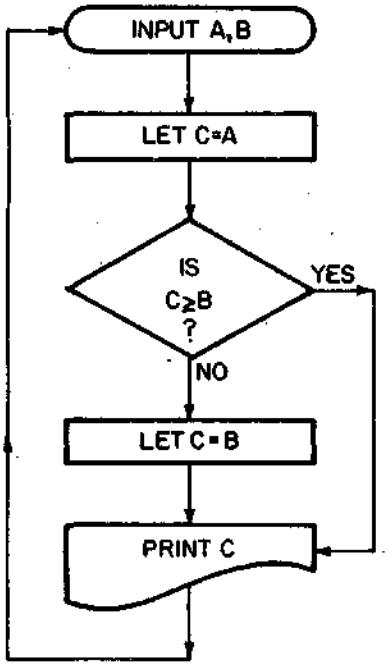
```
10 INPUT A,B
20
30 PRINT B
40 GO TO 10
50 PRINT A
60 GO TO 10
99 END
RUN

?3,4
 3
?5,2
 2
```



Exercise 52. We have written a program directing the computer to print two numbers in *ascending* order with the smaller on the left and the larger on the right. You do the flowchart here.

```
10 INPUT A,B
20 IF A<B THEN 50
30 PRINT B,A
40 GO TO 10
50 PRINT A,B
60 GO TO 10
99 END
RUN
?3,4
 3              4
?5,2
 2              5
```

There is usually another way. In this case, another way to direct the computer to select and print the *larger* of two numbers.



First, enter the two numbers.

Guess that $A > B$ and set $C = A$.

If $C \geqslant B$ is TRUE, our guess is correct. In this case, follow the YES path.

But if $C \geqslant B$ is FALSE, we were wrong. We acknowledge our mistake and set $C = B$.

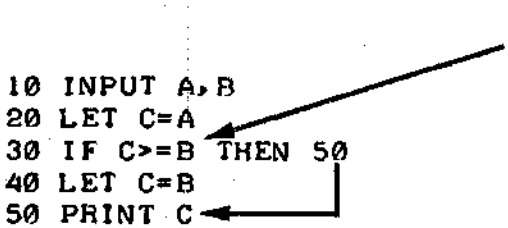
By either path,
$C = \text{MAX}(A, B)$ = maximum of A and B. So we print the value of C.

Again, there are two possible paths through the flowchart. Mark them in two colors. If you mark the path for $A = 3, B = 4$ in green and the path for $A = 5, B = 2$ in red, which path (green or red) should we follow if

$A = 2, B = 1$? _____ $A = 9, B = 10$? _____

$A = 7, B = 7$? _____ $A = 0, B = -1$? _____

O.K., here is the BASIC program to go with the flowchart on the preceding page.



```
10 INPUT A,B
20 LET C=A
30 IF C>=B THEN 50
40 LET C=B
50 PRINT C
60 GO TO 10
99 END
RUN

?3,4
 4
?5,2
 5
```

Note how we write $\geqslant$ in an IF statement.

. . . . . Follow me if and only if $C \geqslant B$ is TRUE.

$A = 3, B = 4.$
$C = 4.$

$A = 5, B = 2.$
$C = 5.$

Your turn . . . complete the following *trace* of the above program.

STATEMENT	*A*	*B*	*C*	OUTPUT	REMARKS
10 INPUT A,B	3	4			
20 LET C = A	3	4	3		
30 IF C >= B THEN 50	3	4	3		$C \geqslant B$ is FALSE.
40 LET C = B					*You* carry on.
50 PRINT C					
60 GO TO 10					
10 INPUT A,B	5	2			
20 LET C = A	5	2	5		
30 IF C >= B THEN 50	5	2	5		TRUE or FALSE? Where to next?

The IF Statement

The IF statement directs the computer to examine a *relation* between two expressions and then follow one (and only one) of two paths depending on whether the relationship is TRUE (YES) or FALSE (NO).

In general, the IF statement looks like this:

n IF e_1 r e_2 THEN t

where
- n = line number of the IF statement
- e_1 = any BASIC expression
- e_2 = any BASIC expression
- r = any legal BASIC *relation* (see below)
- t = line number of the statement executed next if and only if e_1 r e_2 is TRUE

Relation between two expressions, e_1 and e_2

For example,

35	IF	X	=	INT(X)	THEN	60
↑	↑	↑	↑	↑	↑	↑
n	IF	e_1	r	e_2	THEN	t

X = INT(X)
- is TRUE if X is an integer.
- is FALSE if X is not an integer.

Here is a handy table showing BASIC relations and corresponding mathematical ralations.

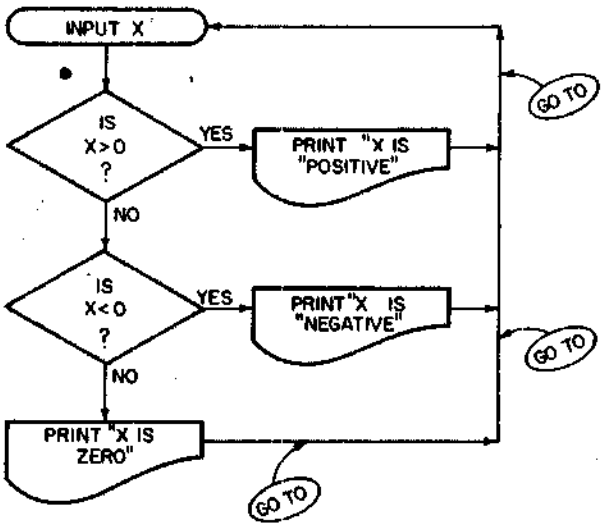
MATH	BASIC	RELATION
=	=	Equal to
<	<	Less than
>	>	Greater than
⩽	<=	Less than or equal to
⩾	>=	Greater than or equal to
≠	<>	Not equal to

A Problem. Write a program directing the computer to tell us whether a number is positive, negative or zero.

Let's solve it. Since there are three possibilities ($X > 0$, $X < 0$, or $X = 0$), we can't solve the problem with a single IF statement. But we can do it with *two* IF statements.

For example,

If $X > 0$ is FALSE, then $X < 0$ or $X = 0$. Which one?



There are three possible paths through the program. Mark them in three colors.

What color path if $X = 7$? _____

What color path if $X = -3$? _____

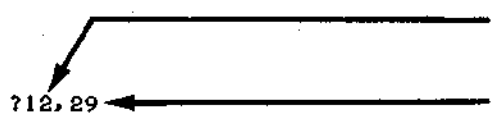
What color path if $X = 0$? _____

Exercise 53. Write a BASIC program to go with our flowchart. RUN it for $X = 7$, $X = -3$ and $X = 0$.

VARYING PATTERNS

Rectangular Patterns

We wrote a program to print rectangular patterns on the Teletype. We ran it . . . here is what happened.




```
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
*****************************
```

Our program lets us specify the number of *rows* and *columns* in this rectangular pattern.

This pattern has 12 rows and 29 columns. That is, it has 12 rows and there are 29 asterisks in each row.

Using our program, we can direct the computer to print rectangular patterns with as many rows as we want. The number of columns, however, is limited to 72 since there are only 72 printing positions across the paper.

In the above pattern, there are

$$12 \times 29 = 348$$

asterisks in all.

Let's RUN the program again.

```
?3, 10
```

This time we want 3 rows, 10 columns.

```
**********
**********
**********
```

Here they are . . . count them.

Now let's look at the program.

Below is part of the program. This part directs the computer to print one row of asterisks with N asterisks in the row.

```
10 INPUT N
15 PRINT
```

N specifies the number of asterisks to be printed in the row.

```
  30 FOR C=1 TO N
  40 PRINT "*";
  50 NEXT C
```

Here is the heart of the program. It causes the computer to print N asterisks, *all in one row* because the PRINT statement ends with a semicolon.

```
60 PRINT
80 GO TO 10
99 END
```

This PRINT causes a carriage return and line feed to occur *after* all the asterisks have been printed.

```
RUN
```

Let's try it.

```
?10
**********
```

$N = 10$, so the computer prints a row with 10 asterisks.

```
?20
```

$N = 20$.

```
**********
```

How many asterisks?

```
?72
```

$N = 72$. Computer prints 72 asterisks.

$N = 73$ . . . can't get them all on one line.

```
************************************************************************
?73
************************************************************************
*
```

The 73rd asterisk ended up here.

The computer will print *up to* 72 asterisks on one line . . . but no more. If $N > 72$ then the computer prints 72 asterisks on the first line and continues on the next line. Suppose $N = 200$. How many lines will it take to print the "row" of 200 asterisks?

By adding three statements and changing the INPUT statement in the preceding program, we get a program to print a pattern consisting of M rows with N asterisks in each row.

```
10 INPUT M,N
15 PRINT
  20 FOR R=1 TO M
    30 FOR C=1 TO N
    40 PRINT "*";
    50 NEXT C
  60 PRINT
  70 NEXT R
```

Which lines did we add?

One row with N asterisks

M rows

This program has a "loop within a loop."

```
75 PRINT
80 GO TO 10
99 END
RUN
```

Does it work? RUN it.

```
?5,12
```

We enter $M = 5$, $N = 12$.

```
************
************
************
************
************
```

The computer prints a rectangular pattern with 5 rows and 12 columns.

```
?1,7
```

A 1 by 7 pattern (1 row, 7 columns).

```
*******
```

```
?7,1
```

And a 7 by 1 pattern (7 rows, 1 column).

```
*
*
*
*
*
```

```
?1,1
```

We finish with a 1 by 1 pattern.

```
*
```

```
?0,0
```

P.S. How about a null (empty) pattern?

Exercise 54. If you are tired of seeing asterisks before your eyes, change this statement

```
40 PRINT "*";
```

to this

```
40 PRINT "X";
```

or this

```
40 PRINT "%";
```

or to the statement of your choice. Then RUN the program again.

Exercise 55. Start with our program on page 1-83. Change the INPUT statement to

```
10 INPUT M
```

and change *one* other statement so that the modified program will print right triangular patterns, like these.

```
RUN
?2
*
**
```

This pattern has 2 rows with 1 asterisk in Row 1 and 2 asterisks in Row 2.

```
?3
*
**
***
```

This pattern has 3 rows with 1 asterisk in Row 1, 2 asterisks in Row 2 and 3 asterisks in Row 3.

```
?4
*
**
***
****
```

This pattern has 4 rows with 1 asterisk in Row 1, . . .

and so on.

For an INPUT value of *M*, the pattern has *M* rows with 1 asterisk in Row 1, two asterisks in Row 2, three asterisks in Row 3, . . ., *M* asterisks in Row *M*.

In the following program, we use a "brute force" approach to tell the computer to print a "leaning tower" pattern.

```
10 PRINT "********"
20 PRINT " ********"
30 PRINT "  ********"
40 PRINT "   ********"
50 PRINT "    ********"
60 PRINT "     ********"
70 PRINT "      ********"
80 PRINT "       ********"
99 END
```

spaces

Each PRINT statement tells the computer to print a certain number of spaces followed by 8 asterisks.

```
RUN
********
 ********
  ********
   ********
    ********
     ********
      ********
       ********
```

spaces

You can use the brute force method to print just about any pattern you want.

Let's make the tower lean the other way.

```
10 PRINT "         ********"
20 PRINT "        ********"
30 PRINT "       ********"
40 PRINT "      ********"
50 PRINT "     ********"
60 PRINT "    ********"
70 PRINT "   ********"
80 PRINT "  ********"
99 END
RUN
         ********
        ********
       ********
      ********
     ********
    ********
   ********
  ********
 ********
```

Print 7 spaces, then 8 asterisks.
Print 6 spaces, then 8 asterisks.
And so on.

The TAB Function

Yes, we're leading up to something . . . here is another way to do the first leaning tower pattern.

```
10 PRINT TAB(0);"********"
20 PRINT TAB(1);"********"
30 PRINT TAB(2);"********"
40 PRINT TAB(3);"********"
50 PRINT TAB(4);"********"
60 PRINT TAB(5);"********"
70 PRINT TAB(6);"********"
80 PRINT TAB(7);"********"
99 END
RUN
********
 ********
  ********
   ********
    ********
     ********
      ********
       ********
```

We wrote this program primarily to introduce the TAB function.

The TAB function causes the computer to space over to the printing position enclosed in the ( ).

We hope your version of BASIC provides the TAB function . . . if not, you might as well skip the rest of this section.[6] If your computer *does* provide the TAB function, read on!

EXAMPLE	EXPLANATION
TAB(7)	Space the teletypewriter over to printing Position 7.
TAB(25)	Space the teletypewriter over to printing Position 25.

There are 72 printing positions across the page . . . they are numbered 0 through 71.

```
************************************************************************
```

Position 0 Position 37 Position 71

Your turn . . . circle printing Positions 10, 20, 30, 40, 50, 60 and 70 above.

[6] EduSystem 10 does not include the TAB function; larger systems do.

In general, the TAB function looks like this

TAB(e)

where e is any BASIC expression. The expression is evaluated and the computer spaces the Teletype to the position specified by the value of the expression. If the value of e is not a whole number, then the computer spaces the teletype to the *whole number part* of the value of e.

EXAMPLES	EXPLANATION
TAB(K)	Move to Position K. $(0 \leqslant K \leqslant 71)$
TAB(R − 1)	Move to Position $R-1$. $(0 \leqslant R-1 \leqslant 71)$
TAB (2*X + 3)	Move to Position $2*X + 3$. $(0 \leqslant 2*X + 3 \leqslant 71)$

Here is yet another way to print the leaning tower pattern.

```
10 FOR R=1 TO 8
20 PRINT TAB(R-1);"********"
30 NEXT R
99 END
RUN
********
 ********
  ********
   ********
    ********
     ********
      ********
       ********
```

Space over to printing Position $R - 1$. When $R = 1$, this is Position 0; when $R = 2$, this is position 1; and so on.

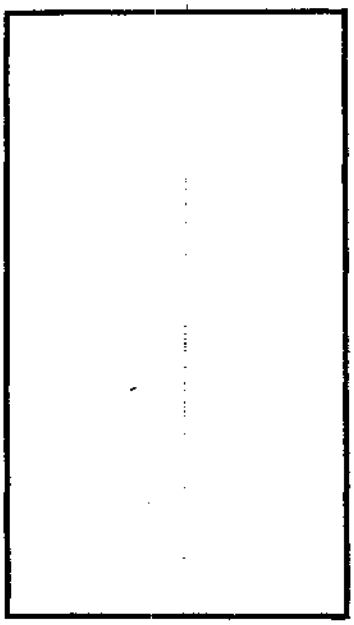
The computer cannot space the Teletype ← to the left. So, if it is already at Position 30, you cannot space it to Position 20.

There are no negative printing positions. TAB(−1) is ignored. TAB (K) is ignored if $K < 0$.

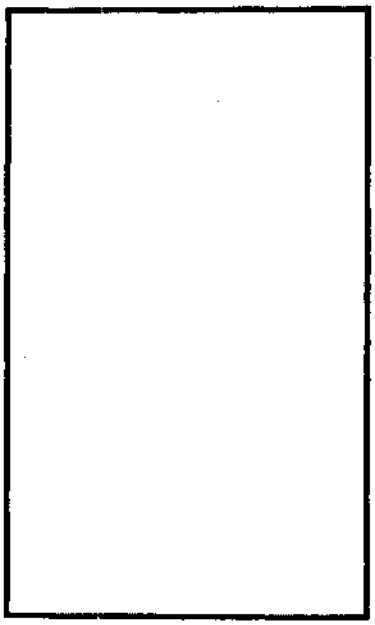
Exercise 56. By changing *one* statement in the preceding program you can get a program to print the leaning tower in which the top leans to the right. Do it! Then RUN it.

Exercise 57. Here are two programs. Read them. Figure out what they do. *Without using the computer* write down what you think the computer will print. Then (and not until then) RUN the programs. (Were you correct?)

```
20 FOR R=1 TO 8
30 PRINT TAB(R-1);
40 FOR C=1 TO 8
50 PRINT "*";
60 NEXT C
70 PRINT
80 NEXT R
99 END
```



```
20 FOR R=1 TO 8
30 PRINT TAB(8-R);
40 FOR C=1 TO 2*R-1
50 PRINT "*";
60 NEXT C
70 PRINT
80 NEXT R
99 END
```



Exercise 58. By adding an INPUT statement and changing a few things in the programs of the preceding exercise, you can get a program that will let you specify the size of the pattern. Try it.

Exercise 59. Use the brute force method (page 1-85) to cause the computer to print a pattern of your choice: SNOOPY, a dove, a flower, you name it.

MEANDERING

Random Numbers

Random numbers—what are they? According to one definition, a random number is a number that is chosen by chance from a given set of numbers.

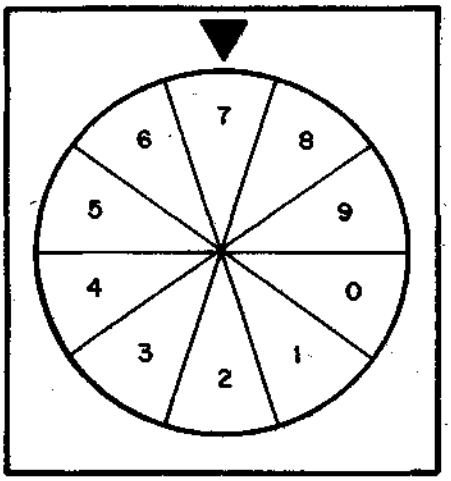
- We flipped a coin ten times. If it came up HEADS, we wrote "1" and if it came up TAILS we wrote "0."
  Here is what happened: 1 0 1 1 1 0 1 0 0 1
- We rolled a die 20 times. Each time we wrote down the number of spots showing on top.
  Here is what happened: 5 2 1 5 3 3 4 2 1 4
  6 2 5 1 5 1 3 2 6 1

In each case we got a *random sequence* of numbers. Each number in the sequence was *selected at random* from a given set of numbers.

- When we flip a coin, we select numbers at random from the set (0,1).
- When we roll a die, we select numbers at random from the set (1,2,3,4,5,6).

By "select at random" we simply mean that we use a selection process in which each member of the set has the same chance of being selected as any other member of the set. That is, the *probability* of selecting any member of the set is the same as the probability of selecting any other member.

We can obtain a random sequence of numbers from the set (0,1,2,3,4,5,6,7,8,9) by using a spinner like the one pictured below.



Spin the wheel . . . select the number at which it stops.

We show the wheel stopped at seven.

There is a BASIC function called RND. It generates "random numbers." Here is a sequence of 50 random numbers.

```
10 FOR K=1 TO 50
20 PRINT RND(0),
30 NEXT K
99 END
RUN
 .3481899     .9928119     .8231623     3.666706E-03 .6135398
 .6482347     .3675558     .371222      .91933       .1749821
 .7759228     .08069808    .5008833     .2790171     .1661529
 .4857633     .4192038     .1433537     .08728769    .2335427
 .6156673     .5921191     .01170888    .7411813     .341708
 .3796163     .2023254     .7974058     .9635064     .6043865
 .9547609     .2890875     .1416765     .2482717     .2145417
 .05280478    .3859534     .8404774     .5692836     .8514056
 .9848808     .2466345     .61588       .4755698     .3104984
 .5828625     .7026891     .9703719     .4980298     .2548316
```

Every number in the random sequence is *greater than zero* but *less than one*. In other words,

$$0 < \text{RND}(0) < 1.$$

Every time the computer evaluates RND(0), it generates another random number between zero and one. In the above program, RND(0) occurred in a FOR-NEXT loop and was evaluated 50 times. Therefore, 50 random numbers were printed.

BEWARE!

If you RUN our program,
don't expect to obtain the
same results.

But what if we want a random sequence in which each number in the sequence is zero or one? Here is one way.

```
10 FOR K=1 TO 20
20 PRINT INT(2*RND(0))
30 NEXT K
99 END
RUN
 0  1  1  0  0  0  1  1  0  0  1  0  1  1  1  0  1  1  1  0
```

Print the *integer* part of *twice* the random number.

Why does the computer print only zeros and ones? Because, if you recall,

$$0 < \text{RND}(0) < 1.$$

Therefore, $0 < 2*\text{RND}(0) < 2$

and INT(2*RND(0)) is either 0 or 1.

The RND function is useful when we want to use the computer to *simulate* (imitate) a real life activity in which chance plays a part. Since we can flip a coin to obtain random numbers, why not use random numbers to simulate coin flipping?

```
10 FOR K=1 TO 20
20 LET R=INT(2*RND(0))
30 IF R=1 THEN 60
40 PRINT "TAILS",
50 GO TO 70
60 PRINT "HEADS",
70 NEXT K
99 END
RUN
HEADS        HEADS        HEADS        TAILS        HEADS
TAILS        TAILS        HEADS        HEADS        HEADS
HEADS        TAILS        TAILS        HEADS        HEADS
TAILS        TAILS        HEADS        HEADS        HEADS
```

R is either 0 or 1.

If $R = 1$, "HEADS" is printed. Otherwise ($R = 0$), "TAILS" is printed.

REMEMBER . . . If you RUN our program you will probably get a different set of results.

Exercise 60. Write a program to simulate *N* tosses of a coin, where the value of *N* is entered in response to an INPUT statement. Here is a RUN of *our* program for this exercise.

```
RUN
?10
HEADS          HEADS          TAILS          HEADS          TAILS
TAILS          HEADS          TAILS          HEADS          HEADS
READY.
```

and so on.

Exercise 61. *The possibility set* for INT(2*RND(0)) is (0,1). The possibility set is the set of all *possible* values of the complete expression. What is the possibility set for each of the following?

(1) INT(3*RND(0)) _____ (2) 3*INT(RND(0)) _____

(3) INT(6*RND(0)) _____

(4) INT(6*RND(0)) + 1 _____

(5) INT(10*RND(0)) _____

(6) INT(10*RND(0))/10 _____

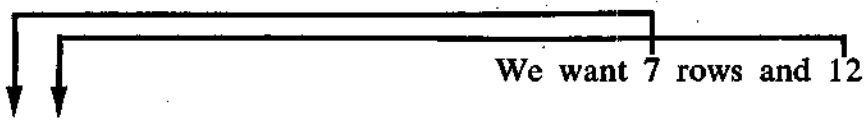
Exercise 62. Simulate something. Write a program to generate a random sequence that imitates a real life activity—like rolling one die, rolling two dice, spinning a spinner like the one shown on page 1-89, a game of chance, or you name it!

Constellations

Remember this program? (See page 1-83)

```
10 INPUT M,N
20 FOR R=1 TO M
30 FOR C=1 TO N
40 PRINT "*";
50 NEXT C
60 PRINT
70 NEXT R
75 PRINT
80 GO TO 10
99 END
RUN
```

This program prints an M by N rectangular pattern of "stars."

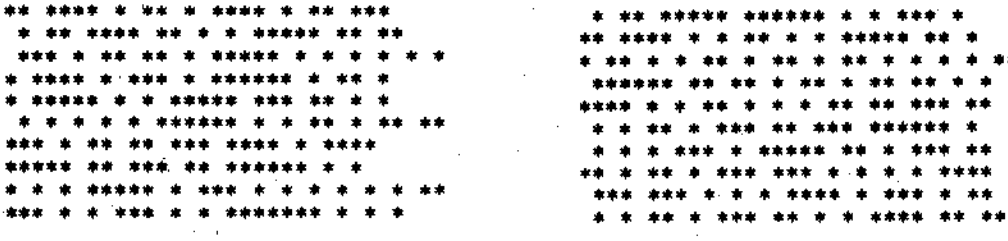


We want 7 rows and 12 columns.

```
?7,12
************
************
************
************
************
************
************
```

This pattern uses $7 \times 12 = 84$ printing positions. A "star" is printed in each printing position.

Now suppose that we print an $M \times N$ pattern but instead of printing a star (*) in each position, we flip a coin . . . if it comes up HEADS we print a star . . . if it comes up TAILS we print a *space*. Here are some 10×20 patterns obained in this manner.



O.K., we didn't actually sit at the Teletype and flip coins. Instead, we added some statements to our program to generate rectangular patterns and let the computer do the work. Here it is.

```
10 INPUT M,N
15 PRINT
20 FOR R=1 TO M
30 FOR C=1 TO N
35 IF INT(2*RND(0))=1 THEN 50
40 PRINT " ";
50 PRINT "*";
60 NEXT C
65 PRINT
70
70 NEXT R
75 PRINT
80 GO TO 10
99 END
RUN
```

We let the computer "flip" the coin. If it comes up HEADS (1) then print a "star"; otherwise print a space.

How many stars were printed in all?

```
?10,20

** * * * ** ** ** ** *** * * **
** * * * ** *** *** *** ** * *
* * * * ** *** * ** *** * * * * *
 * * ** * * ** * * ******** * *
 *** ** ***** ** ** ** ** * *
 * * ** * ** ** * * * * **** * * *
**** *** ** ***** ** ** * *
 * ** ** ** * * * * ** *** ****
 ***** * * ****** ** * * ***
 * * * * * * * *** ** * *** ** * *
```

The *probability* of printing a star is ½. The *probability* of printing a space is ½.

Let's look more closely at statement 35.

```
35 IF  INT(2*RND(0)) = 1 THEN 50
```

The possible values are 0 and 1.

The following statement will also do the job. Try it!

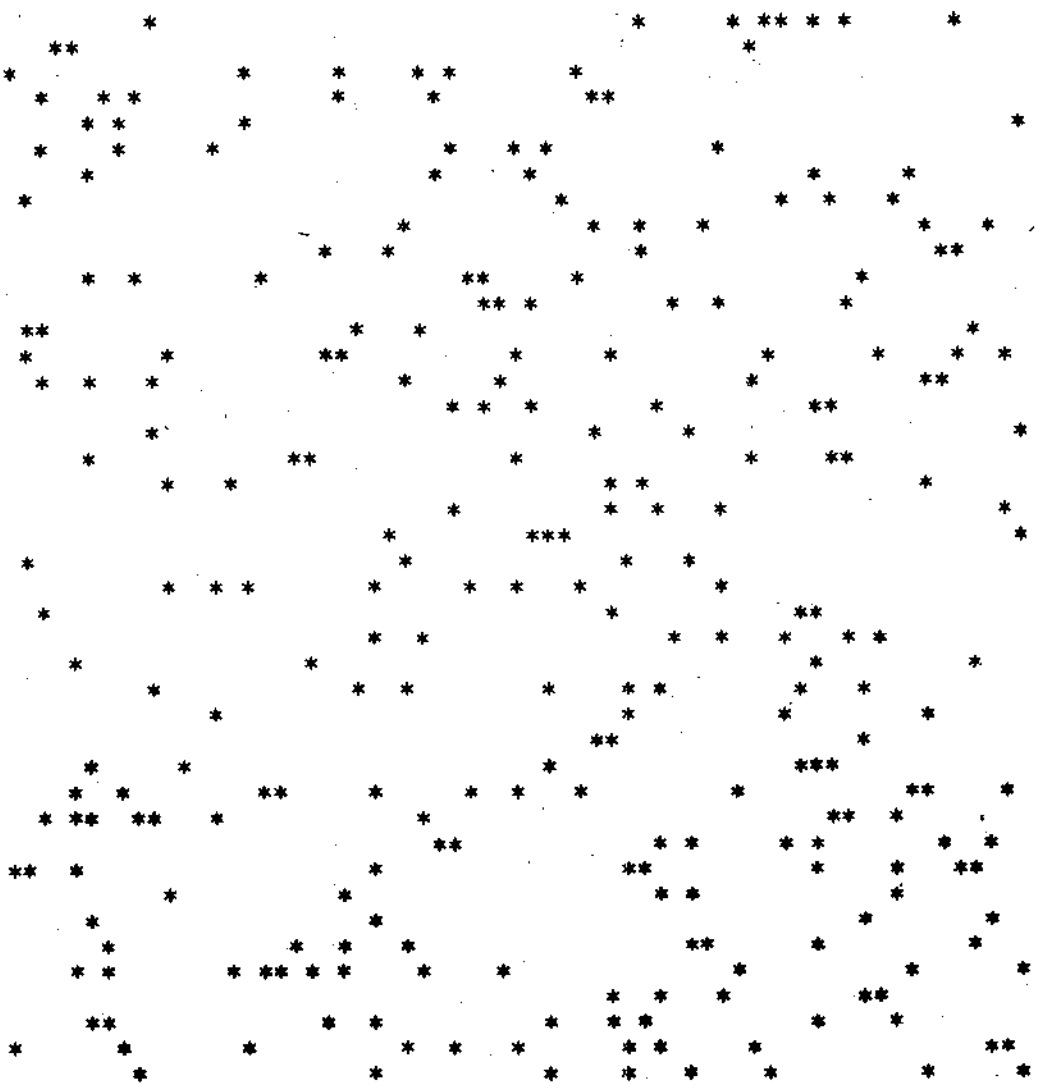
```
35 IF  RND(0) < 1/2 THEN 50
```

Because

RND(0) < .5 is TRUE about ½ the time.
RND(0) < .5 is FALSE about ½ the time.

In the following pattern the probability of printing a star is 1/10 and the probability of printing a space is 9/10. LOOK FOR CONSTELLATIONS . . . MAKE UP YOUR OWN . . . ARE ANY SIGNS OF THE ZODIAC HERE?

```
?42,65
```



Did you look at it from the left? The right? The top?

Exercise 63. Write a program to generate random patterns like the one on the preceding page. You can do it by changing *one* statement in our program on page 1-94. RUN it—look for constellations.

Exercise 64. Modify your program of the preceding exercise so that the probability of printing a star is 1/20 and the probability of printing a space is 19/20. RUN it—look for constellations.

Exercise 65. Why not enter the probability of printing a star along with *M* and *N*? Begin your program with the statement

```
10 INPUT M, N, P
```

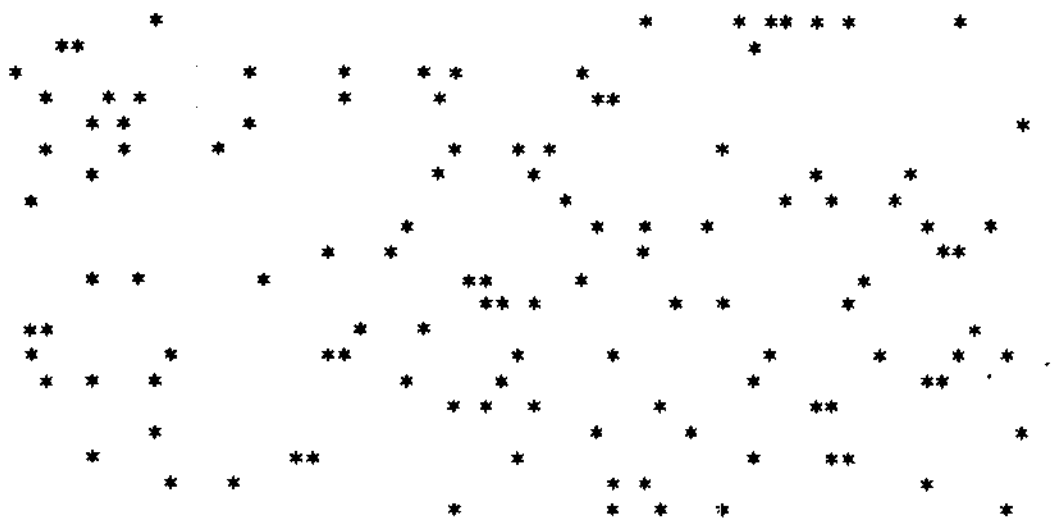
where

M is the number of rows,
N is the number of columns, and
P is the probability of printing a star.

But note: *P* must be entered as a decimal numeral such as .5 or .1 or .05.

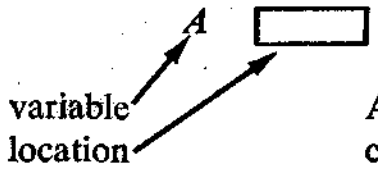
Here is a RUN of our program.

```
?20,65,.1
```



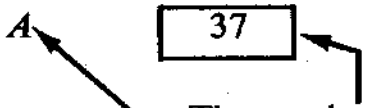
LITTLE BOXES

Remember . . . in BASIC, a variable refers to a *location* in the computer's memory. You can think of the variable as the name of the location.



A location is a "box" in which we can store a number.

Here is a box (location) with a number in it.



The number in the location is the *value* of the variable. (The value of A is 37.)

Subscripted Variables

Are you acquainted with subscripted variables?

Here is a subscripted variable → x_3

This is the subscript

The symbol 'x_3' *is read* '*x* sub 3.'

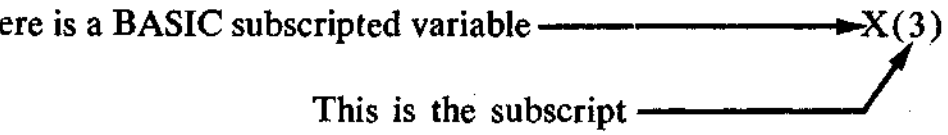
Here are some additional examples of subscripted variables.

p_0	c_7	
k_8	x_1	y_4
a_1	x_2	z_2
r_2	x_3	z_1

w_9

Read 'a_1' as '*a* sub 1'
Read 'c_7' as '*c* sub 7'
Read 'r_2' as '*r* sub 2'
and so on.

In BASIC, we can use subscripted variables, but we write them in a slightly different way. Like this:



The subscripted variables $X(1)$, $X(2)$ and $X(3)$ each correspond to a location

Exercise 66. LET X(1) = 73. In other words, take pencil in hand and write the numeral "73" in the box labelled "$X(1)$." Then do the following in similar fashion.

(1) LET X(2)= 67 (2) LET X(3) = 85

Suppose that $X(1)$, $X(2)$ and $X(3)$ are scores obtained on three quizzes. Here is a program to compute and print the average score (the arithmetic mean of the three scores).

```
10 LET X(1)=73
12 LET X(2)=67
14 LET X(3)=85
20 LET S=X(1)+X(2)+X(3)
30 LET M=S/3
40 PRINT M
99 END
RUN
 75
```

Quiz 1. The score is 73.

Quiz 2. The score is 67.

Quiz 3. The score is 85.

Add the three scores, then divide the sum by 3 and print the result.

Yes, we know you can write a shorter program . . . but did you learn anything about subscripted variables?

Here is a better program.

```
10 INPUT X(1),X(2),X(3)
20 LET S=X(1)+X(2)+X(3)
30 LET M=S/3
40 PRINT M
50 PRINT
60 GO TO 10
99 END
RUN
?73, 67, 85
 75
```

With this program, we INPUT the three scores.

X(1) 73

X(2) 67

X(3) 85

? ← O.K., enter *your* scores.

Exercise 67. Complete the following program to compute and print the arithmetic mean (average) of *four* scores:

X(1), X(2), X(3) and X(4)

```
10 INPUT _____
20 LET S= _____
30 LET M= _____
40 PRINT M
50 PRINT
60 GO TO 10
99 END
RUN

?  _____
   _____

?  _____
   _____
```

RUN your program for the following sets of data. Each set consists of four scores.

73, 67, 85, 83
82, 88, 97, 90

SHOW THE RESULTS OF THE RUN

Next, let's change our program so that the data is called in by means of a READ statement instead of an INPUT statement.

```
10 PRINT "X(1)","X(2)","X(3)","MEAN"
15 READ X(1),X(2),X(3)
20 LET S=X(1)+X(2)+X(3)
30 LET M=S/3
40 PRINT X(1),X(2),X(3),M
60 GO TO 15
80 DATA 73,67,85
81 DATA 82,88,97
82 DATA 93,89,95
83 DATA 77,71,67
99 END
RUN
```

Four sets of data. Each set has three scores.

```
X(1)          X(2)          X(3)          MEAN
 73            67            85            75
 82            88            97            89
 93            89            95            92.33333
 77            71            67            71.66667

DATA ERROR AT LINE 15
```

Exercise 68. Change the above program to a program to compute and print the mean of *four* numbers $X(1)$, $X(2)$, $X(3)$, and $X(4)$. Show the modified program below along with a RUN for the following sets of data:

73, 67, 85, 83
82, 88, 97, 90

Generalizing

But what if there are five scores or six scores or seven scores? Shall we write a separate program for each case? Why not write *one* program that takes care of any number of scores . . . let's try.

We want to write a program to compute the arithmetic mean of N numbers (quiz scores, golf scores, measurements . . . you name it). Since we have N numbers, let's call them

$$X(1), X(2), ..., X(N).$$

We tried the following program but it didn't work. We kept getting error messages.

```
10 READ N
20 READ X(1),X(2),...,X(N)
30 LET S=X(1)+X(2)+...,LIST
10 READ N
20 READ X(1),X(2),000,X(N)
30 LET S=X(1)+X(2)+000+X(N)
40 LET M=S/N
50 PRINT N,S,M
60 GO TO 10
70 DATA 3,73,67,85
71 DATA 3,82,88,97
72 DATA 4,73,67,85,83
73 DATA 5,66,78,71,82,75
99 END
```

Our computer didn't like these two statements.

Note that we want N, S and M printed, but not the Xs.

Each DATA statement has the value of N followed by the N values of

$$X(1), X(2), \ldots, X(N).$$

For example

```
92 DATA 4, 73, 67, 85, 83
        N  X(1) X(2) X(3) X(4)
```

Our program above didn't work because we did not work within the rules of the BASIC language. Read on!

Variable Subscripts

The subscripted variable ⟶ X(K)

has a variable subscript ⟶ K

If K = 1, then X(K) is X(1).
If K = 2, then X(K) is X(2).
If K = 3, then X(K) is X(3).

Got the idea? Let's find out.

A(1)	8	B(1)	3.7	I	1
A(2)	−6	B(2)	9.2	J	2
A(3)	10	C(1)	3	K	3
A(4)	13	C(2)	4	X	4

Write the value of each variable below.

A(1) = _____ I = _____ A(I) = _____

A(2) = _____ J = _____ A(J) = _____

A(K) = _____ A(X) = _____ B(I) = _____

B(J) = _____ C(1) = _____ C(J) = _____

Ready for some slightly more difficult ones?

A(I + 1) = _____ A(I + 2) = _____ A(I + 3) = _____

A(2*I) = _____ A(2*J − 1) = _____ A(X − 3) = _____

A(I + J) = _____ A(X − K + J) = _____ A(J*K − X) = _____

Here is a program to compute and print the sum and mean of N numbers $X(1), X(2), ..., X(N)$.

```
10 READ N
20 FOR K=1 TO N
23 READ X(K)
27 NEXT K
30 LET S=0
33 FOR K=1 TO N
35 LET S=S+X(K)
37 NEXT K
40 LET M=S/N
50 PRINT N,S,M
60 GO TO 10
70 DATA 3,73,67,85
71 DATA 3,82,88,97
72 DATA 4,73,67,85,83
73 DATA 5,66,78,71,82,75
99 END
RUN
 3             225           75
 3             267           89
 4             308           77
 5             372           74.4

DATA ERROR AT LINE 10
```

READ X(1), X(2), . . . , X(N)

LET S = X(1) + X(2) + . . . + X(N)

Here is a partial trace of the program, using the first set of data (the data in Line 70).

STATEMENT	N	K	$X(1)$	$X(2)$	$X(3)$
10 READ N	3				
20 FOR K = 1 TO N	3	1			
23 READ X(K)	3	1	73		
27 NEXT K	3	2	73		
23 READ X(K)	3	2	73	67	
27 NEXT K	3	3	73	67	
23 READ X(K)	3	3	73	67	85
27 NEXT K	3	4	73	67	85
		$K > N$. Therefore the loop terminates.			

Exercise 69. Let's continue the trace that we began on the preceding page. We will show the statements; *you* fill in the *K*, *S* and *M* columns. Remember . . . we left $X(1)$, $X(2)$ and $X(3)$ as follows.

X(1)	73	X(2)	67	X(3)	85

STATEMENT	*N*	*K*	*S*	*M*	OUTPUT
30 LET S = 0	3				
33 FOR K = 1	3				
35 LET S = S + X(K)	3				
37 NEXT K	3				
35 LET S = S + X(K)	3				
37 NEXT K	3				
35 LET S = S + X(K)	3				
37 NEXT K	3				
40 LET M = S/N	3				
50 PRINT N, S, M	3				
60 GO TO 10	3				
and so on.					

Exercise 70. For each program segment (portion of a program) fill in the affected locations.

```
10 FOR N=1 TO 4
20 LET P(N)=2↑N
30 NEXT N
```

P(1) ☐ P(2) ☐

P(3) ☐ P(4) ☐

```
70 LET F(1)
75 FOR K=2 TO 6
80 LET F(K)=K*F(K-1)
85 NEXT K
```

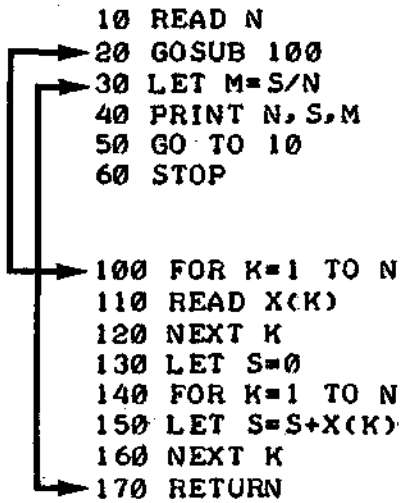
F(1) ☐ F(2) ☐

F(3) ☐ F(4) ☐

F(5) ☐ F(6) ☐

Subroutines

A new idea—subroutines—and three new statements, GOSUB, RETURN, and STOP. Let's use them in a rewrite of the program on page 1-103.



```
10 READ N
20 GOSUB 100
30 LET M=S/N
40 PRINT N, S, M
50 GO TO 10
60 STOP


100 FOR K=1 TO N
110 READ X(K)
120 NEXT K
130 LET S=0
140 FOR K=1 TO N
150 LET S=S+X(K)
160 NEXT K
170 RETURN
```

main program

subroutine

```
300 DATA 3, 73, 67, 85
310 DATA 3, 82, 88, 97
320 DATA 4, 73, 67, 85, 83
330 DATA 5, 66, 78, 71, 82, 75
999 END
```

data base

This program has a main program, one subroutine and a data base. The subroutine in lines 100-170 is called by the GOSUB in line 20.

Now, RUN the above program. The results should be the same as for the program on page 1-99.

When the GOSUB in line 20 calls the subroutine, the computer goes to line 100 and executes the commands from line 100 to 160. The RETURN statement in line 170 sends the computer to the line following the GOSUB (line 30) that sent the computer to the subroutine. The STOP statement in line 60 sends the computer to the END statement, thus signaling the completion of the program.

A program may contain more than one subroutine. Suppose that we wish to print headings for each of the columns printed by the above program. We can do this by adding one statement to the main program and including one more subroutine.

```
5 GOSUB 200                    new main program statement


200 PRINT "N","S","M"          new subroutine
210 PRINT
220 RETURN
```

Now LIST the program and then RUN it.

```
5 GOSUB 200
10 READ N
20 GOSUB 100
40 PRINT N,S,M
50 GO TO 10
60 STOP
100 FOR K=1 TO N
110 READ X(K)
120 NEXT K
130 LET S=0
140 FOR K=1 TO N
150 LET S=S+X(K)
160 NEXT K
170 RETURN
200 PRINT "N","S","M"
210 PRINT
220 RETURN
300 DATA 3,73,67,85
310 DATA 3,82,88,97
320 DATA 4,73,67,85,83
330 DATA 5,66,78,71,82,75
999 END

RUN
N              S              M

 3              225            0
 3              267            0
 4              308            0
 5              372            0

DATA ERROR AT LINE 10
```

A program may have many GOSUB's, each of which go to the same subroutine; this is especially helpful when the same (or similar) statements appear more than once in the same program.

When you use subroutines, keep them distinct from the main program. This is normally done by placing them after the main program. Remember that the last statement in the main program must be a STOP or a GO TO (e.g., GO TO 999—the END statement).

SNOOPY AND THE RED BARON

It is 1976 and the candidates for President of the United States are Snoopy and the Red Baron. We conducted a poll by asking 29 of our friends to respond to the following question.

> Who should be President in 1976? Circle the number to the left of the candidate of your choice.
>
> 1. Snoopy
> 2. Red Baron

You guessed it . . . we are going to write a program to direct the computer to *count* the votes for each candidate, Snoopy and the Red Baron. First, let's record the 29 votes in one or more DATA statements. Remember, each vote is a "1" or a "2."

```
91 DATA 2,1,2,1,1,2,2,2,1,1,2,2,2,1,1
92 DATA 2,2,1,1,2,2,2,1,2,1,1,1,2,2
```

Before continuing, we ask you the following questions.

How many votes did Snoopy get?_____

How many votes did Red Baron get?_____

Add the number of votes obtained by Snoopy and the number obtained by the Red Baron. The total should be 29 . . . if not, check your work.

We think you answered the questions on the preceding page as follows.

1. You counted the number of "1s" in the two DATA statements. The result is the number of votes obtained by Snoopy.

2. Then you counted the number of "2s" in the two DATA statements. The result is the number of votes obtained by the Red Baron.

That's fine, but we are going to use a different method. First, we define two subscripted variables and assign the value zero (0) to each. Like this:

Then we read the 29 votes, one by one. If the vote (V) is a "1" we increase the value of $C(1)$ by one. But if the vote is a "2," we increase the value of $C(2)$ by one. After we have read all 29 votes, $C(1)$ will contain the number of votes for Snoopy and $C(2)$ will contain the number of votes for the Red Baron. Here is the program and the results of a RUN.

If you don't understand the program, study the trace on the next page.

```
10 LET C(1)=0
20 LET C(2)=0
30 FOR K=1 TO 29
40 READ V
50 LET C(V)=C(V)+1
60 NEXT K
70 PRINT C(1),C(2)
91 DATA 2,1,2,1,1,2,2,2,1,1,2,2,2,1,1
92 DATA 2,2,1,1,2,2,2,1,2,1,1,1,2,2
99 END
RUN
 13             16
```

13 — Vote count for Snoopy.

16 — Vote count for the Red Baron.

Exercise 71. Study the following partial trace of our vote counting program and fill in the missing information under the *K*, *V*, *C*(1) and *C*(2) columns.

STATEMENT	*K*	*V*	*C*(1)	*C*(2)
10 LET C(1) = 0			0	
20 LET C(2) = 0			0	0
30 FOR K = 1 TO 29	1		0	0
40 READ V	1	2	0	0
50 LET C(V) = C(V) + 1	1	2	0	1
60 NEXT K	2	2	0	1
40 READ V	2	1	1	1
50 LET C(V) = C(V) + 1	2	1	1	1
60 NEXT K	3	1	1	1
40 READ V	3	2	1	1
50 LET C(V) = C(V) + 1	3	2	1	2
60 NEXT K	4	2	1	2
40 READ V				
50 LET C(V) = C(V) + 1				
60 NEXT K				
40 READ V				
50 LET C(V) = C(V) + 1				
60 NEXT K				
40 READ V				
50 LET C(V) = C(V) + 1				
60 NEXT K				
40 READ V				
50 LET C(V) = C(V) + 1				
60 NEXT K				
and so on.				

Exercise 72. We conducted another poll, asking the same question. But this time, we asked 32 people to vote for Snoopy or the Red Baron. Here are the votes.

1, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1

2, 2, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1, 1, 2

Use our program to process the above data. You will have to change Lines 30, 91 and 92.

Exercise 73. Conduct your own poll. Then use our program to process the data. You will probably have to change Lines 30, 91 and 92.

Exercise 74. Figure out how to use the following program to count votes. Change it to count the votes of either Exercise 68 or 69.

```
10 READ N <------------------------------ N votes
20 LET C(1)=0                                  |
25 LET C(2)=0                                  |
30 FOR K=1 TO N <------------------------------|
40 READ V
50 LET C(V)=C(V)+1
60 NEXT K
70 PRINT C(1),C(2)
90 DATA 29 <------------------------------ Value of N.
91 DATA 2,1,2,1,1,2,2,2,1,1,2,2,2,1,1
92 DATA 2,2,1,1,2,2,2,1,2,1,1,1,2,2
99 END
RUN
 13             16
```

No Opinion

Here is a slightly different questionnaire.

> Who should be President in 1976? Circle the number to the left of the candidate of your choice.
>
> 1. Snoopy
> 2. Red Baron
> 3. No Opinion

We asked 37 people and got the following votes.

1, 3, 1, 2, 3, 1, 1, 2, 3, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2

2, 1, 3, 2, 2, 1, 2, 1, 1, 3, 1, 1, 2, 3, 1, 1, 2, 3

How many votes for Snoopy? _____

How many votes for the Red Baron? _____

How many No Opinions? _____

TOTAL of above categories? _____

Now we have three categories, so we use three subscripted variables.

Exercise 75. Change our program of Exercise 74 into a program to process data such as the above (three categories instead of two). You will have to add a line to set $C(3)$ to zero, change Line 70 and, of course, change Lines 90, 91 and 92.

Exercise 76. In a newspaper, the results of a poll might be presented in a table such as the following.

CANDIDATE	VOTES	PERCENT
Snoopy	17	46%
Red Baron	13	35%
No Opinion	7	19%
	(% of total vote)	

Modify your program of Exercise 75 so that the results are printed in a table like the one shown above.

Exercise 77. But suppose there are *three* candidates. For example, our questionnaire might look like this.

> Who should be President in 1976? Circle the number to the left of the candidate of your choice.
>
> 1. Snoopy
> 2. Red Baron
> 3. Lucy
> 4. No Opinion

Write a program to count votes obtained by using the above questionnaire. Invent your own data . . . print results in a table similar to the table in Exercise 76.

More Choices

We have used questionnaires with two choices, three choices and four choices. In each case, the choices have been political candidates or a "No Opinion" category. Our categories don't have to be political, however. Here are several questionnaires that you can use to gather data. Do so . . . then use the program on the following page to compute the results of each of your polls.

What is your favorite musical group? Circle the number to the left of your choice.

1. Beatles
2. Blood, Sweat and Tears
3. Crosby, Stills, Nash & Young
4. Jefferson Airplane
5. Lawrence Welk
6. Rolling Stones
7. None of the Above

Which breakfast cereal do you prefer? Circle the number to the left of your choice.

1. Trix
2. Total
3. Cheerios
4. No Opinion

What is your favorite individual sport? Circle the number to the left of your choice.

1. Badminton	2. Bowling	3. Boxing
4. Fencing	5. Golf	6. Gymnastics
7. Handball	8. Ice Skating	9. Karate
10. Roller Skating	11. Snow Skiing	12. Squash
13. Surfing	14. Swimming	15. Tennis
16. Track	17. Water Skiing	18. Wrestling
19. None of Those Listed		

If someone circles two or more choices, throw out their questionnaire . . . our program can't handle this case.

Here is our program. The lines that begin with the word "REMARK" are remarks! They have no effect on the execution of the program. You may insert a REMARK line wherever you please.

```
100 REMARK***THE PEOPLE'S POLL
200 REMARK***M IS THE NUMBER OF CATEGORIES (M<=99)
210 DIM C(99)
220 READ M
300 REMARK***SET C(1),C(2),...,C(M) ALL TO ZERO
310 FOR K=1 TO M
320 LET C(K)=0
330 NEXT K
400 REMARK***PRINT A HEADING
410 PRINT "CATEGORY", "VOTES"
420 PRINT
500 REMARK***THERE ARE N VOTES
510 READ N
600 REMARK***READ N VOTES AND COUNT BY CATEGORY
610 FOR K=1 TO N
620 READ V
630 LET C(V)=C(V)+1
640 NEXT K
700 REMARK***PRINT RESULTS (M LINES,2 NUMBERS PER LINE)
710 FOR K=1 TO M
720 PRINT K,C(K)
730 NEXT K
900 REMARK***BEGIN DATA LISTS
901 DATA
902 DATA
903 DATA
999 END
```

Let's talk about this below.

Put your value of *M* here.

Put your value of *N* here.

Record your votes, beginning here. Use as many DATA statements as necessary.

*** O.K., now RUN it! ***

The statement

```
210 DIM C(99)
```

is a DIMension statement.[7] It tells the computer to permit subscripts of *C* up to and including 99. Without a DIM statement, the largest permissible subscript is 10. We chose 99 arbitrarily. We could have chosen 25 or 100 or 150 or any limit we wanted.

[7] Not necessary on EduSystems 20 or 25.

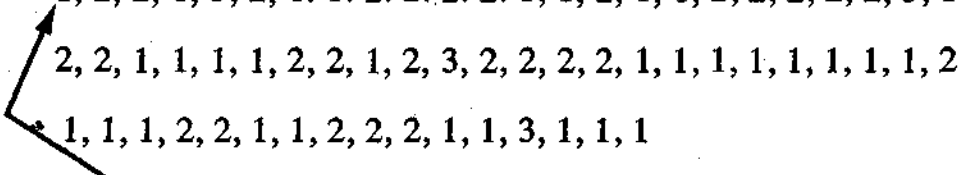
Generation Gap

Snoopy and the Red Baron . . . a new questionnaire.

> Q1. Who should be President in 1976? Circle the number to the left of your choice.
>
> 1. Snoopy
> 2. Red Baron
> 3. No Opinion
>
> Q2. Circle the number to the left of the phrase that describes your age group.
>
> 1. Under 30 years old
> 2. 30 years old or older.

The first question has three possible responses (1, 2 or 3) and the second question has two possible responses (1 or 2). Each completed questionnaire gives us two numbers. For example,

We conducted a survey. We asked 32 people to vote, therefore we got back 32 *pairs* of numbers . . . 64 numbers in all.



Begin here and circle or underline each *pair* of numbers. The first number in each pair is the vote (1, 2, or 3) and the second number in each pair is the age group of the voter.

How many people under 30 voted for Snoopy? _____

How many people 30 or older voted for Snoopy? _____

Fortunately, BASIC permits us to use variables with *two* subscripts.

		UNDER 30		30 OR OVER
Snoopy	C(1, 1)		C(1, 2)	
Red Baron	C(2, 1)		C(2, 2)	
No Opinion	C(3, 1)		C(3, 2)	

Here is a variable with *two* subscripts: C(3, 1)

First subscript

Second subscript

Have you guessed the following?

C(1, 1) = number of votes for Snoopy by people under 30.

C(1, 2) = number of votes for Snoopy by people 30 or over.

C(2, 1) = number of votes for Red Baron by people under 30.

Your turn. Complete the following.

C(2, 2) = _____

C(3, 1) = _____

C(3, 2) = _____

Here is our data again, copied from the preceding page. Use it to fill in the boxes (*C*(1,1), *C*(1,2), etc.) at the top of the page. We suggest that you look at each *pair* of numbers and, for each pair, put a mark in the appropriate box. After you have used all pairs, simply count the number of marks in each box and write in the number.

1, 2, 2, 1, 3, 2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 3, 1

2, 2, 1, 1, 1, 1, 2, 2, 1, 2, 3, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2

1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 3, 1, 1, 1

Here is a BASIC program to count the votes. We have included several REMARKS to describe what's happening.

```
100 REMARK***SNOOPY, RED BARON AND THE GENERATION GAP
200 REMARK***PUT ZEROS IN THE BOXES
210 FOR I=1 TO 3
220 FOR J=1 TO 2
230 LET C(I,J)=0
240 NEXT J
250 NEXT I
300 REMARK***READ A NUMBER PAIR AND UPDATE A BOX
310 REMARK***V=VOTE AND A=AGE GROUP
320 FOR K=1 TO 32
330 READ V,A
340 LET C(V,A)=C(V,A)+1
350 NEXT K
400 REMARK***PRINT THE RESULTS
410 PRINT "CANDIDATE","UNDER 30","30 OR OVER"
420 PRINT
430 PRINT "SNOOPY",C(1,1),C(1,2)
440 PRINT "RED BARON",C(2,1),C(2,2)
450 PRINT "NO OPINION",C(3,1),C(3,2)
900 REMARK***HERE IS THE DATA
901 DATA 1,2,2,1,3,2,1,1,2,2,2,2,1,1,2,1,1,2,2,2,2,2,3,1
902 DATA 2,2,1,1,1,1,2,2,1,2,3,2,2,2,2,1,1,1,1,1,1,1,1,2
903 DATA 1,1,1,2,2,1,1,2,2,2,1,1,3,1,1,1
999 END
```

```
RUN
CANDIDATE      UNDER 30       30 OR OVER

SNOOPY          10             6
RED BARON       4              8
NO OPINION      2              2
```

Study the above program and understand it before you move on. Perhaps the following comments will help.

If $I = 1$ and $J = 1$ then C(I, J) is C(1, 1)
If $I = 1$ and $J = 2$ then C(I, J) is C(1, 2)
If $I = 2$ and $J = 1$ then C(I, J) is C(2, 1)

and so on.

Remember . . .

C(V, A)

$V = 1, 2, \text{ or } 3$ $A = 1 \text{ or } 2$

If $V = 1$ and $A = 2$ then C(V, A) is C(1,2).

Exercise 78. Modify our program on the preceding page so that the results are printed as follows.

```
CANDIDATE     UNDER 30     30 OR OVER     TOTAL VOTES

SNOOPY        10           6              16
RED BARON     4            8              12
NO OPINION    2            2              4
```

Exercise 79. Modify your program of Exercise 78 so that instead of printing the number of votes, the computer prints the *percent* of the total number of votes, rounded to the nearest whole number percent. For example, using the same data as in Exercise 78, the computer should print the following results.

```
CANDIDATE     UNDER 30     30 OR OVER     TOTAL

SNOOPY        31 %         19 %           50 %
RED BARON     13 %         25 %           38 %
NO OPINION    6  %         6  %           12 %
```

Exercise 80. Change the questionnaire. Add a candidate and add an age group as follows.

CANDIDATES	AGE GROUPS
1. Snoopy	1. Under 21
2. Red Baron	2. 21-29
3. Lucy	3. 30 or older
4. No Opinion	

Write a program to count votes and print the results under the following headings.

```
CANDIDATE     UNDER 21     21-29     30 OR OVER
```

Reprise

A BASIC variable may have

no subscript:	A, B, C, D, . . .
one subscript:	A(3), B(K), C(X + 2), . . .
two subscripts:	A(1,2), B(I,J), C(X,X + 1), . . .

A subscript may be

a *numeral* or a *variable* or an *expression.*

A subscript must have a non-negative value. Furthermore, if the subscript is not a whole number, then the computer uses the whole number of the subscript. For example:

X(3.7) is interpreted as X(3).

If K = 2.9 then P(K) is P(2).

The computer permits a subscript value of zero (0).

```
10 LET A(0)=100
20 PRINT A(0)
99 END
```

If a subscript value exceeds 10, you may have to use a DIM statement. Otherwise, this may happen

No DIM statement here.

```
10 LET A(10)=110
20 PRINT A(10)
30 LET A(11)=111
40 PRINT A(11)
99 END
RUN
 110

SUBSCRIPT ERROR AT LINE 30
```

In Line 30, we used a subscript of 11.

So let's add a DIM statement

```
5 DIM A(11)
10 LET A(10)=110
20 PRINT A(10)
30 LET A(11)=111
40 PRINT A(11)
99 END
RUN
 110
 111
```

This DIM statement tells the computer that the subscript of *A* may be *at most* 11.

Now the program works as desired.

A DIM statement has the following general form

line number DIM list of subscripted variables

For example:

```
10 DIM A(20),B(30),C(20,12)D(5,7)
```

line number

DIM

list of subscripted variables

The above DIM statement tells the computer that:

The subscript of *A* must me $\leqslant 20$.

The subscript of *B* must be $\leqslant 30$.

The *first* subscript of *C* must be $\leqslant 20$.

The *second* subscript of *C* must be $\leqslant 12$.

The *first* subscript of *D* must be $\leqslant 5$.

The *second* subscript of *D* must be $\leqslant 7$.

If you don't mention a subscripted variable in a DIM statement, then the computer assumes that its subscript or subscripts must be less than or equal to 10.

Below are two programs. RUN each program using the given data. Describe to yourself what each program does, then make up your own data and RUN it again.

```
10 DIM K(50),A(50)
15 READ N
20 FOR J=1 TO N
25 READ K(J)
30 NEXT J
35 LET S=0
40 FOR J=1 TO N
45 READ A(J)
50 IF A(J)<>K(J) THEN 60
55 LET S=S+1
60 NEXT J
65 PRINT S
70 GO TO 35
80 DATA 12
81 DATA 2,1,2,3,4,1,3,2,1,4,4,1
82 DATA 2,1,1,3,4,1,3,1,1,3,4,2
83 DATA 2,1,2,3,3,1,3,2,1,4,4,1
84 DATA 1,1,1,3,4,1,3,2,2,4,4,1
85 DATA 3,4,2,1,4,1,2,4,3,4,1,3
99 END
```

Maybe you can figure out what the program does without a RUN. Try it!

```
100 DIM X(30)
200 READ N
210 FOR K=1 TO N
220 READ X(K)
230 NEXT K
300 FOR J=1 TO N-1
310 FOR K=J TO N
320 IF X(J)=X(K)LIST
100 DIM X(30)
200 READ N
210 FOR K=1 TO N
220 READ X(K)
230 NEXT K
300 FOR J=1 TO N-1
310 FOR K=J TO N
320 IF X(J)<=X(K) THEN 360
330 LET T=X(J)
340 LET X(J)=X(K)
350 LET X(K)=T
360 NEXT K
370 NEXT J
400 FOR K=1 TO N
410 PRINT X(K);
420 NEXT K
900 DATA 20
901 DATA 66,75,59,93,77,85,48,92,67,78
902 DATA 83,47,96,70,66,73,59,75,80,53
999 END
```

```
100 REMARK***DECIMAL MULTIPLICATION TABLE
200 REMARK***GENERATE AND STORE THE TABLE
210 FOR A=1 TO 9
220 FOR B=1 TO 9
230 LET C(A,B)=A*B
240 NEXT B
250 NEXT A
300 REMARK***PRINT THE TABLE
310 FOR A=1 TO 9
320 FOR B=1 TO 9
330 PRINT C(A,B);
340 NEXT B
343 PRINT
347 PRINT
350 NEXT A
999 END

READY

RUN

 1   2   3   4   5   6   7   8   9

 2   4   6   8   10   12   14   16   18

 3   6   9   12   15   18   21   24   27

 4   8   12   16   20   24   28   32   36

 5   10   15   20   25   30   35   40   45

 6   12   18   24   30   36   42   48   54

 7   14   21   28   35   42   49   56   63

 8   16   24   32   40   48   56   64   72

 9   18   27   36   45   54   63   72   81
```

We avoided $A = 0$ and $B = 0$. You may wish to change the program to include these values.

Exercise 81. Modify the program so that the computer prints the decimal *addition* table.

Exercise 82. Modify the program so that the computer prints the *octal* (base 8) multiplication table.[8] (Remember—the base 8 digits are: 0, 1, 2, 3, 4, 5, 6 and 7.)

[8] If you don't understand octal, see *Introduction to Programming 1972.*

KALEIDOSCOPE

Coin Tosser

The following program causes the computer to simulate (imitate) coin tossing. During the simulated tossing, no printout occurs. Instead, the computer counts the number of heads and tails that happen. After the required number of tosses have been completed, the computer prints the counts.

```
10 PRINT "HOW MANY TOSSES";
15 INPUT T
20 LET N(1)=0
25 LET N(2)=0
30 FOR K=1 TO T
40 LET X=INT(2*RND(0))+1
50 LET N(X)=N(X)+1
60 NEXT K
70 PRINT
80 PRINT "RESULTS:";N(1);"HEADS AND";N(2);"TAILS"
85 PRINT
90 GO TO 10
99 END
```

T is the number of tosses.

N(1) = number of HEADS.
N(2) = number of TAILS.

1 for HEADS and 2 for TAILS.

```
RUN
HOW MANY TOSSES? 100

RESULTS: 50 HEADS AND 50 TAILS

HOW MANY TOSSES? 100

RESULTS: 51 HEADS AND 49 TAILS

HOW MANY TOSSES? 1000

RESULTS: 500 HEADS AND 500 TAILS

HOW MANY TOSSES? 1000

RESULTS: 499 HEADS AND 501 TAILS

HOW MANY TOSSES? 1000

RESULTS: 501 HEADS AND 499 TAILS

HOW MANY TOSSES?
```

Your turn. Carry on.

Dice

Next . . . a program to simulate rolling a pair of dice. For each roll, the computer prints the number of spots showing for each die and the total spots—both dice.

```
10 PRINT "HOW MANY ROLLS";
20 INPUT T
30 FOR R=1 TO T
50 LET B=INT(6*RND(0))+1
60 PRINT A, B, A+B
70 NEXT R
80 PRINT
90 GO TO 10
99 END
```

Let's have T rolls of the pair of dice.

A = number of spots, die (1).

B = number of spots, die (2).

Total spots, both dice.

Exercise 83. Modify the above program so that the computer does *not* print the results of each roll. Instead, after all rolls have been completed, have it print a *frequency distribution* showing the number of times the total number of spots (both dice) came up 2, 3, 4, 5, . . . , 12. Here is a printout of *our* program.

```
HOW MANY ROLLS? 1000

TOTAL SPOTS      NUMBER OF TIMES

2                25
3                55
4                82
5                105
6                135
7                180
8                139
9                96
10               86
11               65
12               32
```

23 Matches

Have you ever played 23 matches? It goes like this . . . we start with 23 matches. You move first. You may take 1, 2 or 3 matches. Then I move . . . I may take 1, 2 or 3 matches. You move, I move, and so on. The one who is forced to take the last match loses. Here is a program to enable you to play 23 matches against the computer. Enter it and type RUN.

```
100 REMARK***23 MATCHES
110 LET M=23
115 PRINT
120 PRINT "WE START WITH 23 MATCHES. WHEN IT IS YOUR"
130 PRINT "TURN, YOU MAY TAKE 1,2, OR 3 MATCHES. THE"
140 PRINT "ONE WHO MUST TAKE THE LAST MATCH LOSES."
150 PRINT
200 REMARK***THE HUMAN MOVES
205 PRINT
210 PRINT "THERE ARE NOW";M;"MATCHES"
215 PRINT
220 PRINT "HOW MANY MATCHES DO YOU TAKE?";
230 INPUT H
240 IF H>M THEN 260
250 IF H=INT(H) THEN 252
251 GOTO 260
252 IF H>0 THEN 254
253 GOTO 260
254 IF H<4 THEN 280
260 PRINT "YOU CHEATED! I'LL GIVE YOU ANOTHER CHANCE."
270 GOTO 215
280 LET M=M-H
290 IF M=0 THEN 410
300 REMARK***THE COMPUTER MOVES
310 LET R=M-4*INT(M/4)
320 IF R<>1 THEN 350
330 LET C=INT(3*RND(0))+1
340 GOTO 360
350 LET C=(R+3)-4*INT((R+3)/4)
360 LET M=M-C
370 IF M=0 THEN 440
375 PRINT
380 PRINT "I TOOK";C;"MATCHES"
390 GOTO 200
400 REMARK***SOMEBODY WON(SEE LINES 290 AND 370)
410 PRINT
420 PRINT "I WON!!! BETTER LUCK NEXT TIME."
430 GOTO 100
440 PRINT
450 PRINT "O.K. SO YOU WON. LET'S PLAY AGAIN."
460 GOTO 100
999 END
```

If you move first, you can always win. But if you make a mistake, the computer will win!

We show a RUN on the following page (we lost).

```
WE START WITH 23 MATCHES. WHEN IT IS YOUR
TURN, YOU MAY TAKE 1,2, OR 3 MATCHES. THE
ONE WHO MUST TAKE THE LAST MATCH LOSES.

THERE ARE NOW 23 MATCHES
HOW MANY MATCHES DO YOU TAKE?? 3
I TOOK 3 MATCHES
THERE ARE NOW 17 MATCHES
HOW MANY MATCHES DO YOU TAKE?? 2
I TOOK 2 MATCHES
THERE ARE NOW 13 MATCHES
HOW MANY MATCHES DO YOU TAKE?? 3
I TOOK 1 MATCHES
THERE ARE NOW 9 MATCHES
HOW MANY MATCHES DO YOU TAKE?? 4
YOU CHEATED! I'LL GIVE YOU ANOTHER CHANCE.
HOW MANY MATCHES DO YOU TAKE?? 1
I TOOK 3 MATCHES
THERE ARE NOW 5 MATCHES
HOW MANY MATCHES DO YOU TAKE?? 2
I TOOK 2 MATCHES
THERE ARE NOW 1 MATCHES
HOW MANY MATCHES DO YOU TAKE?? 0
YOU CHEATED! I'LL GIVE YOU ANOTHER CHANCE.
HOW MANY MATCHES DO YOU TAKE?? 2
YOU CHEATED! I'LL GIVE YOU ANOTHER CHANCE.
HOW MANY MATCHES DO YOU TAKE?? 1
I WON!!! BETTER LUCK NEXT TIME.
```

Human (that's me) takes 3 matches.

Computer takes 3 matches.

Quite illegal!

We got caught!

Nice try, but again we got caught.

Can you beat the computer? Try it!

Rounding a Number

Sometimes we want to *round* a number. If you don't know what we mean by "round," study the following program and the RUN of it.

```
10 REMARK*** ROUND TO THE NEAREST INTEGER
20 PRINT " X","X ROUNDED"
30 PRINT
40 READ X
50 LET R=INT(X+.5)
60 PRINT X,R
70 GOTO 40
80 DATA 0,1,2,2.78,3.14,8.999,9.5,-3.7,-3.2
99 END
```

← $R = X$ rounded to nearest integer.

```
RUN
 X              X ROUNDED

 0               0
 1               1
 2               2
 2.78            3
 3.14            3
 8.999           9
 9.5             10
-3.7            -4
-3.2            -3
```

Exercise 84. Modify the program so that X is rounded to two decimal places instead of to the nearest integer. (To the nearest penny!) Use the following DATA statements.

```
DATA ERROR AT LINE 40

DATA 3.142,2.718,6.555,9.993,7.995
DATA -3.142,-2.718,-6.555,-7.993,-7.995
```

Miscellaneous Math

Previously we discussed two BASIC functions (INT and SQR) that are used to perform mathematical operations. BASIC also includes other mathematical functions to help you calculate logarithms, exponential equivalents, absolute values, and signs of values. The following examples illustrate these additional functions.

LOG

The LOG function returns the natural logarithm of X to the base e ($\log_e X$). Line 20 in the following program contains a LOG function used to convert several values to their logarithmic equivalents.

```
10 READ X
20 PRINT LOG(X)
30 DATA 54.59815,22026.47,12345,200,.720049E11
40 GO TO 10
99 END
RUN

4
10
9.421006
4.60517
25
DATA ERROR AT LINE 10
```

EXP

The exponential (EXP) function raises the number e to the power of x. EXP is the inverse of the LOG function. The following program prints the exponential equivalents of several values. Note that the input values below are the output values from the LOG function example.

```
10 READ X
20 PRINT EXP(X)
30 DATA 4,10,9.421006,4.60517,25
40 GO TO 10
99 END
RUN

54.59815
22026.46
12345
99.99999
7.200490E+10
DATA ERROR AT LINE 10
```

ABS

The absolute (ABS) function returns an absolute value for any input value. Absolute value is always positive. In the following program, various input values are converted to their absolute values and printed.

```
10 READ X
20 LET X=ABS(X)
30 PRINT X
40 DATA -35,.7,2,25E10,105555567,10.12345
50 GO TO 10
99 END
RUN

35.7
2
2.500000E+11
1.055556E+08
10.12345
DATA ERROR AT LINE 10
```

SGN

The sign (SGN) function returns the value 1 if x is a positive value, 0 if x is 0, and −1 if x is negative. The following program illustrates the use of the SGN function.

```
10 READ A,B,C
20 PRINT "A="A,"B="B,"C="C
30 PRINT "SGN(A)="SGN(A),"SGN(B)="SGN(B),"SGN(C)="SGN(C)
40 DATA -7.32,.44,0
50 GO TO 10
99 END
RUN

A=-7.32       B=.44       C=0
SGN(A)=-1     SGN(B)=1    SGN(C)=0
DATA ERROR AT LINE 10
```

Say Something in Trigonometry

If you haven't had trig, just skip this page. But if you do know something about trig, read on!

Let *R* be the measure of an angle, given in *radians.*

```
10 PRINT "R","SIN(R)","COS(R)","TAN(R)","ATN(R)"
20 PRINT
30 READ R
40 PRINT R, SIN(R),COS(R),TAN(R),ATN(R)
50 GOTO 30
60 DATA 0,1,2,3,3.14159,3.1416,4,5,6,7,8,100
99 END

READY

RUN
R             SIN(R)          COS(R)          TAN(R)          ATN(R)

0              0               1               0               0
1              .841471         .5403023        1.557408        .7853982
2              .9092974       -.4161468       -2.18504         1.107149
3              .14112         -.9899925       -.1425466        1.249046
3.14159        2.668363E-6    -1              -2.668363E-6     1.262627
3.1416        -7.349700E-6    -1               7.349700E-6     1.262628
4             -.7568025       -.6536436        1.157821        1.325818
5             -.9589243        .2836622       -3.380515        1.373401
6             -.2794155        .9601703       -.2910062        1.405648
7              .6569866        .7539022        .8714481        1.428899
8              .9893582       -.1455          -6.795713        1.446441
100           -.5063656        .8623185       -.5872141        1.560797

DATA ERROR AT LINE 30
```

Exercise 85. The above program requires that the value of *R* be given in *radians.* Modify the program so that the data is given in *degrees.* Include the following DATA statements.

```
DATA 0,30,45,90,120,135,150,180
DATA 210,225,240,270,300,315,330,360
DATA 1000,2000,3000,3630,3645,3660
```

Do It Yourself Functions

When you have mastered BASIC and are writing your own programs, you may wish to define functions other than those provided by BASIC. The DEF statement allows you to do this.

To show you how the DEF statement works, we defined as a function the formula for converting Fahrenheit to Centigrade. We provided the computer with Centigrade values. We included a check to ensure that all values were above −273. We used the formula:

$$F = \frac{9}{5}C + 32$$

to convert Centigrade to Fahrenheit. And we request that the converted values be printed.

Here is the program we used:

```
READY
 10 DEF FNF(T)=(9/5)*T+32
 20 GOSUB 100
 30 PRINT FNF(X)
 40 GOTO 10
 50 STOP
100 REMARK***READ AND TEST DATA
110 READ X
120 IF X>-273 THEN 140
130 PRINT "INVALID DATA"
140 RETURN
200 DATA -40,0,100,23,9,37,-274
999 END
RUN
-40
 32
 212
 73.4
 48.2
 98.6
INVALID DATA
-461.2

DATA ERROR AT LINE 110
```

The DEF statement in line 10 defines a function FNF which may then be used elsewhere in the program. Defined functions must have three letter names, starting with FN. The format of the DEF statement is as follows:

(line number) DEF FNF(T) = (9/5)*T + 32

argument formula argument

F may be any letter. The argument (T) has no significance but must be the same on both sides of the equal sign. The formula may be expressed in terms of numbers, several variables, other functions (INT, SQR, etc.), or mathematical expressions.

Once the function has been defined, it is called in the same manner as other BASIC functions, e.g. FNF(X). Only one DEF statement is permitted in an EduSystem 10 program. The larger EduSystems allow multiple DEF statements in a program.

Exercise 86. Define a function to find the area of a circle. Remember that the formula is:

$A = 3.14 \times R^2$ or as we say in BASIC A = 3.14*R↑2

Use the function to find the areas of circles with radii of 6, 8, and 12. Use the computer to check your answers.

PAST AND FUTURE BASIC

Look back . . . you have learned a lot about the language called BASIC.

- Direct Statements[9] SCR, RUN, LIST
- Statements: PRINT, END, LET, INPUT, GO TO, READ, DATA, FOR, NEXT, IF, DIM, REMARK, GOSUB, RESTORE
- Operations: +, −, *, /, ↑
- Relations: =, <, >, <=, >=, <>
- Functions: INT, SQR, TAB, RND
- Also: Numerals, numerical expressions, variables, variable expressions, traces, messages, loops, flowcharts, subscripted variables

Briefly mentioned:

- The SIN, COS, TAN, and ATN functions
- The LOG, EXP, ABS, and SGN functions

---

[9] A *direct statement* does not have a line number. It is executed immediately (when you press RETURN) instead of being stored in the computer's memory for later execution. Some people use the term "immediate statement" instead of "direct statement."

Now you can "say" simple things in BASIC, but you are not yet fluent. As with any language, if you want to become fluent, you must use the language and also learn more about it.

Look ahead . . . more to learn about BASIC . . . things that we haven't covered at all or have only mentioned briefly.

- Statements DEF, GOSUB, ON . . . GO TO . . . , RESTORE, RETURN, STOP
- Logical Operations: AND, OR, NOT
- Functions: ABS, ATN, COS, EXP, LOG, SIN, TAN

The above are included in many versions of BASIC. But there are also some hopped-up versions of BASIC that have additional features. Look for:

- Strings and String Variables
- String Functions
- Files

Where do you look? For each EduSystem there is a chapter in this handbook that describes the exact characteristics of the language for that EduSystem. The following chart summarizes the BASIC statements, edit and control commands, and functions and indicates the EduSystems on which they are available.

Table 1-1. BASIC Statements

Statement	Format	Description	EduSystems 5	10	15	20	25	30	40	50
Input/Output										
CLOSE	CLOSE	Close open ouput data file.					x			
CLOSE	CLOSE f	Close file f.								x
DATA	DATA $n_1,n_2,...n_n$	Numbers n_1 through n_n are variables for READ.	x	x	x	x	x	x	x	x
GET	GET f,1,r	Read record r, form as in line 1, from file f.								x
INPUT	INPUT $v_1,v_2,...v_n$	Get v_1, through v_n input from Teletype.	x	x	x	x	x	x	x	x
INPUT#	INPUT#, v	Get v (can be numeric and/ or string variable) from input data file.					x			
KILL	KILL F$	Delete a stored data file named by F$.					x			
LINPUT	LINPUT $v\$_1,v\$_2,...v\$_n$	Get long character string from Teletype.					x			x
LPRINT	LPRINT $e_1,e_2,...e_n$	Print values of specified text or expressions on line printer or high-speed paper tape punch.								x

Table 1-1 (Cont.). BASIC Statements

Statement	Format	Description	EduSystems 5	10	15	20	25	30	40	50
OPEN	OPEN f,n$	Open a file named n$ as file f.								x
	OPEN A$ FOR INPUT	Open an existing data file named by A$.					x			
	OPEN B$ FOR OUTPUT,x	Create or reopen an existing data file named by B$; x is number of blocks reserved for this file.					x			
OPEN-ELSE	OPEN f ELSE n	Open a file; go to line n if unavailable.								x
PRINT	PRINT $e_1,e_2,...e_n$	Print values of specified text, variables, or expressions. Format control (, or ;).	x	x	x	x	x	x	x	x
PRINT#	PRINT#,n	Write data (numeric or string) onto the output data file.					x			
PUT	PUT f,1,r	Write record r, form as in line 1, in file f.								x
READ	READ $v_1,v_2,...v_n$	Read variables v_1 through v_n from DATA list.	x	x	x	x	x	x	x	x
RESTORE	RESTORE	Reset DATA pointer to beginning value.	x	x	x	x	x	x	x	x

Table 1-1 (Cont.). BASIC Statements

Statement	Format	Description	EduSystems 5	10	15	20	25	30	40	50
RESTORE*	RESTORE*	Reset DATA pointer for numeric data only.								x
RESTORE$	RESTORE$	Reset DATA pointer for character string data only.								x
UNSAVE	UNSAVE f	Delete file from disk storage.								x
WRITE	WRITE $n_1...n_n$	Record data n_1 through n_n on mass storage file.			x			x	x	x
Transfer of Controls										
GO TO	GO TO n	Transfer control to line n.	x	x	x	x	x	x	x	x
IF-GO TO	If e1 r e2 GO TO n	If relationship r between e1 and e2 is true, transfer control to line n.	x	x	x			x	x	x
IF-THEN	IF e1 r e2 THEN n	Same as IF-GO TO.	x	x	x	x	x	x	x	x
	IF e1 r e2 THEN x	If relationship r between e1 and e2 is true, then perform executable BASIC statement.				x	x			
ON-GO TO	ON e1 GO TO $I_1,I_2,...I_n$	Computed GO TO.				x	x		x	x
Loops and Subscripts										
DIM	DIM $v(d_1),v(d_1,d_2)$	Dimensions, variables subscripted.	x	x	x	x	x	x	x	x

Table 1-1 (Cont.). BASIC Statements

Statement	Format	Description	EduSystems 5	10	15	20	25	30	40	50
FOR-TO-STEP	FOR v=e1 TO e2 STEP e3	Set up program loop. Define v values beginning at e1 to e2, incremented by e3.	x	x	x	x	x	x	x	x
NEXT	NEXT v	Terminate program loop increment value of v until v (>e2).	x	x	x	x	x	x	x	x
Subroutines										
GOSUB	GOSUB n	Enter subroutine at line n.	x	x	x	x	x	x	x	x
ON-GOSUB	ON e1 GOSUB $I_1,I_2,...I_n$	Computed GOSUB.				x	x			
RETURN	RETURN	Return from subroutine to statement following GOSUB or ON-GOSUB.	x	x	x	x	x	x	x	x
STOP	STOP	Transfer control to END statement.	x	x	x	x	x	x	x	x
Others										
CHAIN	CHAIN n$	Link to next user program.			x		x	x	x	x
CHAIN$	CHAIN$ A$	Link to public library program named in A$.					x			
CHANGE	CHANGE v_1, TO v_2	Change character string to array of character codes.								x
DEF	DEF FNA(x)=f(x) DEF FNA(x,y)=(x,y)	Define a function.	x	x	x	x	x	x	x	x

Table 1-1 (Cont.). BASIC Statements

Statement	Format	Description	EduSystems 5	10	15	20	25	30	40	50
END	END	End of program.	x	x	x	x	x	x	x	x
LET	LET v=f	Assign value of formula f to variable v. The word LET is optional.	x	x	x	x	x	x	x	x
RANDOMIZE	RANDOMIZE	Randomizes random number routine.			x	x	x	x	x	x
RECORD	RECORD $v_1,v_2,...v_n$	Define the size and composition of record.								x
REMARK	REM text	Insert a remark or comment.	x	x	x	x	x	x	x	x
SLEEP	SLEEP n	Cause program pause for n seconds.								x

Table 1-2. BASIC Edit and Control Commands

Command	Format	Description	EduSystems 5	10	15	20	25	30	40	50
BYE	BYE	Leave BASIC Monitor.								x
CATALOG	CAT	List names of user programs in storage area.			x		x	x	x	x
CAT$	CAT$	List names of public library programs.					x			
COMPILE	COMname	Compile program in core, save on disk.								x
CTRL/C	CTRL/C	Stop program execution, return to edit mode.	x	x	x	x	x	x	x	x
DELETE	DEL n	Delete line n.				x	x		x	x
	n	Delete line n.	x	x	x	x	x	x	x	x
	DEL n,m	Delete lines n through m inclusive.				x	x		x	x
EDIT	EDI n (c)	Search line n for character c. (See appropriate chapters for instructions on use.)				x	x		x	x
FILELOG	FIL	List the data files stored by this user.					x			
FILELOG$	FIL$	List the public data files.					x			
KEY	KEY	Return to keyboard mode after TAPE.		x		x	x	x	x	x

Table 1-2 (Cont.). BASIC Edit and Control Commands

Command	Format	Description	EduSystems 5	10	15	20	25	30	40	50
LIST	LIST	List entire program in core.	x	x	x	x	x	x	x	x
	LIST n	List program starting at line n.	x	x	x			x	x	
	LIST n	List line n only.				x	x		x	x
	LIST n,m	List lines n through m inclusive.				x	x		x	x
	LISTNH	List entire program, no header.			x			x	x	
	LISTNHn	List program starting at line n, no header.			x			x	x	
LLIST	LLIST	List program to line printer.								x
NEW	NEW	Clear core, request program name.			x		x	x	x	x
OLD	OLD	Clear core, bring program to core from storage area.			x		x	x	x	x
OLD$	OLD$	Clear core, request public library program name, bring program to core from storage area.					x			
REPLACE	REP	Replace old file on disk with version in core.								x
	REP name	If name not specified, old name retained.								x

Table 1-2 (Cont.). BASIC Edit and Control Commands

Command	Format	Description	EduSystems 5	10	15	20	25	30	40	50
RUN	RUN	Compile and run program in core.	x	x	x	x	x	x	x	x
	RUN NH	Same as RUN without header.			x			x	x	
SAVE	SAVE name	Store program named on storage device.			x		x	x	x	x
SCRATCH	SCR	Erase current program from core.	x	x	x	x	x	x	x	x
TAPE	TAP	Read paper tape; suppress printing on Teletype.			x	x	x	x	x	x
UNSAVE	UNSAVE name	Delete program named from storage area.			x		x	x	x	x
BATCH	BATCH	Commence batch processing.			x			x	x	
ECHO	ECHO	Switch from printout to non-printout mode or vice versa.			x			x	x	
LPT	LPT	Print output on line printer.			x			x	x	
LENGTH	LENGTH	Request number blocks to store current program.			x			x	x	
NAME	NAME	Same as NEW but does not delete existing program.			x			x	x	
PTP	PTP	Punch entire program on paper tape.				x				

Table 1-2 (Cont.). BASIC Edit and Control Commands

Command	Format	Description	EduSystems 5	10	15	20	25	30	40	50
PTR	PTR	Read paper tape.				x				
PUNCH	PUNCH	Punch entire program on paper tape.			x			x	x	
	PUNCH n	Punch program starting at line n.			x			x	x	
RENAME	REN	Change name of program in core.					x			
RESEQUENCE	RESEQUENCE	Renumber program lines.			x			x	x	
TTY	TTY	Print output on Teletype.			x			x	x	
PRIVILEGE	PRIVILEGE	Enable use of privileged commands.			x			x	x	
	(password)	Insert password, no echo.								
Privileged Control Commands										
BATCH	BATCH n	Same as BATCH, limit program runs to n.			x			x	x	
HEADER	HEADER (header)	Change header; type new header (maximum 12 characters) for next batch run.			x			x	x	
LOG	LOG	Print system log.			x			x	x	

Table 1-2 (Cont.). BASIC Edit and Control Commands

Command	Format	Description	EduSystems 5	10	15	20	25	30	40	50
MAX	MAX n	Set instruction limit n times 200 per program for next batch run.			x			x	x	
PASSWORD	PASSWORD (new password)	Change password, no echo.			x			x	x	
SAVE	SAVE	Save program in storage area.			x			x	x	
STACK	STACK	Start unattended batch operation.			x			x	x	
	STACK n	Same as STACK; limit runs/ program.			x			x	x	
UNSAVE	UNSAVE	Delete program from storage area.			x			x	x	

Table 1-3. BASIC Functions and Arithmetic Operations

Functions	Description	EduSystems 5	10	15	20	25	30	40	50
SQR(x)	Square root of x($\sqrt{x}$)	x	x	x	x	x	x	x	x
SIN(x)	Sine of x (x in radians)	x	x	x	x	x	x	x	x
COS(x)	Cosine of x (x in radians)	x	x	x	x	x	x	x	x
TAN(x)	Tangent of x (x in radians)	x	x	x	x	x	x	x	x
ATN(x)	Arctangent of x (x in radians; result in radians)	x	x	x	x	x	x	x	x
EXP(x)	e^x (e=2.712818)	x	x	x	x	x	x	x	x
LOG(x)	Natural log of x ($\log_e x$)	x	x	x	x	x	x	x	x
ABS(x)	Absolute value of x ($\|x\|$)	x	x	x	x	x	x	x	x
INT(x)	Greatest integer of x	x	x	x	x	x	x	x	x
SGN(x)	Sign of x (+1 if positive, −1 if negative, 0 if zero)	x	x	x	x	x	x	x	x
RND(x)	Random number between 0 and 1	x	x	x	x	x	x	x	x
FIX(x)	Truncates decimal portion of x				x	x		x	x
TAB(x)	Controls printing head position on Teletype			x	x	x	x	x	x
CHR$(x)	Converts character code to character. Used only with PRINT statement.			x	x	x	x	x	x

Table 1-3 (Cont.) BASIC Functions and Arithmetic Operations

Functions	Description	EduSystems 5	10	15	20	25	30	40	50
MID(A$,M,N)	Returns N characters, starting at the Mth character of A$.					x			
LEN(A$)	Returns the number of characters in A$.					x			
CAT(A$,B$)	Returns a string of A$ concatenated with B$ (maximum of 6 characters returned).					x			

Arithmetic Operations

SYMBOLS

↑ exponentiation
* multiplication
/ division
\+ addition
– subtraction

ORDER OF EXECUTION

1. Parenthetical expressions
2. Exponentiation
3. Multiplication and Division
4. Addition and Subtraction

chapter 2
edusystem 5

INTRODUCTION

EduSystem 5 is a BASIC-speaking *supercalculator—calculator* because it can be used like a calculator to obtain fast, accurate results, *super* because it is a computer that uses BASIC and does much more than calculate. EduSystem 5 has the ability to operate in two modes: immediate and programmable. Immediate mode allows the user to perform arithmetic calculations and obtain immediate results without writing programs. The problem and the solution are printed at the Teletype to provide a hard copy for future reference. In programmable mode, one uses BASIC to write programs and type them on the Teletype keyboard. Programs are stored in computer memory and can be printed via the teletype and, if desired, punched on paper tape to be used again in the future.

EduSystem 5 BASIC contains all the elements needed to write and execute meaningful programs. In addition, it provides several special features.

- Several commands may be typed on a single line. Programs using this feature require less storage in the computer, thus enabling users to write longer programs.
- A colon (:) may be used in place of the PRINT command to save time and storage space.
- Typing errors are easily corrected with the use of the ALT MODE (or ESCAPE), ← back arrow, or RUBOUT key.
- INPUT statement responses may be either mathematical expressions or numeric values.

All these features, and more, are yours with EduSystem 5. And it expands. If your needs grow beyond EduSystem 5, you can expand it, with a few simple additions, to an intermediate-scale EduSystem 15.

System Components

EduSystem 5 is composed of a table-top computer (PDP-8/F) with 4096 words of core memory and a Teletype with paper tape reader and punch. An optional off-line Teletype with paper tape reader and punch allows users to prepare paper tapes of their programs before coming to the EduSystem 5 and increases the number of persons who may use it each day. Each EduSystem 5 includes the BASIC language processor, a user's guide, and a self-teaching workbook for learning the BASIC language.

System Expansion

EduSystem 5 is easily expanded to EduSystem 15 by adding a DECtape magnetic tape drive (TD8-E), memory extension control, 256 word read-only memory, and an EduSystem 15 software kit. (See Chapter 4 for a full description of the capabilities of EduSystem 15.)

BASIC LANGUAGE CAPABILITIES

EduSystem 5 BASIC includes the language elements shown in Table 2-1. These elements are used as explained in Chapter 1. Differences in usage for EduSystem 5 are discussed below.

Line Numbers

In EduSystem 5 BASIC there is no upper limit on the size of the line number for any statement.

Single-Character PRINT Command

EduSystem 5 permits the use of a colon (:) in place of the PRINT command. This abbreviation can be used in place of PRINT in either programmable or immediate mode. The statement format is the same as that of the PRINT command, for example:

```
10  :5+10  is the same as  10  PRINT 5+10
```

Multiple Statements per Line

EduSystem 5 allows more than one command to be typed on a single line. Commands after the first begin with a back slash character (\), typed as SHIFT/L on the keyboard. A program is often more understandable when statements, such as a series of

LET's[1], are grouped into a single line. For example, the program:

```
10  A=1\B=4\C=6
20  :(A+B)*C
99  END
```

is the same as

```
10  LET A=1
20  LET B=4
30  LET C=6
40  PRINT (A+B)*C
99  END
```

and will produce the same result when the RUN command is typed:

```
RUN
30
```

This capability is helpful when the program to be written is too big for EduSystem 5. Commands take less storage in the computer when they are grouped as a single statement.

Immediate Mode

EduSystem 5 allows certain BASIC statements to be used in immediate mode, that is, to be issued and executed immediately without being included in a formal program. Commands commonly used with immediate mode are PRINT (or:), LET,[1] FOR, and NEXT. Immediate mode is a quick way to calculate expressions and equations. For example, the statement

```
:SIN(1),COS(1),TAN(1)
```

followed by the RETURN key, causes the sine, cosine, and tangent of 1 radian to be printed immediately, as follows:

```
 0.841471        0.540302        1.55741
```

Typing multiple commands per line is especially useful in the immediate mode. A table of square roots of the first 10 integers

[1]Remember that the word LET is optional in the LET statement.

could, for example, be generated by typing the following single line and pressing the RETURN key:

```
   FOR I=1 TO 10\: SQR(I),\NEXT I

1            1.41421      1.73205      2      2.23607
2.44946      2.64575      2.82843      3      3.16228
```

Although they are rarely used, other BASIC commands are available in immediate mode. The immediate GOTO command may be used to start a program at a point other than the beginning. This is accomplished by loading the program into the computer memory and typing, for example:

```
GO TO 200
```

After the RETURN key is pressed, the program execution will begin automatically at line number 200. In this case, the RUN command need not be typed.

INPUT Statement

EduSystem 5 allows the student to respond to the INPUT query (?) with either a mathematical expression or a numeric value. An expression entered as input may contain one or more arithmetic operations and may use any available BASIC function. For example, the BASIC statement:

```
10  INPUT X
```

could be answered in any of the following ways:

```
?10↑5+6        ?LOG(186)        ?SQR(4↑2-2)
```

As explained in Chapter 1, the INPUT statement may have multiple inputs. These inputs may be either mathematical expressions or numeric values. For example, the BASIC statement

```
10  INPUT A,B,C
```

could be answered as follows:

```
?512,INT(876.33),7+6↑25
```

NOTE

Remember that you use CTRL/C to stop a program that is running.

PROGRAM EDITING

There are two times when a program may require editing procedures. The first occurs while a line is being typed but *before* the RETURN key has been pressed. The second occurs when a line has been completely typed and the RETURN key has been pressed. Each situation has its own editing procedures.

Procedure 1: Before the RETURN key is pressed.

Three keys may be used to correct typing errors: ALT MODE (or ESCAPE), ← (back arrow), or RUBOUT.

ALT MODE (or ESCAPE) is used to delete an entire line. When this key is used, BASIC prints $DELETED, erases that line from the program, and returns the carriage so that line may be retyped.

← (back arrow), SHIFT/0 on the keyboard, or RUBOUT is used to delete a character from a line. BASIC prints a back arrow, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

Procedure 2: After the RETURN key is pressed.

Once a line of the program has been transmitted to computer memory via the RETURN key, several methods of correction may be used. Lines may be inserted, deleted, or changed.

INSERTION: To add a line to a program, assign a line number that falls between two existing lines, type the line number and text, and press RETURN.

DELETION: To erase a line from computer memory, type the line number *only* and press the RETURN key.

CHANGE: To change an individual line, simply retype it. The old instruction is replaced by the new one.

ERROR MESSAGES

EduSystem 5 checks all commands before executing them. If it cannot execute a command, it informs the user by printing one

of the following messages and the line number in which the error was found.

Message	Explanation
SYNTAX ERROR	Command does not have correct syntax. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors.
FUNCTION ERROR	The function used was deleted at system load time and thus is not available. A DEF statement will produce this message if the DEF capability was deleted.
TOO-BIG ERROR	Program and variables exceed computer capacity. Reducing one or the other may help. If the program has undergone extensive revision, try punching it out, typing SCR, and reloading.
SUBSCRIPT ERROR	The subscript used is outside the DIM statement limits.
LINENO ERROR	A branch statement (GOTO, GOSUB, or IF) references a nonexistent line.
FOR ERROR	FOR loops are too deeply nested.
NEXT ERROR	FOR and NEXT statements are improperly paired.
GOSUB ERROR	Subroutines are too deeply nested.
RETURN ERROR	GOSUB and RETURN statements are improperly paired.
DATA ERROR	No more items are in the data list.
ARGUMENT ERROR	A function has been given an illegal argument, e.g., SQR(−1).

To correct the error indicated by the message, the appropriate line in the program must be corrected in the manner described under Program Editing, Procedure 2.

OPERATING INSTRUCTIONS

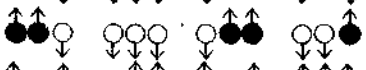
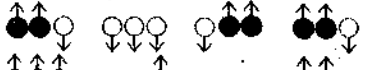
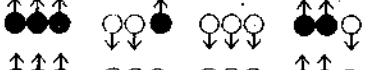
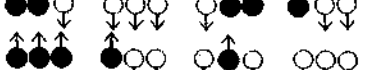
Initial Installation

When EduSystem 5 is first installed, it must be loaded with a special software program, the BASIC language processor. Once this software is loaded, it need not be reloaded. Perform the following steps to load BASIC.

1. Plug the EduSystem 5 computer into a standard 3-prong

electrical outlet. Plug the Teletype into a second standard outlet. Turn the key lock on the front of the computer to the power position and turn the Teletype to line. Set all switches on the SWITCH REGISTER (to the left of the ADDR LOAD switch) to the "down" position and press the EXTD ADDR LOAD switch.

2. Perform the following set of switch manipulations. In each step, there are 12 figures which correspond to the 12 switches labeled SWITCH REGISTER (SR) on the front of the computer. The ● symbol indicates that the corresponding switch should be set to its "up" position. The ○ symbol means that the corresponding switch should be set to its "down" position. The octal instructions which correspond to this diagram appear on the right.[2]

		OCTAL
Set SR to: 	then press ADDR LOAD	7756
Set SR to: 	then lift DEP	6032
Set SR to: 	then lift DEP	6031
Set SR to: 	then lift DEP	5357
Set SR to: 	then lift DEP	6036
Set SR to: 	then lift DEP	7106
Set SR to: 	then lift DEP	7006
Set SR to: 	then lift DEP	7510
Set SR to: 	then lift DEP	5357
Set SR to: 	then lift DEP	7006
Set SR to: 	then lift DEP	6031
Set SR to: 	then lift DEP	5367
Set SR to: 	then lift DEP	6034
Set SR to: 	then lift DEP	7420
Set SR to: 	then lift DEP	3776
Set SR to: 	then lift DEP	3376
Set SR to: 	then lift DEP and again lift DEP	5356

[2]The octal instructions are provided for those familiar with the octal, or base 8, number system. An explanation of this system is included in *Introduction to Programming 1972*.

3. Place the tape labeled EDUSYSTEM-5 in the Teletype paper tape reader. Position the tape at the single row of holes punched at the beginning of the tape.
4. Set the SR to  (7756); then press ADDR LOAD. Press the CLEAR switch, then the CONT switch. Push the paper tape reader switch to the START position. The tape should read in. If it stops before the end of tape, an error has occurred. Repeat steps 2, 3, and 4.
5. When the tape has read in properly, BASIC prints the following message:

```
SELECT THE SMALLEST SET OF FUNCTIONS NEEDED FROM THE
FOLLOWING CHOICES
ATN           ! !X! ! ! ! ! !
LOG+EXP       ! !X!X! ! ! ! !
SIN+COS+TAN   ! !X!X!X! ! ! !
DEF(FN)       ! !X!X!X!X! ! !
SQR           ! !X!X!X!X!X! !
RND           ! !X!X!X!X!X!X!

OPTION         A B C D E F G

TYPE OPTION LETTER?
```

At this time, it is possible to delete any functions which will not be used. In response to the question "TYPE OPTION LETTER?", type the letter of the option that represents the functions needed.

Deleting functions increases the size of the BASIC program which may be accommodated. If all functions are deleted (option A), the maximum program size is approximately 60 lines. If all functions are retained (option B), the maximum program size is approximately 30 lines.

6. After the functions have been selected, BASIC prints the following question:

```
DO SUBSCRIPTS START AT Ø OR 1?
```

Indicate whether subscripts will begin at 0 or 1. Many BASIC programs do not use the zero element of an array. If this is the case, setting subscripts to start at 1 allows larger programs to be run.

7. EduSystem 5 is now ready for use. Turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

Turning Off the System

Perform the following steps to turn off the EduSystem 5:

1. Type CTRL/C to stop any program that is running.
2. Turn the key lock to OFF.

Restarting the System

Perform the following steps to restart the EduSystem 5:

1. Turn the key lock to POWER.
2. Press the CLEAR switch, then the CONT switch.
3. EduSystem 5 is now ready for use. Turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

Reloading the Functions

If a need arises for functions which were deleted at system load time, the functions can be reloaded without reloading the entire system. Perform the following steps to reload the functions:

1. Type CTRL/C to stop any program that is running.
2. Turn the key lock to the POWER position; press the HALT switch, then raise it again.
3. Follow the procedure for Initial Installation, starting at step 3. Use the shorter tape labeled EDUSYSTEM-5 FUNCTIONS ONLY instead of the EDUSYSTEM-5 tape.

Saving Programs on Paper Tape

Once a program has been typed in correctly, it may be saved on paper tape so that it may be reloaded quickly. To save the program, follow this sequence of steps:

1. Turn the Teletype control knob to LINE.
2. Type LIST but do *not* press the RETURN key.
3. Turn the Teletype paper tape punch ON.
4. Turn the Teletype control knob to LOCAL.
5. Press the HERE IS key to produce some leader tape.
6. Turn the Teletype control knob to LINE.
7. Press RETURN.
8. When punching is complete, turn the control knob to LOCAL.
9. Press the HERE IS key to produce some trailer tape.

10. Turn the Teletype punch OFF.
11. Turn the Teletype control knob to LINE.

Reloading Programs from Paper Tape

Programs punched out on paper tape may be reloaded using the Teletype paper tape reader. To reload programs, follow this sequence of steps:

1. Turn the Teletype control knob to LINE.
2. Type SCR, then press the RETURN key.
3. Insert the program tape in the reader.
4. Turn the Teletype reader to START.
5. When the tape has read in, turn the Teletype reader to FREE.

Table 2-1. EduSystem 5 BASIC Statement Summary

Statement	Description
LET	Assign a value to a variable. LET is optional.
PRINT (or:)	Print out the indicated information.
READ	Assign values from data list to variables.
DATA	Provide data for a program.
GOTO	Change order of program execution.
IF GOTO IF THEN	Conditionally change order of program execution.
FOR TO STEP	Set up a program loop.
NEXT	End of program loop.
GOSUB	Go to a subroutine.
RETURN	Return from a subroutine.
INPUT	Get values of expressions from the Teletype.
REM (or ')	Insert a program comment.
RESTORE	Restore the data list.
DEF	Define a function. (Availability must be requested when system is loaded.)
STOP	Stop program execution.
END	End a program.
DIM	Define subscripted variables.
Functions[3]	
ABS(X)	Absolute value of x
ATN(X)	Arctangent of x(result in radians)
COS(X)	Cosine of x(x in radians)
EXP(X)	e^x (e=2.718282)
INT(X)	Greatest integer of x
LOG(X)	Natural logarithm of x
RND(X)	Random number
SGN(X)	Sign of x(+1 if positive, −1 if negative, 0 if zero)
SIN(X)	Sine of x(x in radians)
SQR(X)	Square root of x
TAN(X)	Tangent of x(x in radians)
Editing/Control Commands	
LIST	List all stored program statements.
LIST n	List program statements beginning at line n.
RUN	Run the currently stored program.
SCR	Delete the currently stored program.
CTRL/C	Stop execution of a program or printing of a listing. CTRL/C is typed by pressing C while holding down the CTRL key.

[3]The ABS, INT, and SGN functions are always available. Other functions (and the DEF statement) must be selected when the system is loaded. (See Initial Installation, step 5.)

chapter 3
edusystem 10

INTRODUCTION

EduSystem 10 is a mini-EduSystem with maxi-potential. It speaks a very fluent BASIC with all the standard features and a few special ones. It provides printed output and paper tape reading and punching. EduSystem 10 even tells you when you make a mistake and provides simple corrective measures. So why, with all EduSystem 10 has to offer, do we call it a starter system? Because EduSystem 10 can expand as your needs expand. It has the built-in potential to grow into larger EduSystems—EduSystem 20 or 30 at first, and as big as you want to go thereafter.

Even though it isn't as powerful as the larger EduSystems, EduSystem 10 has some features that even some of the big ones can't duplicate. Two operating modes are available: immediate and programmable. Immediate mode lets you perform arithmetic calculations without writing programs. Programmable mode lets you write programs in BASIC, store them in the computer, and punch them on paper tape. Both modes provide the problem and the solution on printed output. Other features include:

- Multiple statements per line to save computer storage space and let you write longer programs.
- An abbreviated PRINT command, colon (:).
- Special keys (ALT MODE, RUBOUT, and ←) for correcting typing errors.
- Mathematical expressions or numeric values as responses to the INPUT statement.

System Components

EduSystem 10 is composed of a table-top computer (PDP-8/E), 4096 words of core memory, automatic loader (hardware bootstrap), and a Teletype with paper tape reader and punch. Each

EduSystem 10 includes a BASIC language processor and a library of sample programs, textbooks, and curriculum guides. Optional components for EduSystem 10 include one or more Teletypes for off-line preparation of paper tape and a high-speed paper tape reader and punch which facilitates the use of other system capabilities such as FOCAL, FORTRAN, and assembly language.

System Expansion

EduSystem 10 may be easily expanded to either EduSystem 20 or EduSystem 30. To expand to EduSystem 20, add 4096 words (or more) of core memory, an EduSystem 20 software kit, and as many as 7 additional Teletypes with interfaces. To expand to EduSystem 30, add one DECdisk or DECtape, an optical mark card reader, and an EduSystem 30 software kit. (Chapters 5 and 7 fully describe the capabilities of EduSystem 20 and EduSystem 30, respectively.)

BASIC LANGUAGE CAPABILITIES

EduSystem 10 BASIC includes the language elements shown in Table 3-1. Normally, these elements are used in programs as explained in Chapter 1. BASIC usage differences with EduSystem 10 are explained below.

Line Numbers

EduSystem 10 BASIC does not place an upper limit on the size of the line number for any statement.

Single-Character PRINT Command

A colon (:) may be used in place of the PRINT command. This abbreviation may be used in either programmable or immediate mode. The statement format is the same as that of the PRINT command, for example:

```
10  :SQR(A+B) is the same as 10  PRINT SQR(A+B)
```

Multiple Statements per Line

EduSystem 10 allows more than one command to be typed on a single line. Commands after the first begin with a back slash character (\) typed as SHIFT/L on the keyboard. A program is often more understandable when statements, such as a series of

LET's[1], are grouped into a single line. For example, the program

```
100  X=2\Y=8\Z=12
110  :Y/X*Z
999  END
```

is the same as

```
100  LET X=2
110  LET Y=8
120  LET Z=12
130  PRINT Y/X*Z
999  END
```

and will produce the same result when the RUN command is typed:

```
RUN
48
```

The multiple-statement capability is helpful when the program to be written is too big for EduSystem 10. Commands require less storage in the computer when they are grouped as a single statement.

Immediate Mode

EduSystem 10 allows certain BASIC statements to be used in immediate mode, that is, to be issued and executed immediately without being included in a formal program. Commands commonly used with immediate mode are PRINT (or:), LET, FOR, and NEXT. Immediate mode is a quick way to calculate expressions and equations. For example, the statement:

```
: INT(76.87 + 2.9)
```

followed by the RETURN key, causes the value of the nearest integer to be printed immediately, as follows:

```
79
```

Typing multiple commands per line is especially useful in the immediate mode. A table of random numbers could, for example, be generated by typing the following single line and pressing the RETURN key:

[1]Remember that the word LET is optional in the LET statement.

```
FOR I=1 TO 20\PRINT RND(0),\NEXT I

 0.217873      0.696209      0.29751       0.963794      0.463246
 0.767746      0.829399      0.181667      0.159454      6.52568E-2
 0.793194      0.644913      0.927201      0.894656      0.974861
 0.804367      0.992458      0.68785       0.619773      0.731568
```

Although they are rarely used, other BASIC commands are available in immediate mode. The immediate GOTO command may be used to start a program at a point other than the beginning. This is accomplished by loading the program into the computer memory and typing, for example:

```
GO TO 35
```

After the RETURN key is pressed, the program execution will begin automatically at line number 35. In this case, the RUN command need not be typed.

INPUT Statement

The INPUT statement described in Chapter 1 allows a number to be entered from the Teletype as the value for a variable. Edu-System 10 allows the student to respond to the INPUT query (?) with a value or a mathematical expression. An expression may contain one or more arithmetic operations and may use any available BASIC function. For example, the BASIC statement:

```
10 INPUT X
```

could be answered in either of the following ways:

```
?2*8.4/3          ?SQR(100) + 20↑2
```

This capability could be used to enable one program to solve more than one problem.

As explained in Chapter 1, the INPUT statement may have multiple inputs. These inputs may be either mathematical expressions or numeric values. For example, the BASIC statement:

```
100  INPUT X,Y,Z
```

could be answered as follows:

```
?23,INT(284.978),25+86↑9
```

NOTE

When using the INPUT statement in programs, remember that CTRL/C is used to stop a program that is running.

Table 3-1. EduSystem 10 BASIC Statement Summary

Statement	Description
LET	Assign a value to a variable. LET is optional.
PRINT(or :)	Print out the indicated information.
READ	Assign values from data list to variables.
DATA	Provide data for a program.
GOTO	Change order of program execution.
IF GOTO IF THEN	Conditionally change order of program execution.
FOR TO STEP	Set up a program loop.
NEXT	End of program loop.
GOSUB	Go to a subroutine.
RETURN	Return from a subroutine.
INPUT	Get values or expressions from the Teletype.
REM(or ')	Insert a program comment.
RESTORE	Restore the data list.
DEF	Define a function. (Availability must be requested when system is loaded.)
STOP	Stop program execution.
END	End a program.
DIM	Define subscripted variables.
Functions[2]	
ABS(X)	Absolute value of x
ATN(X)	Arctangent of x (result in radians)
COS(X)	Cosine of x (x in radians)
EXP(X)	e^x (e = 2.718282)
INT(X)	Greatest integer of x

[2]The ABS, INT, and SGN functions are always available. Other functions (and the DEF statement) must be selected when the system is loaded. (See Initial Installation, step 5.)

Table 3-1. (Cont.) EduSystem 10 BASIC Statement Summary

Statement	Description
Functions (Cont.)	
LOG(X)	Natural logarithm of x
RND(X)	Random number
SGN(X)	Sign of x (+ 1 if positive, − 1 if negative, 0 if zero)
SIN(X)	Sine of x (x in radians)
SQR(X)	Square root of x
TAN(X)	Tangent of x (x in radians)
Editing/Control Commands	
LIST	List all stored program statements.
LIST n	List program statements beginning at line n.
RUN	Run the currently stored program.
SCR	Delete the currently stored program.
CTRL/C	Stop execution of a program or printing of a listing. CTRL/C is typed by pressing C while holding down the CTRL key.

PROGRAM EDITING

There are two times when a program may require editing procedures. The first occurs while a line is being typed but *before* the RETURN key has been pressed. The second occurs when a line has been completely typed and the RETURN key has been pressed. Each situation has its own editing procedures.

Procedure 1: Before the RETURN key is pressed.

Three keys may be used to correct typing errors: ALT MODE (or ESCAPE), ←(back arrow), or RUBOUT.

ALT MODE (or ESCAPE) is used to delete an entire line. When this key is used, BASIC prints $DELETED, erases that line from the program, and returns the carriage so that the line may be retyped.

←(back arrow), SHIFT/0 on the keyboard, or RUBOUT is used to delete a character from a line. BASIC prints the back arrow, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

Procedure 2: After the RETURN key is pressed.

Once a line of the program has been transmitted to computer memory via the RETURN key, several methods of correction may be used. Lines may be inserted, deleted, or changed.

INSERTION: To add a line to a program, assign a line number that falls between two existing lines, type the line number and text, and press RETURN.

DELETION: To erase a line from computer memory, type the line number *only* and press the RETURN key.

CHANGE: To change an individual line, simply retype it. The old instruction is replaced by the new one.

ERROR MESSAGES

EduSystem 10 checks all commands before executing them: if it cannot execute a command, it informs the user by printing one of the following messages and the line number in which the error was found.

Message	Explanation
SYNTAX ERROR	Command does not have correct syntax. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors.
FUNCTION ERROR	The function used was deleted at system load time and thus is not available. A DEF statement will produce this message if the DEF capability was deleted.
TOO-BIG ERROR	Program and variables exceed computer capacity. Reducing one or the other may help. If the program has undergone extensive revision, try punching it out, typing SCR, and reloading.
SUBSCRIPT ERROR	The subscript used is outside the DIM statement limits.

Message	Explanation
LINENO ERROR	A branch statement (GOTO, GOSUB, or IF) references a nonexistent line.
FOR ERROR	FOR loops are too deeply nested.
NEXT ERROR	FOR and NEXT statements are improperly paired.
GOSUB ERROR	Subroutines are too deeply nested.
RETURN ERROR	GOSUB and RETURN statements are improperly paired.
DATA ERROR	No more items are in the data list.
ARGUMENT ERROR	A function has been given an illegal argument, e.g., SQR(−1).

To correct the error indicated by the message, the appropriate line in the program must be corrected in the manner described under Program Editing, Procedure 2.

OPERATING INSTRUCTIONS

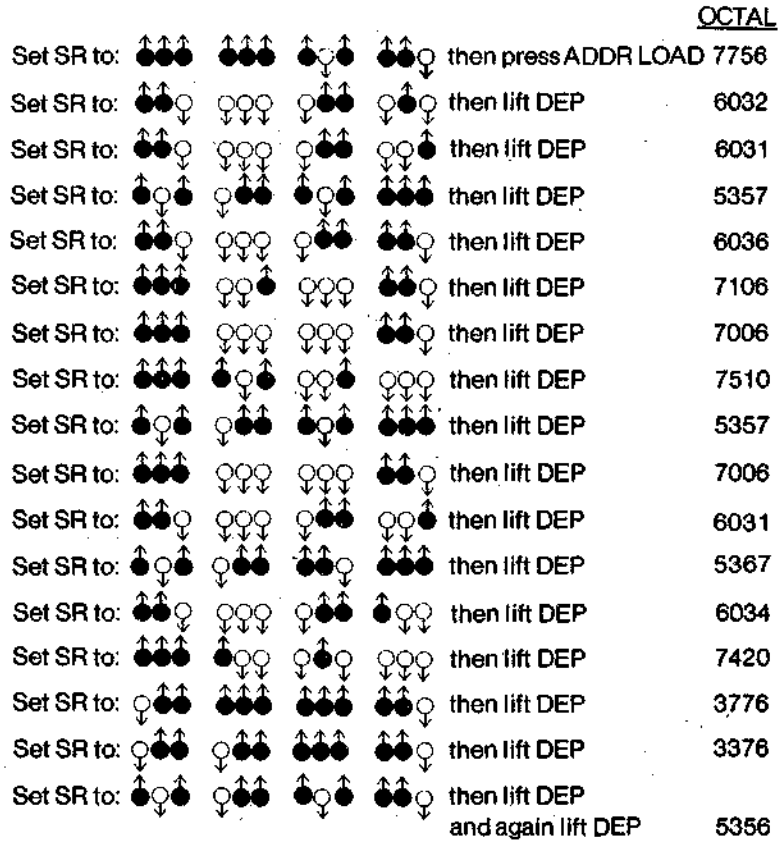
Initial Installation

When EduSystem 10 is first installed, it must be loaded with a special software program, the BASIC language processor. Once this software is loaded, it need not be reloaded. Perform the following steps to load BASIC.

1. Plug the EduSystem 10 computer into a standard 3-prong electrical outlet. Plug the Teletype into a second standard outlet. Turn the key lock on the front of the computer to POWER and the Teletype swith to LINE. Set all switches on the SWITCH REGISTER (to the left of the ADDR LOAD switch) to the "down" position and press the EXTD ADDR LOAD switch.
2. If the computer does *not* include a hardware bootstrap loader, perform the following set of switch manipulations;[3] otherwise, proceed to step 3.
   In each step, there are 12 figures which correspond to the 12 switches labeled SWITCH REGISTER (SR) on the front of the computer. The ● symbol indicates that the

[3]If the EduSystem 10 is equipped with a high-speed paper tape reader and punch, see instructions for the RIM (Read-in-mode) loader in *Appendix A*.

corresponding switch should be set to its "up" position. The ♀ symbol means that the corresponding switch should be set to its "down" position. The octal instructions which correspond to this diagram appear on the right.[4]



		OCTAL
Set SR to:	then press ADDR LOAD	7756
Set SR to:	then lift DEP	6032
Set SR to:	then lift DEP	6031
Set SR to:	then lift DEP	5357
Set SR to:	then lift DEP	6036
Set SR to:	then lift DEP	7106
Set SR to:	then lift DEP	7006
Set SR to:	then lift DEP	7510
Set SR to:	then lift DEP	5357
Set SR to:	then lift DEP	7006
Set SR to:	then lift DEP	6031
Set SR to:	then lift DEP	5367
Set SR to:	then lift DEP	6034
Set SR to:	then lift DEP	7420
Set SR to:	then lift DEP	3776
Set SR to:	then lift DEP	3376
Set SR to:	then lift DEP and again lift DEP	5356

3. Place the tape labeled EDUSYSTEM-10 in the Teletype paper tape reader. Position the tape at the single row of holes punched at the beginning of the tape.
4. If the computer does not include a hardware bootstrap loader, perform the operations in step 4b. If it has a bootstrap loader, perform the steps in 4a.
   a. Set the SR to (5356); then lower and lift the switch labeled SW. Proceed to step 6.
   b. Set the SR to (7756); then press ADDR LOAD.
5. Press the CLEAR switch, then the CONT switch. Push the paper tape reader switch to the START position. The tape

[4]Octal instructions are provided for those familiar with the octal or base 8 number system. An explanation of this system is included in *Introduction to Programming 1972.*

should read in. If it stops before the end of tape, an error has occurred. Repeat steps 2, 3, and 4.

6. When the tape has read in properly, BASIC prints the following message:

```
SELECT THE SMALLEST SET OF FUNCTIONS NEEDED FROM THE
FOLLOWING CHOICES
ATN           ! !X! ! ! ! ! !
LOG+EXP       ! !X!X! ! ! ! !
SIN+COS+TAN   ! !X!X!X! ! ! !
DEF(FN)       ! !X!X!X!X! ! !
SQR           ! !X!X!X!X!X! !
RND           ! !X!X!X!X!X!X!

OPTION         A B C D E F G

TYPE OPTION LETTER?
```

At this time, it is possible to delete any functions which will not be used. In response to the question "TYPE OPTION LETTER?", type the letter of the option that represents the functions needed.

Deleting functions increases the size of the BASIC program which may be accommodated. If all functions are deleted (option A), the maximum program size is approximately 60 lines. If all functions are retained (option B), the maximum program size is approximately 30 lines.

7. After the functions have been selected, BASIC prints the following question:

```
DO SUBSCRIPTS START AT 0 OR 1?
```

Indicate whether subscripts will begin at 0 or 1. Many BASIC programs do not use the zero element of an array. If this is the case, setting subscripts to start at 1 allows larger programs to be run.

8. EduSystem 10 is now ready for use. Turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

Turning Off the System

Perform the following steps to turn off the EduSystem 10:

1. Type CTRL/C to stop any program that is running.
2. Turn the key lock to OFF.

Restarting the System

Perform the following steps to restart the EduSystem 10:

1. Turn the key lock to POWER.
2. Press the CLEAR switch, then the CONT switch.
3. EduSystem 10 is now ready for use. Turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

Reloading the Functions

If a need arises for functions which were deleted at system load time, the functions can be reloaded without reloading the entire system. Perform the following steps to reload the functions:

1. Type CTRL/C to stop any program that is running.
2. Turn the key lock to the POWER position; press the HALT switch, then raise it again.
3. Follow the procedure for Initial Installation, starting at step 3. Use the shorter tape labeled EDUSYSTEM-10 FUNCTIONS ONLY instead of the EDUSYSTEM-10 tape.

Saving Programs on Paper Tape

Once a program has been typed in correctly, it may be saved on paper tape so that it may be reloaded quickly. To save the program, follow this sequence of steps:

1. Turn the Teletype control knob to LINE.
2. Type LIST but do *not* press the RETURN key.
3. Turn the Teletype paper tape punch ON.
4. Turn the Teletype control knob to LOCAL.
5. Press the HERE IS key to produce some leader tape.
6. Turn the Teletype control knob to LINE.
7. Press RETURN.
8. When punching is complete, turn the control knob to LOCAL.
9. Press the HERE IS key to produce some trailer tape.
10. Turn the Teletype punch OFF.
11. Turn the Teletype control knob to LINE.

Reloading Programs from Paper Tape

Programs punched out on paper tape may be reloaded using the Teletype paper tape reader. To reload programs, follow this sequence of steps:

1. Turn the Teletype control knob to LINE.
2. Type SCR, then press the RETURN key.
3. Insert the program tape in the reader.
4. Turn the Teletype reader to START.
5. When the tape has read in, turn the Teletype reader to FREE.

chapter 4
edusystem 15

INTRODUCTION

EduSystem 15 combines an extended BASIC language with on-line DECtape storage to provide a reliable, powerful system. EduSystem 15 BASIC has all the standard elements of Dartmouth BASIC plus several extended features. BASIC programs run on EduSystem 15 can be virtually limitless in size, up to 10,000 lines, due to a chaining feature that allows programs to be written in sections, then connected. A mini-string feature permits users to input, manipulate, and output alphanumeric character data, one character at a time.

The ability to store programs on the system DECtape and to retrieve them when needed eliminates the time required to read in paper tapes or type in lengthy programs. And EduSystem 15 offers protection too: a series of privileged commands that control storing programs on DECtape and deleting stored programs from DECtape. These privileged commands can be used only if the user knows the secret password. These features and more make EduSystem 15 an extremely useful classroom tool.

System Components

EduSystem 15 is composed of a table-top computer (PDP-8/F), 4096 words of core memory, a 256-word Read-Only Memory (ROM) for automatic loading, TD8-E DECtape, and a Teletype with paper tape reader and punch. Each EduSystem 15 includes the BASIC language processor and a library of sample programs, textbooks, and curriculum guides. Optional components for the EduSystem 15 include a second off-line Teletype for preparation of programs and an optional mark card reader for card processing. The system can also support high-speed paper tape reader/punch and line printer.

BASIC LANGUAGE CAPABILITIES

EduSystem 15 BASIC includes the language elements shown in Table 4-1 at the end of this chapter. Normally, these elements are used as explained in Chapter 1. EduSystem 15 also includes many advanced BASIC features to allow the user to perform more complicated and lengthy problem solving routines. BASIC usage differences and advanced features are explained in this section.

Entering Programs

EduSystem 15 BASIC expects each program to have an assigned name. At the beginning of each programming session, the NEW command should be typed to clear any existing program and define the name of the new program to be entered.[1] To use the NEW command, the user types:

```
NEW
```

and the computer asks for:

```
NEW FILE NAME--
```

The user then types any name of 1 to 6 characters, followed by the RETURN key. BASIC assigns that name to the program to be entered. The user may change the name of the program being entered at any time by typing the NAME command. BASIC again asks for NEW FILE NAME and assigns a new name to the program being entered. The NAME command does not delete the existing program.

Using Random Numbers

The RND function allows the use of random numbers within a program. Each time it is used, the RND function returns as its value a random value between 0 and 1. Unlike the other functions, the value returned by RND is not a function of its argument. However, all functions in BASIC must be followed by an argument. Therefore, RND should always be followed by a dummy argument, such as zero, which is enclosed in parentheses.

---

[1]If the user does not wish to assign a program name, he can delete any existing program by typing the SCRATCH command.

NOTE

Note that it is possible to generate random numbers over any range. For example, the expression:

```
(B-A)*RND(0)+A
```

has a random value in the range $A < n < B$.

Repeated uses of RND in a program return different values between 0 and 1. The sequence of numbers is, however, the same each time the program is run. Thus, the sequence is reproducible for later checking of the program. The RANDOMIZE statement allows the user to make the random number sequence returned by the RND function different each time a program is run. That is, when executed, the RANDOMIZE statement causes the RND function to select randomly a new sequence of random numbers. If RANDOMIZE is used, it normally appears as one of the first lines in a program.

Listing the Program

The LIST command may be used to list out all or a part of the current program. LIST prints the program statements in their proper order, regardless of the order in which they were entered. EduSystem 15's LIST command has four different forms, as shown below.

Command	Meaning
LIST	List the entire program. Precede it by a header line[2] giving the name of the program.
LIST n	List the program starting at the given line number (n). Precede it by a header line. The line number must be separated from LIST by two spaces.
LISTNH	List the entire program but do not print a header line.

[2]A header line consists of the program name followed, on the same line, by the system name (EDU BASIC). If no program name was assigned, the system prints "*NONE* EDU BASIC".

Command	Meaning
LISTNHn	List the program starting at the given line number (n) but do not print a header line.

NOTE

The programmer may stop a listing at any time by typing CTRL/C on the keyboard.

Executing the Program

The programmer may execute a program at any time by typing the RUN command. The existing program is inspected for errors; if none exist, it is executed. If an error is detected, an error message (see Error Messages) is printed. In either case, at the end of the run, BASIC prints READY, indicating that the program may now be changed or rerun. There are two types of RUN commands: RUN and RUN NH. RUN executes the current program, preceding it by a header line. RUN NH executes the current program but does not print a header line (RUN and NH must be separated by a single space).

Privileged Control Commands

Several optional commands are available which modify and control a program run. All of them are considered to be privileged instructions in the sense that the use of them is restricted. The privileged commands are available only if the privileged command capability was selected when EduSystem 15 was loaded. During normal system operation these commands are locked out; if a user attempts to use a privileged command, it is ignored and the system prints WHAT?

A special command, the PRIVILEGE command, is used to unlock and make these privileged instructions available. To use it, the user types PRIVILEGE and then the RETURN key. The system then waits for the user to type a one to six character password code. (The typed characters are not printed.) At the time the system was loaded, a password was typed by the user or assigned by the system. The characters typed in after the PRIVILEGE command are compared to this password. If they match, the PRIVILEGE command is successful and all privileged commands may then be used. If they do not match, the message INVALID PASSWORD is printed and all privileged commands continue to be unavailable.

In short, a user must know the password in order to use any privileged command. It is important that the password be kept secret. For this reason, the password is never printed when the user types it. It is also possible to change the code at any time. The instruction to change the code, PASSWORD, is, of course, a privileged instruction. The other privileged commands, SAVE and UNSAVE, are discussed below.

DECtape System Storage Capability

EduSystem 15 allows the system DECtape to be used for permanent on-line storage of programs. Programs stored in this way may be loaded instantly, without the need to load a paper tape or type in a program.

Two commands, SAVE and UNSAVE, may be used to change the contents of the DECtape storage area. Because the amount of storage space is limited, and to prevent accidental erasure of stored programs, both SAVE and UNSAVE are privileged commands. During normal system operation they are disabled. They may only be used after a successful PRIVILEGE command has been executed.

The SAVE command stores the current program in the DECtape system storage area and gives it the name specified by the last NEW, OLD, or NAME command. Any existing program stored under this name is deleted. Thus, all stored programs have names which may be used to recall them in the future. If a SAVE is attempted when the privileged commands are locked out, the system types WHAT? and ignores the command. If a successful PRIVILEGE command has been executed, but the storage area is full, the message NO SPACE is typed and the program is not stored.

The UNSAVE command is used to delete a program already stored. UNSAVE must be preceded by a NEW, OLD, or NAME command which specifies the name of the file to be deleted. The user must be certain to use exactly the same program name as he used when he first identified the program. Like SAVE, UNSAVE will be ignored unless preceded by a successful PRIVILEGED command. If the program to be deleted does not exist in the system storage area, the message NO SUCH FILE will be printed. No program will be deleted.

The CATALOG command may be used to obtain a list of the names of all programs available in the system storage area. The

CATALOG list also includes the number of storage blocks used by the program.

EduSystem 15 includes 1348 blocks of storage space. The CATALOG command may be used to determine how many of these blocks have been used and hence how many are free. If the storage space is almost full and another program is to be saved, the LENGTH command may be used to determine if there is enough room to store the current program. If enough room is not available, an existing program must be deleted first. In all cases, the maximum number of stored programs, regardless of size, is 62.

The user may load programs stored in the system storage area at any time by typing the OLD command. After the OLD command is entered, the system prints OLD PROGRAM NAME. The user then types the name of the program to be loaded. The user must be certain to use exactly the same program name as he used when he first identified the program.

ADVANCED SYSTEM CAPABILITIES

Some users will want to write programs which are too large or too complicated to be handled in the normal way. For these users, the system includes several capabilities for advanced program.

Running Very Long Programs

EduSystem 15 will run programs of up to 6000 characters or approximately 250 lines. In some instances, programs which are at or near the 6000-character limit and which contain many complex FOR, IF, and GOSUB sections will be too big to be run. In this case, the NOLINE command may be used to gain more space. If NOLINE is used, the program logic errors which are detected while the program is executing will cause an error message to be typed but the line number where they occur will not be typed. NOLINE allows substantially longer programs to be run.

If the program to be run is substantially longer than the 6000-character limit, it may still be run by means of the technique known as chaining. The program is broken down into pieces, each of which is less than 6000 characters. A chained program may have many of these pieces and, hence, may be indefinitely long. Each piece of the program is then stored in the DECtape system storage area with the SAVE command. The final command to be executed in all but the last section is a CHAIN statement which

contains the name of the next section of the program. For example, the statement:

```
950 CHAIN "PART10"
```

would cause the system to load and execute the stored program whose name is PART10.

The name of the next section of the program must be encoded in quotation marks and must be exactly six characters long. If the actual name of the next section is less than six characters, one or more spaces must be inserted before the second quotation mark to make a total of six characters. For example, if the next section of the program is named LINK2, the following CHAIN statement would be used:

```
955 CHAIN "LINK2 "
```

Execution of the CHAIN statement loads and executes the named program. The previous section of the program is deleted. Thus, the user only needs to load the first section and run it. All succeeding sections of the chained program are loaded and executed automatically.

Using a Data File

Just as some very advanced programs may be too large to be executed in the normal way, other sophisticated programs may need to store and use more data than may be accommodated under normal system operation. If this is the case, data may be temporarily stored in the system storage area. Data stored in this way is referred to as a data file.

The data file is actually a part of the program data which is defined by a program's DATA statement. All of a program's data is gathered from the DATA statements into a DATA list which is read by READ statements. As items are read from the list, they are marked as already having been used. A READ statement always fetches the *next* item from the list. In fact, the data list may be thought of having a movable marker which remembers which item of the list is next. It starts out marking the first data item. As READ statements are executed, this marker moves down the list. A RESTORE statement moves it back to the top of the list.

The data file capability allows a program, by means of a WRITE statement, to change and add to this data list as well as to read it. The WRITE statement format is the same as the DATA statement format. Writing a variable puts the value of that variable in the next place in the data list. The data item that was there previously is replaced by the new value. If a WRITE statement follows a RESTORE, it will change the first item or items in the data list. If it follows one or more READs (or WRITEs), the WRITE statement will change data items further down in the data list. The total number of items which may be put in the data list depends on the size of the program. Maximum sized BASIC programs may have up to 1000 items; small programs have room for 2000 items.

Programs which write data out to the data list must keep track of how much data has been written and the order in which it was output. If data which has been written is to be subsequently read, a RESTORE command must be executed to move the marker back to the top of the data list. If data has been written off the end of the data list, the program must remember how many items the data list contains, and be careful not to try to READ more data items than are there. The normal BASIC check for end of data does not exist for a written data list. The program must also be sure that it does not write more data than the data list can contain (1000-2000 items). Writing too much data causes part of the user's BASIC program to be destroyed.

The data file is frequently used in conjunction with chaining since data written onto the data list by one program section may be read by the next section. The program section which writes the data must execute a RESTORE just before the CHAIN statement. The next section, which will read this, must not have any DATA statement of its own since this data would destroy the data items written by the previous section.

Character Variables and String Capability

All of the standard BASIC statements deal only with numbers. All variables are assumed to be decimal values. In fact, BASIC is capable of doing many interesting operations on characters or words instead of numbers. The character handling capability of BASIC depends on the concept of representing individual characters as numbers. Each character has its own numeric code or character code, as indicated in Appendix B. When a character is

input, it is converted to a numeric code. All internal processing of that character uses this code. Since the code is a number, it may be used and manipulated with the standard BASIC statements. When the program outputs a character, it uses the character code and converts it back into a character. In short, characters stored in a BASIC program are indistinguishable from numeric values. The only difference is in the way they are used, i.e., that certain numeric values actually stand for characters.

The standard INPUT statement is used to input characters. A dollar sign ($) is placed in front of the variable name to indicate that a character code is to be input rather than a decimal number. When the character is typed, its character code is stored in the indicated variable. It is important not to confuse the inputting of characters with the inputting of numbers. The potential confusion lies in the fact that the numeric values are themselves characters. The value 192 is in fact made up of the three characters 1, 9 and 2. If these three characters were input to a BASIC program as character variables, they would be entered as three separate numeric (character code) values rather than as the single value 192. But the physical characters typed at the interactive terminal would be identical. Again, the difference is entirely in the way that the input is interpreted.

Unlike the numeric INPUT statement, character INPUT statements do *not* cause a question mark to be printed. Therefore, a series of characters may be typed without intervening question marks. Programs doing character input must therefore indicate, by PRINT statements, when input is expected. In the following examples, each program executes an INPUT statement. In the program on the left, three characters are entered and three variables are set up. In the example on the right, a single numeric value is input.

```
10 PRINT "ENTER VALUE ";
20 INPUT $X1,$X2,$X3
30 PRINT X1;X2;X3
40 END

RUN NH

ENTER VALUE 234 50  51  52

READY
```

```
10 PRINT "ENTER VALUE";
20 INPUT X
30 PRINT X
40 END

RUN NH

ENTER VALUE?234
 234

READY
```

Note that INPUT $A accepts the input character immediately *without* a carriage return. Note that the dollar sign is not a part of the variable name. It is used only in INPUT statements to indicate that typed characters are to be converted to their numeric character codes before being stored in the variable.

Character codes may be converted back to their respective characters by means of the special PRINT command function CHR$. CHR$ is the opposite of the dollar sign INPUT convention. It is a function which takes as its argument a single constant or variable and prints the single character whose character code corresponds to that value. For example, PRINT CHR$ (65) prints the character A. CHR$ may only be used in PRINT statements.

One of the most frequent uses of the character capability is to allow words or characters to be entered into BASIC programs in response to questions. For example, a program might ask the user if he wants to run the program again with a different set of input data. The user responds by typing Y if he wants to run again or by typing N if not. The program then checks the character code of the character entered to see if it equals the character code for Y. If so, it branches back to the beginning of the program. Otherwise, the program stops. The following program illustrates the use of character variables in making a run-time decision.

```
10 PRINT
20 PRINT "WOULD YOU LIKE TO DO THIS AGAIN (Y OR N)?";
30 INPUT $A
40 IF A=#Y THEN 10
50 IF A<>#N THEN 90
60 PRINT
70 PRINT "O.K. IT'S YOUR CHOICE."
80 STOP
90 PRINT
100 PRINT "Y OR N?";
110 GOTO 30
120 END
RUN NH
WOULD YOU LIKE TO DO THIS AGAIN (Y OR N)?Y
WOULD YOU LIKE TO DO THIS AGAIN (Y OR N)?8
Y OR N?Z
Y OR N?N
O.K. IT'S YOUR CHOICE.

READY
```

The comparisons shown in the preceding program are facilitated by a special BASIC language feature. Pound sign (#) followed by a single character may be used to indicate the character code of the single character following pound sign. In line 40 above, using #Y relieves the programmer of the need to remember or reference the actual character code for Y.

Often, the character capability is used to input a series, or string, of characters, such as a last name. The string may be any number of characters up to a full line. In this case, the program must read each character and see if it is the carriage return character (character code 13) which indicates the end of the line. Subscripted variables are used to store such a series of characters.

```
10 DIM A(72)
15 PRINT "TYPE YOUR NAME:";
20 FOR I=1 TO 72
30 INPUT $A(I)
40 IF A(I)=13 THEN 60
50 NEXT I
60 END
RUN NH
TYPE YOUR NAME:SUPERSTAR
READY
```

PROGRAM EDITING

There are two times when a program may require editing procedures. The first occurs while a line is being typed but before the RETURN key is pressed. The second occurs when a line has been completely typed and the RETURN key has been pressed. Each situation has its own editing procedures.

Procedure 1: Before the RETURN key is pressed.

Three keys may be used to correct typing errors: ALT MODE (or ESCAPE), ← (back arrow), or RUBOUT. ALT MODE (or ESCAPE) is used to delete an entire line. When this key is used, BASIC prints DELETED, erases that line from the program, and returns the carriage so that the line may be retyped.

← (back arrow), SHIFT/0 on the keyboard, or RUBOUT is used to delete a character from a line. BASIC prints the back arrow, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

Procedure 2: After the RETURN key is pressed.

Once a line of the program has been transmitted to computer memory via the RETURN key, several methods of correction may be used. Lines may be inserted, deleted, changed, or renumbered.

INSERTION: To add a line to a program, assign a line number that falls between two existing lines, type the line number and text, and press RETURN.

DELETION: To erase a line from computer memory, type the line number *only* and press the RETURN key.

CHANGE: To change an individual line, simply retype it. The old instruction is replaced by the new one.

RENUMBER: Occasionally, repeated editing and insertions result in there being no more room in an area of a program to insert new lines. It is then necessary to *spread out* the line numbers so there is room for new insertions. The RESEQUENCE command is used for this purpose. To renumber a program, type the RESEQUENCE command. This command changes the first line's number to 100 and renumbers each succeeding line with an increment of 10. RESEQUENCE also automatically changes all GOTO, GOSUB, and IF statements to correspond to the new line numbers.

ERROR MESSAGES

Some programs execute correctly the first time they are tried. Most others, especially if they are at all complex, have errors in them. EduSystem 15 checks all statements when they are entered and before executing them. If it cannot execute a statement it informs the user by printing one of the following types of messages.

Program Loading Errors

As each line is typed, EduSystem 15 checks it for program loading errors. If it finds an error, it prints one of the following error messages immediately after the erroneous line.

Message	Explanation
WHAT?	Line does not make sense to the system. It does not begin with a line number and is not a valid system command.

Message	Explanation
LINE NO. TOO BIG	The line number of a line or the argument of a system command is greater than 4095.
LINE TOO LONG	Line just entered is longer than 80 characters.
NO ROOM	There is no room in memory to store the line just entered.
FILE NOT SAVED	The program named as the operand of an OLD command was not previously saved on the system device.
NO SPACE	There is not enough space on the DECtape to SAVE the current program.
Bell	If an invalid character is entered, the Teletype bell rings and the character is ignored.
I/O ERROR	An input or output error occurred on the DECtape unit. Be sure that the unit is on-line, write-enabled, and the unit number is set correctly. Retry whatever was interrupted by the error. If the problem persists, there is a hardware problem.
INVALID PASSWORD	The password typed after a PRIVILEGE command is not the system password. Privileged mode is not entered.

Coding Errors[3]

After the RUN (or RUN NH) command is typed, EduSystem 15 checks each statement and command before executing it for mistakes in the BASIC program coding. If it cannot execute a

[3]To correct the coding errors indicated by the messages, the appropriate line in the program must be corrected in the manner described under Program Editing, Procedure 2.

statement or command, it informs the user by printing one of the following messages and the line number in which the error was found.

Message	Explanation
CH	There is an illegal character in the line.
EN	Program does not have END statement as the last line in the program.
FN	Not enough NEXT statements in the program. There must be a NEXT statement for each FOR statement in the program.
FO	FOR and NEXT statements do not match. There is a NEXT statement in the program whose variable is not the same as the variable in the corresponding FOR statement.
LI	Line contains an improperly written decimal number or constant. It may, for example, have two decimal points or have an alphabetic character in it.
M1 M2	The program as a whole is too large to be run by the system. Making the program smaller, reducing the size of subscripted variables, or using the NOLINE command may help.
NE	Program has too many (more than 8) FOR-NEXT loops one within another.
PC	Line contains an improperly used parenthesis. Generally, the problem is an expression which does not have an equal number of left and right parentheses.
RO	Statement contains an invalid *relational* operator (<, =,>, <=, >=). Relational operators may only be used in IF statements.
S1 S2	Statement as a whole is not properly written and, as a result, does not conform to proper BASIC syntax. For example, a semicolon is allowed in a PRINT statement but not in a READ or INPUT statement.

Message	Explanation
ST	Statement's command word is *not* one of the BASIC statement types.
TB	The program is too large to be run. Cause is usually an extremely large number of PRINT statements.
TO	Program is either too large or too complex to be run. The total number of variables, constants, functions, and line numbers should be reduced, if possible.
UL	A GOSUB, GOTO, or IF statement contains a line number which does not exist.
UQ	A quotation mark indicating the beginning of a string of text does not have a corresponding quotation mark at the end of the text string.

Program Logic Errors[4]

Some errors do not show up until the program is actually executed. An example of this type of error is an expression which uses a square root of a variable. If, when this square root is actually calculated, the variable has a negative value, a program logic error has occurred. EduSystem 15 prints the following messages if program logic errors occur.

Message	Explanation
CH	A CHAIN statement tried to chain to a program which was not available in the DECtape storage area.
CO	Program ran too long and was automatically stopped by the system.
DA	The program ran out of data. It attempted to do a READ after all data had been read.

[4]Some program logic errors may be corrected by the method described under Program Editing, Procedure 2. Most, however, necessitate the rewriting of the program.

Message	Explanation
D0	The program attempted to divide by zero. Instead of dividing by zero, BASIC divides by the smallest possible number, giving a result of about 10^{500}. This error does not cause the program to stop.
FN	An expression contains a function which was not defined in a DEF statement.
GS	The program is too complex to be executed. The problem is generally that too many subroutines have themselves executed GOSUB instructions.
LG	Program attempted to take the logarithm of a negative number or zero.
RE	A RETURN statement was used outside of a subroutine or a subroutine was entered by a GOTO instead of a GOSUB.
SP	See GS.
SQ	Program attempted to take the square root of a negative number. BASIC automatically takes the square root of the absolute value of the number instead. This error does not cause the program to stop.
SS	Program used a subscript which was too large for the variable. The maximum size of a subscript is specified in a DIM statement.
WR	There is no more room on the DECtape to write data. The program attempted to do a WRITE statement when the data list was full. (Note that if this error occurs, the program text will no longer be intact. A NEW, OLD, or SCR command must be used to clear the program area.)

OPERATING INSTRUCTIONS

Loading the System

When EduSystem 15 is first installed, it must be loaded with a special software program, the BASIC language processor. Once this

software has been loaded, it need not be reloaded. Perform the following steps to load BASIC.

INITIALIZE THE DECTAPE UNIT

Perform the following steps to prepare the TD8E DECtape unit for software loading.

1. Set the REMOTE/OFF/LOCAL switch to OFF.
2. Place a DECtape on the left spindle with the DECtape label out.
3. Wind four turns of tape onto the right spindle.
4. Set the REMOTE/OFF/LOCAL switch to LOCAL.
5. Wind a few turns of tape onto the right spindle with the → switch to ensure that the tape is properly mounted.
6. Dial 0 on the unit selector dial.
7. Set the REMOTE/OFF/LOCAL switch to REMOTE.
8. Set the WRITE ENABLE/WRITE LOCK switch to WRITE ENABLE.

INITIALIZE COMPUTER MEMORY

To initialize the PDP-8/F memory, perform the following steps.

1. Turn the key lock on the front of the computer to POWER.

2. Perform the following set of switch manipulations on the SWITCH REGISTER (to the left of the ADDR LOAD switch). In each step, there are 12 figures which correspond to the 12 switches labeled SWITCH REGISTER (SR) on the front of the computer. The ● symbol indicates that the corresponding switch should be set to its "up" position. The ○ symbol means that the corresponding switch should be set to its "down" position. The octal instructions which correspond to this diagram appear on the right.[5]

[5]Octal instructions are provided for those familiar with the octal, or base 8, number system. An explanation of this system is included in *Introduction to Programming 1972*.

		OCTAL
Set SR to: 	then press EXTD ADDR LOAD	0000
Set SR to: 	then press ADDR LOAD	7756
Set SR to: 	then lift DEP	6032
Set SR to: 	then lift DEP	6031
Set SR to: 	then lift DEP	5357
Set SR to: 	then lift DEP	6036
Set SR to: 	then lift DEP	7106
Set SR to: 	then lift DEP	7006
Set SR to: 	then lift DEP	7510
Set SR to: 	then lift DEP	5357
Set SR to: 	then lift DEP	7006
Set SR to: 	then lift DEP	6031
Set SR to: 	then lift DEP	5367
Set SR to: 	then lift DEP	6034
Set SR to: 	then lift DEP	7420
Set SR to: 	then lift DEP	3776
Set SR to: 	then lift DEP	3376
Set SR to: 	then lift DEP and again lift DEP	5356

3. Place the tape labeled EDUSYSTEM 15 in the Teletype paper tape reader. Position the tape at the single row of holes punched at the beginning of the tape. Turn the Teletype control knob to LINE.

4. Set the SR to  (7756); then press ADDR LOAD.

5. Press the CLEAR switch, then the CONT switch. Push the paper tape reader switch to START. The tape should begin to move: If it does not, repeat steps 2, 3, and 4.

System Building Dialog

When the EDUSYSTEM 15 paper tape has read in properly, BASIC prints a series of questions (see Figure 4-1). The user responds by typing Y for yes and N for no on the Teletype.

```
STANDARD SYSTEM?
```

Since EduSystem 15 has optional operating modes and may be used with optional components, if present, this question is always answered no (N).

The next four questions:

```
IS SYSTEM DEVICE A DF32 DISK?
TC01 DECTAPE?
RF08 DISK?
LINCTAPE?
```

are always answered no (N) since none of these devices are available with the EduSystem 15. A response of yes (Y) to any of the above will result in the following message, and EduSystem 15 will begin the dialog again.

```
SYSTEM DEVICE I/O ERROR
```

The system's next question is:

```
TD8E DECTAPE?
```

and the user responds Y.

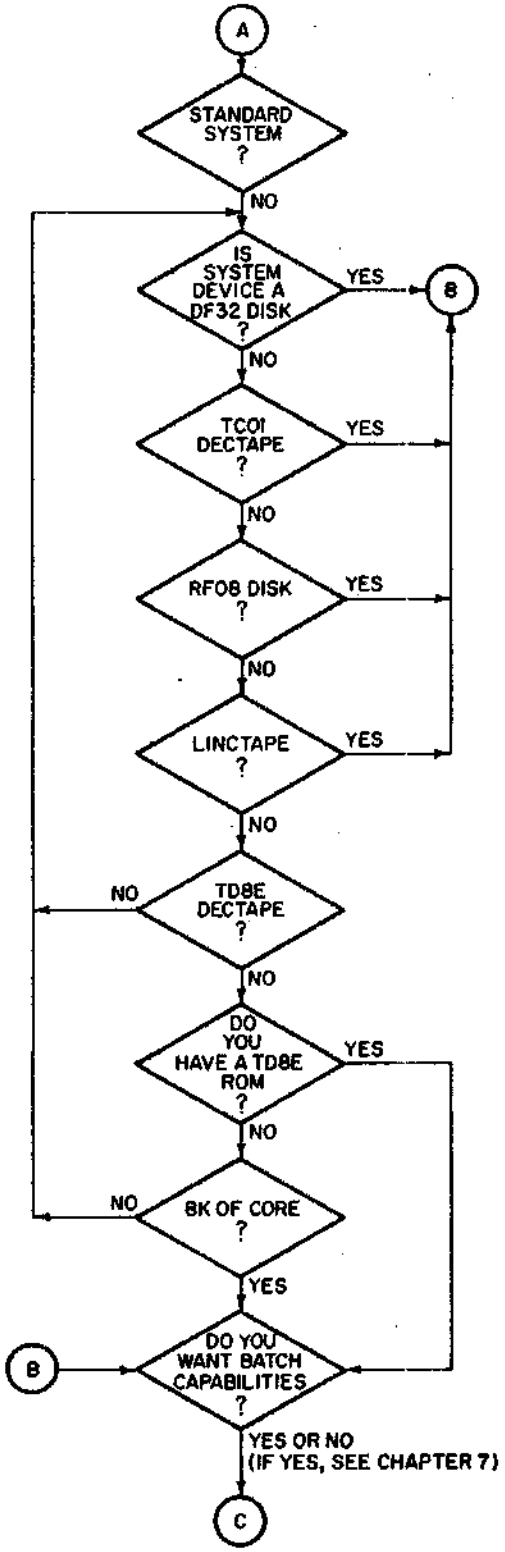


Figure 4-1. System Building Dialog

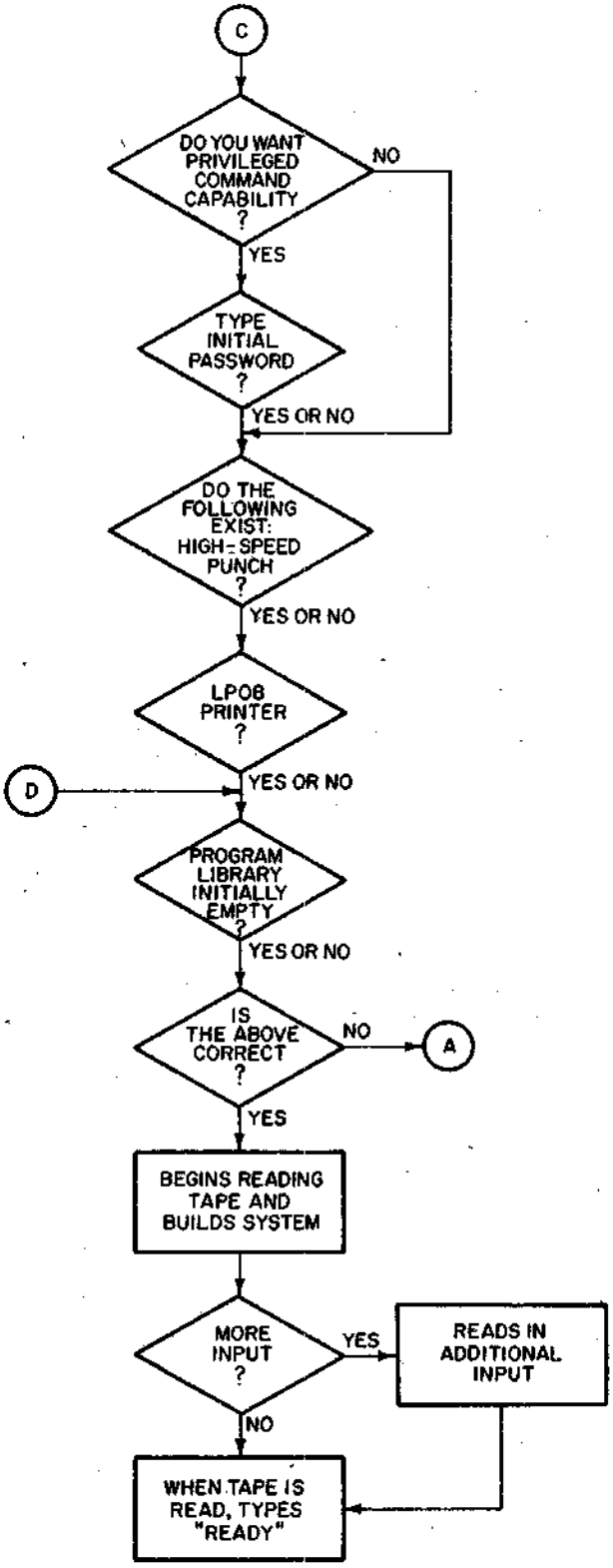


Figure 4-1. (Cont.) System Building Dialog

EduSystem 15 then asks:

```
DO YOU HAVE A TD8E ROM?
```

Answer this Y if the EduSystem 15 has a TD8E Read Only Memory. If the user answers N, the system asks:

```
8K OF CORE?
```

Since the system will have either the Read Only Memory or the 8K of core memory, answer this question Y. (The question will not be asked if the user answered Y to the previous question.)

EduSystem 15 then asks:

```
DO YOU WANT BATCH CAPABILITIES?
```

If the EduSystem 15 has an optional card reader, answer Y and refer to Chapter 7 (EduSystem 30) for additional instructions. If not processing card programs, answer N.

Next the system asks:

```
DO YOU WANT PRIVILEGED COMMAND CAPABILITY?
```

The privileged command capability prevents unauthorized users from executing critical system commands. To establish this protection, the user types Y. The system then prints:

```
TYPE INITIAL PASSWORD
```

The password is a special code which must be known to use privileged commands. Type a 1 to 6 character password, the first character of which is alphabetic. If no privileged capability is desired, respond to the original question with N; the system will not ask for a password.

The system asks if a high-speed paper tape punch and/or line printer are part of the system.

```
DO THE FOLLOWING EXIST:
    HIGH-SPEED PUNCH?
    LP08 PRINTER?
```

The response to each question must be Y if the device exists; N if it does not. Normally, an EduSystem 15 will have neither device.

The system's next question is:

```
PROGRAM LIBRARY INITIALLY EMPTY?
```

The user's response should be Y unless there are programs previously stored within the system which are to be kept. If this is the case, N must be typed as the answer and the questions in boxes 1 through 8 of the flowchart must be answered in the same way as when the system was built when the program library was empty.

When all questions have been answered, the system types:

```
IS THE ABOVE CORRECT?
```

If all questions have been answered properly, type Y. The system will load the rest of the EduSystem 15 paper tape. If any of the responses were erroneous, type N; the set of questions is repeated.

When the entire tape has been read, EduSystem 15 gives the user a chance to load additional DEC-supplied system update tapes by asking:

```
MORE INPUT?
```

If no DEC-supplied update tapes exist, respond N and EduSystem 15 is loaded. If update tapes do exist, load the first one into the tape reader and type Y to begin loading.

Finally, when all input has been read in, EduSystem 15 indicates that it is ready to process BASIC programs by printing:

```
READY
```

At this time, turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

DIAGNOSTIC MESSAGES DURING SYSTEM BUILDING

The following error messages are printed when errors are detected during the building of EduSystem 15.

```
TAPE READY?
```

This message is typed whenever the system is waiting for the paper tape reader to be loaded. It may appear by itself, usually due to a tape tear or reader jam, or it may appear as the last line of another diagnostic message.

ACTION:

1. The portion of the paper tape which is read after the system building dialog has distinct blocks of information about two and one-half tape fanfolds long. The start of such a block is indicated by nine blank tape frames followed by a frame with all positions punched.

   Back up the tape several fanfolds to the beginning of a previously read block. Position the tape such that the blanks at the beginning of the block are over the read station.
2. Type Y on the interactive terminal.

```
BAD PLACE TO START TAPE
TAPE READY?
```

This message means that after a previous message the user did not correctly position the tape to the beginning of a data block. (See discussion under TAPE READY? message.)

ACTION:

1. Correctly position the tape.
2. Type Y on the interactive terminal.

```
CHECKSUM ERROR
TAPE READY?
```

A checksum error occurred while the most recent data block was being read.

ACTION:

1. Back up the tape to the beginning of the block.
2. Type Y to reread the data.

```
SYSTEM DEVICE I/O ERROR
```

If this message occurs before the dialog has been completed, the dialog will automatically begin again. If an I/O error occurs after the dialog is completed the TAPE READY message will appear.

ACTION:

1. Make sure that the system device is on line and write-enabled and the unit number is set correctly.
2. Respond appropriately to the question which follows the message.

Turning Off the System

The system may be powered down when it will not be used for extended periods of time, such as overnight. The procedure is as follows:

1. Type CTRL/C to stop any program that is running.
2. Turn the key lock to OFF; turn the DECtape unit and Teletype to OFF.

Restarting the System

The system may be restarted at any time without reloading by the following procedure:

1. Initialize the DECtape unit.
2. Turn the key lock to POWER and the Teletype to LINE.
3. Set the SR to ●●● ●●○ ○○○ ○○○ (7600); then press ADDR LOAD.
4. Press CLEAR switch, then the CONT switch.
5. EduSystem 15 is now ready for use. Turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

Saving Programs on Paper Tape

Once a program has been typed in correctly, it may be saved on paper tape so that it may be reloaded quickly. To save the program, perform the following steps:

1. Turn the Teletype (TTY) control knob to LINE.
2. Type LISTNH but do not press the RETURN key.
3. Turn the TTY paper tape punch ON.
4. Press SHIFT/CTRL, type "PPPPPP" to produce some leader tape.
5. Press the RETURN key.

6. When punching is complete, press SHIFT/CTRL, type "PPPPPP" to produce some trailer tape.
7. Turn the TTY punch OFF.

Reloading Programs from Paper Tape

Programs punched out on paper tape may be reloaded using the Teletype (TTY) paper tape reader. The TAPE command is used to load programs from paper tape. To reload programs, perform the following steps:

1. Insert the paper tape in the TTY reader.
2. Turn the TTY control knob to LINE.
3. Type NEW, then press the RETURN key.
4. Type the program name.
5. Type TAPE, press the RETURN key.
6. Turn the TTY paper tape reader to START.
7. When the tape has read in, turn the TTY reader to FREE.

A special control command, ECHO, may be used with TAPE to prevent the program from being listed while it is being read. The first time it is used, ECHO inhibits all printout. A second ECHO command restores normal printout.

Table 4-1. EduSystem 15 BASIC Statement Summary

Statement	Description
LET	Assign a value to a variable. LET is optional.
PRINT	Print out the indicated information.
READ	Assign values from data list to variables.
DATA	Provide data for a program.
RESTORE	Restore the data list.
WRITE	Record data on DECtape storage file.
GOTO	Change order of program execution.
IF GOTO IF THEN	Conditionally change order of program execution.
FOR TO STEP	Set up a program loop.
NEXT	End a program loop.
DIM	Define subscripted variables.
GOSUB	Go to a subroutine.
RETURN	Return from a subroutine.
INPUT	Get values from the Teletype.
REMARK (REM)	Insert a program comment.
RANDOMIZE	Cause RND function to randomly select new sequence of random numbers between 0 and 1.
DEF	Define a function.

Table 4-1. (Cont.) EduSystem 15 BASIC Statement Summary

Statement	Description
CHAIN	Link to next section of a program which is stored within the system.
NOLINE	Do not print out the line number in which program logic errors are found. (Allow larger-than-normal programs to be run without chaining.)
STOP	Stop program execution.
END	End a program.
Editing and Control Commands	
LIST	List all stored program statements.
LIST n	List program starting at line n.
LISTNH	List all program statements but do not print a header line.
LISTNHn	List program starting at line n but do not print a header line.
RUN	Execute the current program.
RUN N H	Same as RUN without header line.
SCRATCH(SCR)	Delete the currently stored (in memory) program.
CTRL/C	Stop execution of a program or printing of a listing. CTRL/C is typed by pressing C while holding down the CTRL key.
TAPE	Read a program from paper tape. Ignore any line which does not begin with a line number.
ECHO	Switch from printout to non-printout mode or vice versa.
RESEQUENCE	Renumber program lines.
NEW	Clear memory, request program name.
OLD	Clear memory, bring program to memory from storage area.
NAME	Same as NEW but does not clear memory.
CATALOG (or CAT)	Print out the names of programs in storage area.
LENGTH	Print out the number of blocks needed to store the current program.
PRIVILEGE (or PRI)	Enable use of privileged commands. To be successful, this command must be followed by the correct password. This command is recognized only if the privileged command capability was selected at system load time.

Table 4.1. (Cont.) EduSystem 15 BASIC Statement Summary

Statement	Description
Privileged Commands[6]	
PASSWORD	Change the password code.
SAVE	Save the current program in the system storage area.
UNSAVE	Delete the named program from the system storage area.
Functions	
ABS(X)	Absolute value of x.
ATN(X)	Arctangent of x (result in radians).
COS(X)	Cosine of x (x in radians).
EXP(X)	e^x (e=2.718282).
INT(X)	Greatest integer of x.
LOG(X)	Natural logarithm of x.
RND(X)	Random number.
SGN(X)	Sign of x (+1 if positive, −1 if negative, 0 if zero).
SIN(X)	Sine of x (x in radians).
SQR(X)	Square root of x.
TAN(X)	Tangent of x (x in radians).
TAB(X)	Controls printing head position on Teletype.
CHR$(X)	Converts character code to character. Used only with the PRINT command.

[6]The privileged commands may only be used after a successful PRIVILEGE command has been executed.

chapter 5

edusystem 20

INTRODUCTION

EduSystem 20 is a multi-user BASIC system. The system is composed of from one to eight terminals connected on-line to a PDP-8/E computer. This means that, depending on the number of on-line terminals, EduSystem 20 allows up to eight different BASIC programs to be run at the same time. EduSystem 20 terminals need not be in the same room, or even in the same building as the computer. They may be placed in remote locations and connected to the computer by regular telephone lines.

In addition to letting several persons use BASIC at the same time, EduSystem 20 allows users to operate BASIC in two different modes: immediate and programmable. Immediate mode allows the user to perform arithmetic calculations without writing programs. Programmable mode enables programs to be written in BASIC, stored in computer memory, and, if desired, saved on paper tape. Both modes provide a printed copy of the problem and the solution.

EduSystem 20 BASIC also includes advanced features that enable a user to perform more complex programming tasks. These features include the ability to enter a subroutine if certain conditions are met, to write multiple statements per line, and to use a search character to edit a program line.

System Components

EduSystem 20 is composed of a PDP-8/E computer with 8192 words of core memory, power fail protection, automatic loader (hardware bootstrap), and up to 4 terminals and their associated interfacing. Each EduSystem 20 includes the BASIC language processor and a library of sample programs, textbooks, and curriculum guides. An additional 4096 words of core memory enable EduSystem 20 to handle up to 4 more terminals (for a total of 8). One or more off-line terminals may also be added for paper tape

preparation. The system can also support a high-speed paper tape reader/punch.

System Expansion

EduSystem 20 may be easily expanded to intermediate-sized EduSystems 25 or 40, or to the total school computer system, EduSystem 50. To expand to EduSystem 25, add 4096 words of core memory, DECtape, and an EduSystem 25 software kit. Expansion to EduSystem 40 requires a DECdisk or DECtape, an optional mark card reader, and an EduSystem 30 software kit. (Chapters 6 and 8 fully describe the capabilities of EduSystem 25 and EduSystem 40, respectfully.)

EDUSYSTEM 20 BASIC

EduSystem 20 BASIC has all the standard elements of Dartmouth BASIC plus several extended features. Tables 5-1 and 5-2 summarize the system's BASIC language capabilities. The extended features are discussed below.

Abbreviated Commands

All commands and statement keywords can be abbreviated to the first three letters, as shown in Table 5-1.

Table 5-1. EduSystem 20 BASIC Statement Summary

Command (Abbreviation)	Explanation
LET	Assign a value to a variable. LET is optional.
READ(REA)	Assign values from data list to variables.
DATA(DAT)	Provide data for a program.
PRINT(PRI)	Print out the indicated information on the Teletype.
RESTORE(RES)	Restore the data list.
GOTO(GOT)	Change order of program execution.
IF THEN(IF THE)	Conditionally perform specified operation or conditionally change order of program execution.
FOR TO STEP (FOR TO STE)	Set up a program loop.
NEXT(NEX)	End of program loop.
GOSUB(GOS)	Go to a subroutine.
RETURN(RET)	Return from a subroutine.
ON GOTO(ON GOT)	Conditionally change order of program execution according to evaluation of formula contained in statement.

Table 5-1 (Cont.). EduSystem 20 BASIC Statement Summary

Command (Abbreviation)	Explanation
ON GOSUB (ON GOS)	Conditionally go to a subroutine according to evaluation of formula contained in statement.
INPUT(INP)	Get values or expressions from the Teletype.
REMARK(REM or ')	Insert a program comment.
RANDOMIZE (RAN or RANDOM)	Cause RND function to randomly select new sequence of random numbers between 0 and 1.
DEF FN	Define a function.
STOP(STO)	Stop program execution.
END	End a program.
Editing/Control Commands	
LIST(LIS)	List entire program in memory.
LIST n(LIS n)	List line n.
LIST n, m(LIS n, m)	List lines n through m inclusive.
DELETE n(DEL n)	Delete line n.
DELETE n, m (DEL n, m)	Delete lines n through m inclusive.
EDIT n(EDI n)	Search line n for the character typed.
KEY	Return to KEY (normal) mode.
RUN	Execute the current program.
SCRATCH(SCR)	Erase the current program from memory.
TAPE(TAP)	Read a program from the Teletype paper tape reader or punch a program on the Teletype paper tape punch.
PTR	Read a program from the high-speed paper tape reader.
PTP	Punch a program out on the high-speed paper tape punch.
CTRL/C	Stop a running program, print STOP, and then READY. CTRL/C is typed by pressing C while holding down the CTRL key.
BYE	Same as SCRATCH.
NEW	Same as SCRATCH.

Table 5-2. EduSystem 20 BASIC Function Summary

Function	Description
ABS(X)	Absolute value of x
ATN(X)	Arctangent of x (result in radians)
COS(X)	Cosine of x (x in radians)
EXP(X)	e^x (e=2.712818)
INT(X)	Greatest integer of x

Table 5-2 (Cont.). EduSystem 20 BASIC Function Summary

Function	Description
LOG(X)	Natural logarithm of x
RND(X)	Random number
SGN(X)	Sign of x (+1 if positive, −1 if negative, 0 if zero)
SIN(X)	Sine of x (x in radians)
SQR(X)	Square root of x
TAN(X)	Tangent of x (x in radians)
TAB(X)	Controls printing head position on Teletype.
TAB(X)	Truncates decimal portion of x.
CHR$(X)	Converts character code to character. Used only with PRINT statement.

Multiple Statements per Line

EduSystem 20 allows more than one statement to be typed on a single line. Statements after the first begin with a back slash character (\) which is SHIFT/L on the keyboard. A program is often more understandable when statements, such as a series of LET's, are grouped into a single line. For example, the program:

```
10 X=82\Y=75\Z=98
20 PRINT (X+Y+Z)/3
```

is the same as

```
10 LET X=82
20 LET Y=75
30 LET Z=98
40 PRINT (X+Y+Z)/3
```

and will produce the same result when the RUN command is typed:[1]

```
RUN
 85
READY
```

Since EduSystem 20's memory is being shared by several users, this multiple-statement capability is helpful when writing long programs. Statements require less storage in the computer when they are grouped as a single statement.

[1] Notice that EduSystem 20 BASIC does not require the use of an END statement.

Immediate Mode

EduSystem 20 allows certain BASIC statements to be used in immediate mode, that is, to be issued and executed immediately without being included in a formal program. Statements commonly used with immediate mode are PRINT, LET, FOR, and NEXT. Immediate mode is a quick way to calculate expressions and equations. For example, the statement:

```
PRINT SQR(144)
```

followed by the RETURN key, causes the square root of 144 to be printed immediately, as follows:

```
12
READY
```

BASIC then prints READY to indicate that another immediate mode statement or a program may be entered. Immediate mode statements are not stored in computer memory.

Typing multiple statements per line is especially useful in the immediate mode. A table of random numbers could, for example, be generated by typing the following single line and pressing RETURN:

```
FOR D=1 TO 20\PRINT RND(0),\NEXT D
.2431684     .2988412     .7295008     .3125257     .3095865
.04493979    .4834217     .4961024     .5010026     .04103271
.2373254     .3046887     .1923863     .9121199     .241212
.9882844     .2587987     .03323189    .8701425     .9218898
READY
```

Nearly all the BASIC statements and commands may be executed in immediate mode. This is an excellent way to introduce students to the BASIC language, as the statements and commands can be exercised and understood *before* the student begins programming.

Immediate mode statements can also be used with programs. For example, an immediate GOTO statement may be used to start a program at a point other than the beginning. This is accomplished by loading the program into computer memory and typing; for example:

```
GOTO 50
```

After the RETURN key is pressed, the program execution will begin automatically at line number 50. In this case, the RUN command need not be typed.

INPUT Statement

The INPUT statement described in Chapter 1 allows a number, or numbers, to be entered from the TTY as values for variables. EduSystem 20 allows the user to respond to the INPUT query (?) with a value or mathematical expression. An expression may contain one or more arithmetic operations and may use any BASIC function. For example, the statement:

```
100 INPUT A
```

could be answered in any of the following ways:

```
?48
?10*(146+128)/3
?55↑2+SQR(121)
```

This capability could be used to enable one program to solve more than one problem.

As explained in Chapter 1, the INPUT statement may have multiple inputs. These inputs may be either mathematical expressions or numeric values. For example, the statement:

```
10 INPUT A,B,C
```

could be answered as follows:

```
?33,LOG(33),33↑5
```

Comments

Previously, the use of the REMARK (or REM) command to introduce a comment on a single line was discussed. Comments may also be appended to any line by starting the comment with a single apostrophe ('). All characters typed after the apostrophe on

a single line are ignored when the program is executed. For example:

```
10 LET X=4 'SET X TO ITS INITIAL VALUE
20 GOTO 10 'LOOP BACK TO START
```

When included within a PRINT statement message, the apostrophe is not considered as the start of a comment. For example:

```
12 PRINT "X'S VALUE IS";X
```

prints

```
X'S VALUE IS 4
```

When responding to an INPUT statement, the user may add a comment which will print on the Teletype but have no effect on the running program. For example:

```
10 INPUT A
20 PRINT A
RUN
```

Subscripted Variables

In chapter 1, the DIM statement is used to permit subscripts of more than 10. EduSystem 20 BASIC defines all variables as they occur, so the DIM statement is not necessary. The system imposes the following limits on subscript size:

- Single subscripts: 0 to 2047
- Double subscripts: 0 to 63 for each subscript

Consider the following example. Notice that a variable has a value of 0 until it is assigned a value.

```
10 REM - MATRIX CHECK PROGRAM
20 FOR I=0 TO 6
22 LET A(I,0)=I
25 FOR J=0 TO 10
28 LET A(0,J)=J
30 PRINT A(I,J);
35 NEXT J
40 PRINT
45 NEXT I
RUN
 0   1   2   3   4   5   6   7   8   9   10
 1   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   0   0   0   0
 6   0   0   0   0   0   0   0   0   0   0

READY
```

Some users have complained about ambiguity of above. In no configuration are 2047 vbls. allowed, or a 63×63 matrix! Please make this clear in a future edition (dependence on core size)

IF THEN Statement

The IF THEN statement described in Chapter 1 is used to transfer conditionally from the normal order of program execution. For example:

```
50 IF X>Y THEN 200
```

transfers control to line 200 if X is greater than Y. If X is not greater than Y, control transfers to the line following line 50. Edu-System 20 BASIC uses the IF THEN statement in this way to change the order of program execution. It also allows the IF THEN statement to conditionally perform an operation without changing the order of program execution. For example:

```
50 IF X>Y THEN PRINT "X>Y"
```

If X is greater than Y, BASIC prints X > Y. Whether or not X is greater than Y, the next statement executed is the one following line 50. IF THEN used in this manner does not change the order of program execution.

The IF THEN statement can also be used to transfer control to a subroutine. For example:

```
25 IF A+B=C THEN GOSUB 100
```

If A + B equals C, control is transferred to the subroutine at line 100.

Truncation Function (FIX)

EduSystem 20 BASIC includes all the functions discussed in Chapter 1 (see Table 5-2). In addition, it includes the truncation function (FIX) which returns the integer part of *x*. For example:

```
PRI FIX(-842.756)
-842
```

Notice that FIX is like INT for positive arguments. In fact, FIX could be defined as:

$$FIX(X)=SGN(X)*INT(ABS(X))$$

actually the definition is the other way round, internally, INT is defined in terms of FIX. So what.

CHR$ Function

Occasionally, it is desirable to type a character other than the printing ASCII set (see Appendix B) or to convert ASCII character codes to their respective characters. A special PRINT statement function, CHR$, is used for this purpose. This function takes as its argument a single constant or variable and prints the single character whose character code corresponds to that value. For example:

```
10 FOR I=0 TO 9
20 PRINT CHR$(I+48);
30 NEXT I
RUN
0123456789
READY
```

ALSO INPUT CHR$(34),X ↑ works in INPUT, although this is limited in usefulness, so does TAB

=?

This program prints "0123456789" because 48 to 57 are the ASCII values for the characters 0 through 9. The following special characters can also be printed using the CHR$ function:

TO PRINT "NULL" 5-9 use CHR$(256)

NOTE: User-defined function definitions and DATA statements must be alone on a line!

Bell	CHR$(7)
Line feed	CHR$(10)
Carriage return	CHR$(13)
Quote (")	CHR$(34)
Back arrow (←)	CHR$(95)
Form feed	CHR$(12)

For each ASCII character there is a second acceptable form of CHR$ function. The second code is obtained by adding 128 to the ASCII code given in Appendix B. For example, both CHR$(65) and CHR$(193) cause the character A to be output.

ON-GOTO Statement

The ON-GOTO statement permits the program to transfer control to one of a set of lines depending on the value of a formula. The statement is of the form:

line number ON *formula* GOTO *line number, line number . . .*

The formula is evaluated and then truncated to an integer. This integer is used as an index to tell which of the line numbers receives control. If the integer is 1, the first line number is used; if it is 2, then the second is used; etc. Obviously, the formula after truncation cannot be zero or negative or greater than the number of line numbers in the list. For example:

```
10 ON A+2 GOTO 100,200,300,400
```

If A is 2, then control is passed to line 400. The range A can have in this example is −1 to 2.

ON GOSUB Statement

The GOSUB and RETURN statements are used to allow the user to transfer control to a subroutine and return from that subroutine to the normal course of program execution (see Chapter 1). The ON GOSUB statement is used in the same manner as the ON GOTO statement described previously. The statement is of the form:

line number ON *formula* GOSUB *line number, line number . . .*

The formula is evaluated and then truncated to an integer. Depending on the value of the integer, control is transferred to the subroutine which begins at one of the line numbers listed. When the RETURN statement is encountered, control transfers to the line following the ON GOSUB line. For example:

```
50 ON X GOSUB 200,300,400
```

If X is 1, 2, or 3, control will transfer to line 200, 300, or 400 respectively. If X is not equal to 1, 2, or 3, line 50 is ignored.

This is correct
NO ERROR occurs!!! delete this!

~~ERROR 28 IN 50~~

RANDOMIZE Statement

The random number (RND) function produces a random number between 0 and 1, as described in Chapter 1. A given program that uses the RND function always produces the same random numbers each time it is run. If the user wants the random number generator to calculate different random numbers every time the program is run, EduSystem 20 BASIC provides the RANDOMIZE statement. RANDOMIZE is normally placed at the beginning of a program which uses the RND function. When the program is executed, RANDOMIZE causes the RND function to choose a random starting value so that the same program will give different results each time it is run. For example:

```
10 RANDOMIZE
20 PRINT RND(0)
```

prints a different number each time it is run. For this reason, it is good practice to debug (detect, locate, and correct mistakes) a program completely before inserting the RANDOMIZE statement.

To demonstrate the effect of the RANDOMIZE statement on two runs of the same program, the RANDOMIZE statement was inserted as line 15 in the following example:

```
15 RANDOM
20 FOR I=1 TO 5
25 PRINT "VALUE" I "IS" RND(0)
30 NEXT I
RUN
VALUE 1 IS .2431684
VALUE 2 IS .2988412
VALUE 3 IS .7295008
VALUE 4 IS .3125257
VALUE 5 IS .3095865

READY

RUN
VALUE 1 IS .6181684
VALUE 2 IS .4238412
VALUE 3 IS .1045008
VALUE 4 IS .4375257
VALUE 5 IS .6845865

READY
```

ERROR MESSAGES

EduSystem 20 checks all statements and commands before executing them. If it cannot execute a statement or command, it informs the user by printing an error message and the line number in which the error was found. Error messages for EduSystem 20 are shown in Table 5-3.

Table 5-3. EduSystem 20 Error Messages

Message	Explanation
WHAT?	Immediate mode statement or command not understood. It does not begin with a line number and is not a valid system command.
ERROR 1	Log of negative or zero number was requested.
ERROR 2	Square root of negative number was requested.
ERROR 3	Division by zero was requested.
ERROR 4	Overflow—exponent greater than approximately +38.
ERROR 5	Underflow—exponent less than approximately —38.
ERROR 6	Line too long or program too big.
ERROR 7	Characters are being typed in too fast; use TAPE command for reading paper tapes.

Table 5-3 (Cont.). EduSystem 20 Error Messages

Message	Explanation
ERROR 8	System overload caused character to be lost.
ERROR 9	Program too complex or too many variables. (GOSUB, FOR, or user defined function calls are too deeply nested.)
ERROR 10	Missing or illegal operand or double operators.
ERROR 11	Missing operator before a left parenthesis.
ERROR 12	Missing or illegal number.
ERROR 13	Too many digits in number.
ERROR 14	No DEF for function call.
ERROR 15	Missing or mismatched parentheses or illegal dummy variable in DEF.
ERROR 16	Wrong number of arguments in DEF call.
ERROR 17	Illegal character in DEF expression.
ERROR 18	Missing or illegal variable.
ERROR 19	Single and double subscripted variables with the same name.
ERROR 20	Subscript out of range.
ERROR 21	No left parenthesis in function.
ERROR 22	Illegal user defined function—not FN followed by a letter and a left parenthesis.
ERROR 23	Mismatched parentheses or missing operator after right parenthesis.
ERROR 24	Syntax error in GOTO.
ERROR 25	Syntax error in RESTORE.
ERROR 26	Syntax error in GOSUB.
ERROR 27	Syntax error in ON.
ERROR 28	~~Index out of range in ON.~~ NO! Try things to find out. But it ain't this !!
ERROR 29	Syntax error in RETURN.
ERROR 30	RETURN without GOSUB.
ERROR 31	Missing left parenthesis in TAB function.
ERROR 32	Syntax error in PRINT.
ERROR 33	An unavailable device was requested by the user.
ERROR 34	Missing or illegal line number.
ERROR 35	Attempt to GOTO or GOSUB to a nonexistent line.
ERROR 36	Missing or illegal relation in IF.
ERROR 37	Syntax error in IF.
ERROR 38	Missing equal sign or improper variable left of the equal sign in LET or FOR.
ERROR 39	Subscripted index in FOR.
ERROR 40	Syntax error in FOR.
ERROR 41	FOR without NEXT.
ERROR 42	Syntax error in LET.
ERROR 43	Syntax error in NEXT.

Table 5-3 (Cont.). EduSystem 20 Error Messages

Message	Explanation
ERROR 44	NEXT without FOR.
ERROR 45	Too much data typed in or illegal character in DATA or the data typed.
ERROR 46	Illegal character or function in INPUT or READ.
ERROR 47	Out of data.
ERROR 48	Unrecognized command: RUN mode.

NOTE

To correct the error indicated by the message, the appropriate line in the program must be corrected in the manner described in the Program Editing Section.

PROGRAM EDITING

There are two times when a program may require editing procedures. The first occurs while a line is being typed but before the RETURN key is pressed. The second occurs when a line has been completely typed and the RETURN key has been pressed. Each situation has its own editing procedures.

Situation 1: Before the RETURN key is pressed.

Three keys may be used to correct typing errors: ← (SHIFT/O), RUBOUT, or ALT MODE (or ESC).

← (back arrow), SHIFT/O on the keyboard, or RUBOUT is used to delete a character from a line. BASIC prints the back arrow, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

ALT MODE (ESC on some Teletypes) is used to delete an entire line. When this key is used, BASIC prints $DELETED, erases that line from the program, and returns the carriage so that the line may be retyped.

Situation 2: After the RETURN key is pressed.

Once a line of the program has been transmitted to computer memory via the RETURN key, several methods of correction may be used. Lines may be inserted, deleted, or changed.

INSERTION

To add a line to a program, assign a line number that falls between two existing lines, type the line number and text, and press RETURN.

DELETION

To erase a line from computer memory, type the line number *only* and press the RETURN key. The DELETE command may also be used to erase lines from memory. To erase a single line, type DELETE and the line number and press RETURN. DELETE followed by two line numbers separated by a comma erases all lines between and including the two given. For example:

```
DELETE 10,20
```

erases lines 10 through 20.

CHANGE

Old instructions may be replaced by new ones by retyping the line. This procedure is adequate for changing simple lines. When, however, the line contains a complex formula or a long message to be printed, it may be changed with the EDIT command. The EDIT command allows the user to access a single line and search for the character or characters to be changed. To use this command, type:

EDIT *line number*

and press the RETURN key. BASIC waits for a search character to be typed (BASIC does not print this search character when it is typed). This search character is one that already exists on the line to be changed. After the search character is typed, BASIC prints out the contents of that line until the search character is printed. At this point, printing stops and the user has the following options:

- Type in new characters; BASIC inserts them following the ones already printed.
- Type a form feed (CTRL/L) to cause the search to proceed to the next occurrence, if any, of the search character.
- Type a bell (CTRL/G) to signal a change of search character, then type a new search character.
- Use the RUBOUT or ← key to delete one character to the left each time the key is pressed.

- Type the RETURN key to terminate editing of the line at that point, deleting any text to the right.
- Type the ALT MODE key to delete all the characters to the left except the line number.
- Type the LINE FEED key to terminate editing of the line, saving the remaining characters.

When the EDIT operation is complete, BASIC prints READY. Note that line numbers cannot be changed with the EDIT command.

The following example demonstrates the EDIT command. An incorrect line was typed:

```
60 PRINT "PI=3.14146 ABOUZI"
```

The line was edited as follows:

```
EDIT 60
PRINT "PI=3.14146←←59 ABOUZ←TI"
```

First 6 was entered as the search character. BASIC printed the line to the 6, and the RUBOUT key was typed twice to remove the two incorrect digits (46) and 59 was inserted in the line. CTRL/G was typed and another search character (%) was entered. BASIC printed characters to the % which was removed with a RUBOUT and replaced with a T. The LINE FEED key was typed to terminate the edit and save the remaining characters. If the line is listed, the following is printed on the Teletype.

```
LIST 60

  60 PRINT "PI=3.14159 ABOUTI"

READY
```

OPERATING INSTRUCTIONS

Loading EduSystem 20 BASIC

A paper tape labeled EDUSYSTEM 20 BASIC is provided with the system. This tape, called the system tape, must be loaded into computer memory when the system is installed. Once this system tape is loaded, it need not be reloaded.

The system tape may be loaded in one of two ways: with an automatic loader (hardware bootstrap) or with the Read-In Mode (RIM) loader program. The following loading instructions are for an EduSystem 20 that includes a hardware bootstrap. If the EduSystem 20 does not have a hardware bootstrap, see Appendix A for instructions on using the RIM loader.

1. Turn the key lock on the computer console to POWER.
2. Turn all Teletypes to LINE.
3. Set all Teletype tape readers to FREE.
4. Place the system tape (or configurator tape) in the appropriate reader (high-speed or console Teletype) with the leader code (ASCII 200) over the reader head.
5. Set the SWITCH REGISTER (SR) to 5356 (octal).[2]
6. Press and raise the HALT switch.
7. Turn on the appropriate paper tape reader.
8. Press and raise the SW switch.

The system tape should begin to read in. If it does not, ensure that the correct tape is being used and repeat the above procedures. When the tape has read in, the message:

```
EDUSYSTEM 20  BASIC
```

prints on the console Teletype. If this message is not printed, the system tape did not read in correctly and the loading procedures must be repeated.

When EDUSYSTEM 20 BASIC is printed, perform the following:

9. Remove the tape from the tape reader and turn the reader off.
10. Answer BASIC's initial dialog as explained below.

Initial Dialog

When BASIC has been loaded correctly and has printed the identification message (EDUSYSTEM 20 BASIC), it begins to ask

---

[2]An explanation of the octal, or base 8, number system is included in *Introduction to Programming 1972.*

certain questions which the user must answer. The first question, printed two lines below the identification message, is:

```
NUMBER OF USERS (1 TO 8)?
```

The user responds with a single digit from 1 to 8, depending on the number of Teletype terminals to be used. If the number of users is 1, BASIC asks whether the user has a high-speed reader or punch and concludes the dialog. In this case, the entire dialog might read as follows:

```
NUMBER OF USERS (1 TO 8)?1
DO YOU HAVE A HIGH SPEED PUNCH (Y OR N)?Y
DO YOU HAVE A HIGH SPEED READER (Y OR N)?Y
IS THE ABOVE CORRECT (Y OR N)?Y
```

If the number of users is more than 1, BASIC continues the dialog by asking:

```
PDP-8/L COMPUTER (Y OR N)?
```

If the EduSystem 20 computer is a PDP-8/L, respond Y. When the computer is not a PDP-8/L, respond N; a response of N to this question prompts the following dialog:

```
STANDARD REMOTE TELETYPE CODES (Y OR N)?
```

Standard PT08 or KL8-E device codes are 40, 42, 44, 46, 50, 52, and 54. When a system using PT08 or KL8-E interface units is first installed, determine the specific device code for each Teletype and label each Teletype with its specific device code. If device codes are standard, enter Y for this question and BASIC assumes the standard device codes and continues the dialog. If device codes are not standard, enter N; BASIC then asks:

```
TELETYPE #1 DEVICE CODE?
```

BASIC asks this question for each Teletype to be used, up to seven times for an 8-user system. Respond with the specific 2-digit device code for each Teletype.

When the device codes have been determined, or if the computer is a PDP-8/L, the dialog continues as follows:

```
DO YOU HAVE A HIGH SPEED PUNCH (Y OR N)?
```

Respond Y if a high-speed paper tape punch is present, N if not; BASIC then asks:

```
DO YOU HAVE A HIGH SPEED READER (Y OR N)?
```

Respond Y if a high-speed paper tape reader is present, N if not. If 1 was specified as the number of users, this question completes the BASIC dialog. However, for more than one user, BASIC asks the following question:

```
SAME AMOUNT OF STORAGE FOR ALL USERS?N
```

The above question requires the user to decide whether to partition the available core equally among the users on the EduSystem 20. If Y is the response, BASIC determines the size of the core memory on the system and divides it equally among the users and ends the dialog. If N is the response, BASIC determines the size of the core memory on the EduSystem 20, subtracts the amount used by EduSystem 20 software (4K), and prints the highest core field according to the following:

Field 7—32K core memory
Field 6—28K core memory
Field 5—24K core memory
Field 4—20K core memory
Field 3—16K core memory
Field 2—12K core memory
Field 1— 8K core memory

For explanation purposes, the following dialog is written for a 12K, 8-user EduSystem 20. The system core is to be allocated as follows:

User 1—9 blocks (user 1 is the console terminal)
User 2—4 blocks
User 3—3 blocks

User 4—4 blocks
User 5—3 blocks
User 6—3 blocks
User 7—3 blocks
User 8—3 blocks

Each core field contains 16 blocks; a core field may be divided among several users, but no user may be allotted blocks in more than one core field. To determine the number of blocks, BASIC prints the following dialog and the user answers as shown:

```
FIELD 2
THERE ARE 16 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #1 WILL BE HOW MANY BLOCKS?9
THERE ARE Ø7 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #2 WILL BE HOW MANY BLOCKS?4
THERE ARE Ø3 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #3 WILL BE HOW MANY BLOCKS?3
FIELD 1
THERE ARE 16 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #4 WILL BE HOW MANY BLOCKS?4
THERE ARE 12 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #5 WILL BE HOW MANY BLOCKS?3
THERE ARE Ø9 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #6 WILL BE HOW MANY BLOCKS?3
THERE ARE Ø6 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #7 WILL BE HOW MANY BLOCKS?3
THERE ARE Ø3 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #8 WILL BE HOW MANY BLOCKS?3
```

When an invalid response is made to any of BASIC's questions, an error message is printed and the question is repeated. For example:

```
STANDARD REMOTE TELETYPE CODES (Y OR N)?4
INVALID RESPONSE
STANDARD REMOTE TELETYPE CODES (Y OR N)?
```

When all responses have been entered, BASIC asks:

```
IS THE ABOVE CORRECT (Y OR N)?
```

If an incorrect response was made, answer N and BASIC begins the dialog again. A response of Y ends the dialog and BASIC prints:

```
END OF DIALOGUE

READY
```

BASIC prints READY on each of the Teletypes associated with the specified device codes. EduSystem 20 is now ready to process BASIC programs. At this time, turn the key lock to PANEL LOCK and remove the key to prevent the system from being disturbed.

System Reconfiguration

The EduSystem 20 Configurator Tape is used to change the number of users, allocation of core fields, etc., without completely reloading the System tape. To use the Configurator tape, the system must be inactive, i.e., BASIC must not be running a program and no user typing. CTRL/C is typed to stop a running program or the listing of a program. To ensure that no one starts typing, turn all Teletypes to OFF. When the system is inactive, load the Configurator Tape as explained under Loading the System.[3]

System Shutdown

If power failure detection hardware is available on the system, simply turn the console key lock to OFF. Otherwise, to shut the system down, for overnight or any reason, ensure that the system is inactive, as explained above. Then press the HALT switch and turn the key lock to OFF.

System Restart

If power failure detection hardware is available on the system, simply turn the key lock to PANEL LOCK. Otherwise, perform the following procedures.

1. Turn the key lock to POWER.
2. Set the SR to 0000 and press EXTD ADDR LOAD.
3. Set the SR to 0200 and press ADDR LOAD.
4. Press the CLEAR switch, then the CONT switch.
5. Turn the appropriate Teletypes to LINE.
6. Turn the key lock to PANEL LOCK.

EduSystem 20 is now ready to process BASIC programs.

Program Storing Procedures

Once a program has been typed in correctly, it may be saved on paper tape so that it may be reloaded quickly. Programs may be

---

[3]If a program other than EduSystem 20 BASIC has been loaded into memory since the last use of BASIC, the system tape must be reloaded.

punched on the Teletype (TTY) or high-speed punch. To save the program, perform the following procedures.

TELETYPE PAPER TAPE PUNCH

1. Turn the TTY control knob to LINE.
2. Type TAPE; press RETURN.
3. Turn the TTY paper tape punch ON.
4. Type LIST; press RETURN.
5. When punching is complete, turn TTY punch OFF.
6. Type KEY; press RETURN.

HIGH-SPEED PUNCH

1. Turn the TTY control knob to LINE.
2. Type PTP; press RETURN.
3. Turn high-speed punch to ON.
4. When punching is complete, turn punch OFF.
5. Type KEY; press RETURN.

Program Reloading Procedures

Programs saved on paper tape may be reloaded using the Teletype (TTY) or high-speed paper tape reader. To reload programs, perform the following procedures.

TELETYPE PAPER TAPE READER

1. Turn the TTY paper tape reader to FREE.
2. Turn the TTY control knob to LINE.
3. Insert tape in the reader.
4. Type TAPE; press RETURN.
5. Turn the TTY reader to START.
6. When the tape has read in, turn the TTY reader to FREE.
7. Type KEY; press RETURN.

HIGH-SPEED READER

1. Turn the high-speed reader to ON.
2. Turn the TTY control knob to LINE.
3. Insert tape in the reader.
4. Type PTR; press RETURN.
5. When the tape has read in, turn the high-speed reader OFF.
6. Type KEY; press RETURN.

chapter 6
edusystem 25

INTRODUCTION

EduSystem 25 is a multi-user system for up to eight persons using an extended BASIC language called BASIC-E. In a time-sharing environment, users can store programs and data on DECtape (or the RK8-E disk), greatly reducing the time involved in loading and punching paper tape and making available more access time to a greater number of students. The system operates in either immediate or programmable mode and allows multiple statements per line for more efficient coding. In addition, extended BASIC features enable the use of alphanumeric strings, program chaining, and numerous other extensions to Dartmouth standard BASIC.

EduSystem 25 also allows a maximum of 5 persons at a time to use the FOCAL language, though not simultaneously with BASIC-E. And, when operated as a one-user system, EduSystem 25 runs FORTRAN, assembly language, and all of the OS/8 utility programs. (OS/8 is the operating system for the PDP-8.)

System Components

EduSystem 25 is composed of a PDP-8/E computer with 12,288 (12K) words of core memory, powerfail protection, automatic loader (hardware bootstrap), DECtape, and up to five terminals with their associated interfacing. Each standard EduSystem 25 includes the BASIC-E language processor on DECtape, a library of sample programs, textbooks, and curriculum guides. An additional 4096 (4K) words of core memory enable EduSystem 25 to handle up to three more terminals (using BASIC-E) for a total of eight.[1] Additional core memory (up to 32K) can be added to provide larger user program areas. The system can support an RK8E Disk and TD8-E DECtape in place of TC08 DECtape.

[1] Version 1 of EduSystem 25 supports only 5 users.

note: USER-defined function definitions and DATA statements have to be alone on a line.

System Expansion

EduSystem 25 grows to EduSystem 45[2] with the addition of an optical card reader and an EduSystem 30 software kit. To expand EduSystem 25 to an eight-user EduSystem 50 time-sharing system, add RF/RS08 DECdisk, 4096 (4K) words of core memory, and the necessary terminals.

BASIC LANGUAGE CAPABILITIES

EduSystem 25 includes all the standard elements of BASIC as explained in Chapter 1. Tables 6-1 and 6-2 summarize the EduSystem 25 BASIC-E language capabilities. The system also includes some additional features not discussed in Chapter 1. These additional features are discussed below.

Abbreviated Commands

All commands and statement keywords can be abbreviated to the first three letters, as shown in Table 6-1.

Multiple Statements Per Line

EduSystem 25 allows more than one statement to be typed on a single line. Subsequent statements begin with a backslash character (\), which is SHIFT/L on the keyboard. A program is often more understandable when statements are grouped together on the same line. The following are examples of multiple statements per line.[3]

```
10 FOR I=1 TO 10\PRINT I,SQR(I)\NEXT I
```

```
10 PRINT "WHAT IS YOUR NUMBER";\INPUT X
20 PRINT "THE SQUARE ROOT IS"; SQR(X)
30 PRINT\GOTO 10
```

no! Must NEXT

The only restriction to the use of multiple statements per line is that when a NEXT or GOSUB statement is used it must be the last statement on the line. Since the EduSystem 25 memory is shared by several users, this multiple-statement capability is helpful when writing long programs. Statements require less storage when they are grouped on a single line.

---

[2] EduSystem 45 is a combination of EduSystems 25 and 30, capable of running in either batch or multi-user mode.

[3] Notice that EduSystem 25 does not require the END statement.

Immediate Mode

EduSystem 25 allows most BASIC statements to be used in immediate mode, that is, to be issued and executed immediately without being included in a formal program. Nearly all BASIC statements can be executed in immediate mode. This is an excellent way to introduce students to the BASIC language, as the statements can be used and understood individually *before* the student begins programming.

For example, the statement:

```
PRINT 2*5+SQR(7.8314)
```

followed by the RETURN key, causes the problem solution to be printed immediately. BASIC then prints READY to indicate that another immediate mode statement or a program may be entered. Immediate mode statements are not stored in computer memory.

Typing multiple statements per line is especially useful in the immediate mode. A table of random numbers could, for example, be generated by typing the following single line and pressing RETURN:

```
FOR D=1 TO 20\PRINT RND(0),\NEXT D
 .2431684    .2988412    .7295008    .3125257    .3095865
 .04493979   .4834217    .4961024    .5010026    .04103271
 .2373254    .3046887    .1923863    .9121199    .241212
.9882844    .2587987    .03323189   .8701425    .9218898
```

Immediate mode statements can also be used with programs. For example, an immediate GOTO statement may be used to start a program at some point other than the beginning. This is accomplished by loading the program into computer memory and typing, for example:

```
GOTO 75
```

After the RETURN key is pressed, the program execution begins automatically at line number 75. In this case, the RUN command need not be typed. This immediate GOTO command is especially helpful in debugging by halting a running program, examining the

values of variables (and possibly changing them), and then restarting the program at any point.

INPUT Statement

The INPUT statement described in Chapter 1 allows a number, or numbers, to be entered from the Teletype as values for variables. EduSystem 25 allows the user to respond to the INPUT query (?) with a value or mathematical expression. An expression may contain one or more arithmetic operations and may use any BASIC function. This capability could be used to enable one program to solve more than one problem. For example, the statement:

```
100 INPUT X
```

could be answered with any of the following values or expressions:

```
?48
?7*SQR(495)
?SIN(8.4221)
```

note INPUT syntax, past chapter

As explained in Chapter 1, the INPUT statement may have multiple inputs. These inputs may be either mathematical expressions or numeric values. For example, the statement:

```
10 INPUT A,B,C
```

could be answered as follows:

```
?48,48↑5,SQR(48)
```

Comments

Previously, the use of the REMARK (or REM) statement to introduce a comment on a single line was discussed. Comments may also be appended to any line by starting the comment with a single apostrophe ('). All characters typed after the apostrophe are ignored when the program is executed. For example:

```
10 LET X=4 'SET X TO ITS INITIAL VALUE
20 GOTO 10 'LOOP BACK TO START
```

When responding to an INPUT statement, the user may add a comment which prints on the terminal but has no effect on the running program. For example:

```
10 INPUT X
20 PRINT X
RUN
? 2 'LET X BE 2
 2

READY
```

IF THEN Statement

The IF THEN statement described in Chapter 1 is used to transfer conditionally from the normal order of program execution. For example:

```
25 IF A<=B THEN 100
```

transfers control to line 100 if A is less than or equal to B. If A is greater than B, control transfers to the line following line 25. BASIC-E uses the IF THEN statement in this way to change the order of program execution. It also allows THEN to be followed by an executable BASIC statement. This statement is executed only if the IF relation is true; otherwise control passes to the next separate statement. For example:

```
50 IF 2+2=4 THEN LET A=7\PRINT A
```

Since the statement (2+2=4) is true, BASIC prints 7. The next statement executed is the one following line 50. IF THEN used in this manner does not change the order of program execution.

The IF THEN statement can also be used to transfer control to a subroutine. For example:

```
25 IF A+B=C THEN GOSUB 100
```

If A+B equals C, control is transferred to the subroutine at line 100. Otherwise, the next statement executed is the one following line 25.

ON-GOTO Statement

The ON-GOTO statement permits the program to transfer control to one of a set of lines depending on the value of a formula. The statement is of the form:

line number ON *formula* GOTO *line number,line number. . .*

The formula is evaluated and then truncated to an integer. This integer is used as an index to tell which of the line numbers receives control. If the integer value of the formula is 1, the first line number is used; if the value is 2, then the second is used; etc. Obviously, the formula after truncation cannot be zero or negative or greater than the number of line numbers in the list. For example:

```
10 ON X+2 GOTO 100,200,300,400
```

If X is 2, then control passes to line 400. The allowable range of X in this example is −1 to 2, so that the range of the formula value is 1 to 4.

ON GOSUB Statement

The GOSUB and RETURN statements are used to allow the user to transfer control to a subroutine and return from that subroutine to the normal course of program execution (see Chapter 1). The ON GOSUB statement is used in the same manner as the ON GOTO statement described previously. The statement is of the form:

line number ON *formula* GOSUB *line number,line number. . .*

The formula is evaluated and then truncated to an integer. Depending on the value of the integer, control is transferred to the subroutine which begins at one of the line numbers listed. When the RETURN statement is encountered, control transfers to the line following the ON GOSUB statement. For example:

```
50 ON X GOSUB 200,300,400
```

If X is 1, 2, or 3, control transfers to line 200, 300, or 400, respectively. If X is not equal to 1, 2, or 3, line 50 is ignored.

RANDOMIZE Statement

The random number (RND) function produces a random number between 0 and 1, as described in Chapter 1. A given program using the RND function produces the same random numbers each time it is run. If the user wants the random number generator to calculate different random numbers every time the program is run, EduSystem 25 BASIC provides the RANDOMIZE statement. RANDOMIZE is normally placed at the beginning of a program which uses the RND function. When the program is executed, RANDOMIZE causes the RND function to choose a random starting value so that the same program gives different results each time it is run. For example:

```
10 RANDOMIZE
20 PRINT RND(0)
```

prints a different number each time it is run. For this reason, it is good practice to debug (detect, locate, and correct mistakes) a program completely before inserting the RANDOMIZE statement.

To demonstrate the effect of the RANDOMIZE statement on two runs of the same program, the RANDOMIZE statement was inserted as line 15 below:

```
15 RANDOM
20 FOR I=1 TO 5
25 PRINT "VALUE" I "IS" RND(0)
30 NEXT I
RUN
VALUE 1 IS .3808637
VALUE 2 IS .7119271
VALUE 3 IS .9687586
VALUE 4 IS .03029916
VALUE 5 IS .4629068

READY

RUN
VALUE 1 IS .8916059
VALUE 2 IS .2441537
VALUE 3 IS .3154383
VALUE 4 IS .07033823
VALUE 5 IS .583024

READY
```

Truncation Function (FIX)

EduSystem 25 BASIC includes all the functions discussed in Chapter 1 (see Table 6-2). In addition, EduSystem 25 includes the truncation function (FIX) which returns the integer part of the function argument. For example:

```
PRI FIX(82+100.6775)
 182
PRI FIX(SQR(85.54621))
 9
```

Notice that FIX is like INT for positive arguments. In fact, FIX could be defined as:

FIX(X)=SGN(X)*INT(ABS(X))

EXTENDED SYSTEM CAPABILITIES

In addition to the BASIC language features previously discussed, EduSystem 25 has several extended features that allow the user to write longer, more complex programs. These features include the ability to store programs on DECtape, create and manipulate data files on DECtape, and link program segments together to allow longer user programs. Another extended capability allows character strings to be input, manipulated, compared, and output. These extended features are discussed below.

String Variables

EduSystem 25 BASIC has the ability to manipulate alphanumeric information (commonly called strings). A string is a sequence of six or fewer printing ASCII characters (see Appendix B). If a string contains more than six characters, only the first six are retained. A string variable is signified by one letter followed by the dollar sign ($) character and, optionally, by one or two subscripts. The following are all acceptable string variables:

A$
B$(1)
C$(2,5)

READING STRING DATA

Strings of characters may be read into string variables from DATA statements. Each string data element is composed of one to six characters enclosed in quotation marks. The quotation marks are not part of the actual strings. For example:

Machine puts leading blank 6-8n first string "READ" from a DATA line. NASTY.

```
10 READ A$,B$,C$
20 DATA "JONES","SMITH","TAYLOR"
```

The string JONES is read into A$, SMITH into B$, and TAYLOR into C$. If the string contains more than six characters, excess characters are ignored.

PRINTING STRINGS

The normal PRINT statement may be used to print string information. If a semicolon is used to separate string variables in a PRINT command, the strings are printed with no intervening spaces. For example, the program:

```
10 READ A$,B$,C$
20 PRINT C$;B$;A$
30 DATA "ING","SHAR","TIME-"
RUN
```

causes the following to print :

```
10 READ A$,B$,C$
20 PRINT C$;B$A$ DELETED

10 READ A$,B$,C$
20 PRINT C$;B$;C$ DELETED

READY
```

INPUTTING STRINGS

String information may be entered into a BASIC program by means of the INPUT command. Strings typed at the keyboard may contain any of the standard BASIC characters except back arrow (←) and quotation mark. Back arrow, as always, is used to delete the last character typed. Commas are used to separate string variables, as with numeric data. If a string contains a comma, the entire string must be enclosed in quotation marks. The following program demonstrates string input:

```
10 INPUT A$,B$,C$
20 PRINT A$,B$,C$
RUN
? JONES,SMITH,TAYLOR
JONES          SMITH          TAYLOR

READY
```

Strings and numeric information may be combined in the same INPUT statement as in the following example. Note that if an input string contains more than six characters, only the first six are retained.

```
10 INPUT A,A$,B$
20 PRINT A$,B$,A
RUN
? 01754,MAYNARD,MASS.
MAYNAR          MASS.           1754

READY
```

The numeric variable A is set to 1754 (leading zeroes are not printed). The string MAYNAR is put in the string variable A$, and the string MASS. is put into the string variable B$.

LINE INPUT

Strings of more than six characters may be entered by means of the LINPUT (line input) statement. A LINPUT statement is followed by one non-subscripted variable, e.g., A$. The following program illustrates the use of the LINPUT statement:

```
10 LINPUT A$
20 Z=A$(0)
30 FOR I=1 TO INT(Z/6+.9)
40 PRINT A$(I);
50 NEXT I
60 PRINT
RUN
? MAYNARD, MASS.  01754
MAYNARD, MASS.  01754

READY
```

This program inputs the entire line of type from the terminal and stores the characters so that the values of the string variables are, effectively:

A$(1) = MAYNAR
A$(2) = D, MAS
A$(3) = S. 01
A$(4) = 754
A$(0) = total number of characters stored

Commas and quotation marks are treated as ordinary characters and, hence, are stored in the string variables. LINPUT accepts a line of characters up to a carriage return.

WORKING WITH STRINGS

Strings may be used in both LET and IF THEN statements. For example:

```
10 LET Y$="YES"
20 IF B$<D$ THEN PRINT "WHOOPEE"
```

The first statement stores the string YES in the string variable Y$. The second prints WHOOPEE if B$ is less than (alphabetically prior) to D$. If B$ = "TED" and D$ = "MARY", the expression B$<D$ is false and the PRINT statement is not executed. For two strings to be equal, they must contain the same characters, in the same order, and be the same length. In particular, trailing spaces are significant as they change the length of the string. "YES" is not equal to "YES ".

The relational operators <, >, >=, <=, <> are used with string variables to represent alphabetic order; they could be used to alphabetize a list of strings. The arithmetic operations (+, −, *, /, ↑) are not defined for strings. Thus, statements such as LET A$ = 3*5 and LET C$ = A$+B$ have no meaning and should never be used in a BASIC program. Such statements do not cause an error message to be printed; however, the results of such operations are undefined.

STRING FUNCTIONS

EduSystem 25 BASIC contains several functions for use with character strings. These functions allow the program to access part of a string (MID), generate a numeric string or value (CHR$), determine the number of characters in a string (LEN), and link two strings together (CAT).

CHR$ Function

Occasionally, it is desirable to type a character other than the printing ASCII set (see Appendix B) or to convert ASCII character codes to their respective characters. A special PRINT statement function, CHR$, is used for this purpose. This function takes as its argument a single constant or variable and prints the single character whose character code corresponds to that value.

In Edu25, CHR$(0) produces "null".
In Edu20, you have to use CHR$(256)

For example:

```
10 FOR I=0 TO 9
20 PRINT CHR$(I+48);
30 NEXT I
RUN
0123456789
READY
```

This program prints "0123456789" because 48 to 57 are the ASCII values for the characters 0 through 9. The following special characters can also be printed using the CHR$ function:

Bell	CHR$(7)
Line feed	CHR$(10)
Carriage return	CHR$(13)
Quote (")	CHR$(34)
Back arrow (←)	CHR$(95)
Form feed	CHR$(12)

For each ASCII character there is a second acceptable form of CHR$ function. The second code is obtained by adding 128 to the ASCII code given in Appendix B. For example, both CHR$(65) and CHR$(193) cause the character A to be output.

MID Function

The MID function is used to obtain a portion of a character string. The format of this function is:

MID(A$,M,N)

BASIC accesses the characters in the string variable A$ and returns N characters, beginning with position M. Characters in a string are numbered 1 through 6. Consider the following example:

```
10 LET A$="UVWXYZ"
20 PRINT MID(A$,3,3)
RUN
WXY

READY
```

LEN Function

The LEN function returns the number of characters in the string argument. Characters in a string are numbered 0 through 6. For example:

```
10 R$="RANDOM"
20 PRINT LEN(R$)
RUN
 6

READY
```

CAT Function

The CAT function returns a string of characters which are the result of the concatenation (linking) of two other character strings. A maximum of six characters can be returned by the CAT function; if the first string to be linked contains six characters, only those six characters are returned. For example:

```
10 A$="012345"\B$="678910"
20 PRINT CAT(A$,B$)
RUN
012345

READY
```

If the first string contains less than six characters, they are linked as shown in the following program.

```
10 A$="EDU"\B$="-25"
20 PRINT CAT(A$,B$)
RUN
EDU-25

READY
```

Program Storage/Retrieval

EduSystem 25 allows the system DECtape to be used for permanent on-line program storage. Programs stored in this way may be loaded instantly without handling a paper tape or typing a lengthy program.

STORING USER PROGRAMS

User written programs may be stored on DECtape. Each program to be saved must have an assigned name, entered at the beginning of the programming session. The NEW command is given to clear any existing program and define the name of the new program to be entered. To use the NEW command, the user types:

```
NEW
```

and presses RETURN. The system then asks for:

```
NAME--
```

The user then types any name of 1 to 6 characters (the first of which must be alphabetic) and presses RETURN. BASIC assigns that name to the program to be entered. The user may change the name of the program at any time by typing the RENAME command. BASIC again asks for NAME— and assigns the new name to the program being entered. RENAME does not delete the existing program as does NEW.

Once a program has been named and typed correctly, it may be stored on DECtape with the SAVE command. Any existing program stored under the same name is deleted when a SAVE command is issued. Thus all stored programs have unique names which may be used to recall them in the future. To store a program, the user, having named and entered a program, types:

```
SAVE
```

and presses RETURN. BASIC stores the program on DECtape under the name specified with an extension identifying the user terminal from which the program was saved. For example, a program named ALPHA that is saved from terminal .E2 (see Starting EduSystem 25) is stored as ALPHA .E2. Extensions protect the programs from access by other terminals, i.e., terminal .E1 cannot access a program stored by terminal .E2.

When the program has been stored, BASIC prints READY. At this time, the program still resides in computer memory; storing does not erase computer memory. A NEW or SCR command must be typed to erase the program from memory after it has been stored on DECtape.

RETRIEVING USER PROGRAMS

User programs stored on DECtape are brought into memory with the OLD command. Only programs stored from the same user terminal can be retrieved at that terminal. When recalling a program with the OLD command, the user must give exactly the

same name as was used when the program was stored. To use the OLD command, the user types:

```
OLD
```

and presses the RETURN key. The system then asks for NAME—. The user can also use the OLD command and the program name (e.g., ALPHA) on the same line as follows:

```
OLD ALPHA
```

BASIC brings the stored program into memory but does not erase the program from DECtape. If the user wishes to modify the program, he can make the necessary modifications, then issue another SAVE command. This procedure replaces the original stored program on DECtape with the modified version of the same name.

The user can obtain a listing of all programs stored from his terminal by using the CATALOG command. CATALOG, like all other program storage/retrieval commands, is an immediate command. It is used by typing CATALOG and pressing RETURN. BASIC immediately prints a listing, without extensions, of all programs stored by the terminal from which the CATALOG command was issued.

RUNNING VERY LONG PROGRAMS

EduSystem 25 accommodates BASIC programs up to 250 lines long. If a program is any longer, it may be necessary to break it into several segments. A program that has been broken into more than one piece is called a *chained* program.

Each part of a chained program is saved on DECtape as a separate program, complete with name and system-assigned extension. The last statement of each part to be executed is a CHAIN statement specifying the name of the next section of the program. For example, a program named ALPHA is segmented into three parts: ALPHA, BETA, and GAMMA. ALPHA is loaded with the OLD command.

```
250 CHAIN "BETA"
```

The last statement in ALPHA is a CHAIN statement.
or

```
250 CHAIN B$
```

The string variable (B$) may be used if the segment name was previously assigned to it, for example:

```
248 B$="BETA"
```

When this CHAIN statement is reached, the BETA segment of the program is loaded and executed. The BETA segment then chains to the next (GAMMA) segment of the program with a statement such as:

```
399 CHAIN "GAMMA"
```

Each separate part of the program links automatically to the next part of the program.

The CHAIN command may also be used to branch to a stored program from a working program. For example:

```
10 INPUT "WHAT LESSON DO YOU WANT" A$
20 CHAIN A$
RUN
WHAT LESSON DO YOU WANT? DELTA
```

would cause BASIC to retrieve, load, and execute the program named DELTA.

DELETING STORED PROGRAMS

The UNSAVE command is used to delete a program already stored. UNSAVE must either be preceded by a NEW, OLD, or RENAME command which specifies the name of the program to be deleted or the command must be typed as:

```
UNSAVE ALPHA
```

specifying the name (e.g., ALPHA) of the program, on DECtape, to be deleted. The user must use exactly the same name used when

the program was stored. Only programs which were saved from the terminal being used may be deleted by a user at that terminal, e.g., a user at terminal .E1 may delete only those programs stored with an extension of .E1. (These are programs listed in response to the CATALOG command.)

USING PUBLIC LIBRARY PROGRAMS

Programs stored by the System Manager with the extension .E8 may be accessed by all users. These programs are collectively called the public library. To obtain a listing of the available public library programs, the user types:

```
CAT$
```

and presses the RETURN key. A listing of the public library programs is printed. A sample listing is shown below:

```
ROCKET
LUNAR
POLUTE
ELECT
SNOOPY
GAMBLE

READY
```

The user can access any program in the public library by typing OLD$ and the desired program name. For example, the user types:

```
OLD$ SNOOPY
```

The library program SNOOPY is loaded into memory and the system prints READY. The user can then execute SNOOPY by typing RUN or, if desired, can obtain a listing of SNOOPY by typing LIST. User may access the public library but cannot SAVE or UNSAVE the library programs. These programs are protected by the .E8 extension.

The user can also access public library programs by chaining to them from a program. For example, the user can access the public

library program ELECT by including the following statement in his program:

```
55 CHAIN$ "ELECT"
```

or by including the following statements:

```
50 E$="ELECT"
55 CHAIN$ E$
```

Data File Storage/Retrieval

Just as some programs may be too large to be executed as a single program, other programs may need to store and use more data than may be contained in DATA statements within the program. If this is the case, data may be stored on DECtape. Data stored in this way is called a data file.

EduSystem 25 allows users to read and write DECtape data files. These data files may contain numeric or string data. Data files are the perfect way to test student programs. The instructor may create a data file which all student programs must use and then answers may be stored in another file which may be checked automatically.

Data files are stored automatically by EduSystem 25 on the DECtape mounted on DECtape drive 1. Data files are stored with an extension similar in principle to that previously discussed for programs. Terminal .E1 data files have the extension .D1, .E2 data files use .D2, etc. The system manager's extension (.E8) uses .D8 data files. Normally, .D8 files can only be read; users are limited to reading only by placing DECtape drive 1 on WRITE LOCK.

CREATING DATA FILES

When the user desires to store data on DECtape, he must create a data file, or use an existing data file. Data files are created with the OPEN FOR OUTPUT command. For example, the user wishes to create a data file named DATA10 (date file names, like program file names, can be 1 to 6 characters long, the first character of which must be alphabetic). The data file name can be used in the OPEN statement or a string variable can represent it. A data

file would be created when the following statement was executed in a program:

```
10 OPEN "DATA10" FOR OUTPUT,12
```

The data file name must be enclosed in quotation marks. This statement could also be coded as:

```
10 OPEN A$ FOR OUTPUT,12
```

if A$ has been set equal to DATA10. The 12 at the end of the statement indicates that 12 blocks are to be reserved for this file. Any number of blocks up to 64 may be reserved in this manner. If no blocks are reserved in the OPEN FOR OUTPUT statement, the system assumes that the file is to be 10 blocks long.

When the data file has been created, or reopened, the user may write data into the file with the PRINT # statement. This statement may contain either numeric or string data separated by commas or semicolons. For example, a single data item can be entered by typing the following:

```
20 PRINT #, 248
```

This statement writes 248 and a carriage return/line feed onto a data file that has been opened for output. Several data items may be included in a PRINT # statement, with commas printed as separators between data items on each line of the output file. The commas must be enclosed in quotation marks to have them print on the output file. For example:

```
20 PRINT#,1,2,3,4,5
```

writes each of the above items on the data file as separate data items as follows: 1, 2, 3, 4, 5 and a carriage return/line feed exactly as the items would be input from the terminal.

Character strings are used in the PRINT # statement in the same manner, for example:

```
25 A$="JONES"\B$="SMITH"\C$="TAYLOR"
30 PRINT#,A$,B$,C$
```

would write three separate character strings onto the DECtape as follows: JONES, SMITH, TAYLOR and a carriage return/line feed. Writing PRINT # statements with multiple operands (data items) saves space on the DECtape data file because fewer carriage return/line feeds need to be stored.

When all data has been entered with PRINT # statements, the output file should be closed with the CLOSE statement. Since EduSystem 25 allows only one output file to be opened at a time, the CLOSE statement is used to close the output file. For example:

```
30 CLOSE
```

READING DATA FILES

When the user wishes to use data stored on DECtape data files, he opens a previously created file with the OPEN FOR INPUT statement. If, for example, the user wishes to read data stored in a file named DATA10, he can reference this name in the OPEN statement or use a string variable to represent it. The data file would be opened when the following statement was executed in a program:

```
40 OPEN "DATA10" FOR INPUT
```

The data file name must be enclosed in quotation marks. The above statement could also be coded as:

```
35 B$="DATA10"
40 OPEN B$ FOR INPUT
```

When the data file has been opened for input, the user may read data from the file with the INPUT # statement. The INPUT # statement searches the file for a carriage return/line feed character and inputs the next data item, or items. This statement may contain either alphabetic or string variables separated by commas or semicolons. For example, the statement:

```
50 INPUT #,A,B,C,D,E
```

searches for a carriage return/line feed character and inputs the next five separate data items (separated by either a comma, a semicolon, or a carriage return/line feed) from a data file.

The following program uses the examples in this section to demonstrate data file usage:

```
10 X$="DATA10"
15 OPEN X$ FOR OUTPUT,12
20 PRINT#,1,2,3,4,5
25 A$="JONES"\B$="SMITH"\C$="TAYLOR"
30 PRINT#,A$,B$,C$
35 CLOSE
40 OPEN X$ FOR INPUT
50 INPUT#,A,B,C,D,E
55 INPUT#,A$,B$,C$
60 PRINT
65 PRINT A,B,C,D,E
70 PRINT
75 PRINT A$,B$,C$

RUN
1             2             3
JONES         SMITH         TAYLOR
```

LISTING DATA FILES

The user can obtain a listing of the data files for his terminal by typing the FILELOG command. This immediate mode command is typed as:

```
FILELOG
```

or

```
FIL
```

and produces a listing of the data file names without extensions. For example:

```
FILELOG
DATA10
NUMBER
CIRCLE
TIMER

READY
```

ERASING DATA FILES

Data files can be deleted from DECtape with the KILL command. If, for example, the user wishes to erase a file named DATA10, he types:

```
KILL DATA 10
```

when the RETURN key is pressed, BASIC deletes DATA10 from the data file and prints READY. The KILL command and the file name must be separated by a space. This command deletes only those data files entered from the same terminal, i.e., with the same extension.

USING PUBLIC DATA FILES

Data files stored by the System Manager with the extension .D8 may be accessed by all users. To obtain a listing of the available public data files, the user types:

```
FILELOG$
FIL$
```

and presses the RETURN key. A listing of the public data files is printed. A sample listing is shown below:

```
FIL$
TEST01
TEST02
DATA50
DATA55
DATA60
```

The user can use the data in any public data file by opening the file for input. For example, to use the data in a public file named DATA55, the user types:

```
90 OPEN$ "DATA55" FOR INPUT
```

Included in a program, the above statement would make the data in DATA55 available to the user program. The user can manipulate this data in his program but cannot change the data file in any way. Public data files are protected by the .D8 extension.

Table 6-1. EduSystem 25 BASIC Statement Summary

Command (Abbreviation)	Explanation
Input/Output Statements	
LET	Assign a value to a variable. The word LET is optional.
READ(REA)	Assign values from data list to variables.
PRINT(PRI)	Print out the indicated information on the Teletype.
DATA(DAT)	Provide data for a program.
RESTORE(RES)	Restore the data list.
INPUT(INP)	Get values or expressions from the Teletype.
LINPUT(LIN)	Get long character string from the Teletype.
GOTO(GOT)	Change order or program execution.
IF THEN(IF THE)	Conditionally perform specified operation or conditionally change order of program execution.
FOR TO STEP(FOR TO STE)	Set up a program loop.
NEXT(NEX)	End of program loop.
GOSUB(GOS)	Go to a subroutine.
RETURN(RET)	Return from a subroutine.
ON GOTO(ON GOT)	Conditionally change order of program execution according to evaluation of formula contained in statement.
ON GOSUB(ON GOS)	Conditionally go to a subroutine according to evaluation of formula contained in statement.
REMARK(REM or ')	Insert a program comment.
RANDOMIZE(RAN or RANDOM)	Cause RND function to randomly select new sequence of random numbers between 0 and 1.
DEF FN	Define a function.
STOP(STO)	Stop program execution.
END	End a program (not required with EduSystem 25).
Editing/Control Commands	
LIST(LIS)	List entire program in memory.
LIST n(LIS n)	List line n.

Table 6-1 (Cont.). EduSystem 25 BASIC Statement Summary

Command (Abbreviation)	Explanation
LIST n,m(LIS n,m)	List lines n through m inclusive.
DELETE n(DEL n)	Delete line n.
DELETE n,m(DEL n,m)	Delete lines n through m inclusive.
EDIT n(EDI n)	Search line n for the character typed.
KEY	Return to KEY (normal) mode.
RUN	Execute the current program.
SCRATCH(SCR)	Erase the current program from memory.
BYE	Same as SCRATCH.
TAPE(TAP)	Read a program from the Teletype paper tape reader or punch a program on the Teletype paper tape punch.
CTRL/C	Stop a running program, print STOP, and then READY.
Program Storage/Retrieval Commands	
NEW	Clear memory, request program name.
RENAME(REN)	Change the name of the program currently in core memory.
SAVE	Store program in memory on DECtape, using the name specified by NEW, RENAME, or OLD and an extension determined by the user's terminal.
CHAIN A$	Erase memory, retrieve, load and begin execution of the stored program named in A$. Used to segment a large program into workable sections.
CHAIN$ A$	Chain to public library programs.
OLD	Clear memory, request program name, bring named program to memory from DECtape.
UNSAVE	Delete from DECtape the program named in most recent

Table 6-1 (Cont.). EduSystem 25 BASIC Statement Summary

Command (Abbreviation)	Explanation
	NEW, RENAME, or OLD command.
'CATALOG(CAT)	List names of stored programs for this user (extensions are not printed).
'CAT$	List public library programs.
'OLD$	Request public library program name, bring named program to memory from DECtape.
Data File Commands	
OPEN A$ FOR OUTPUT,X	Create a DECtape file named by A$. If the file already exists on DECtape, this command reopens it. X determines the number of blocks reserved for this file. It may be any integer up to 64. If no X is entered, 10 is assumed.
OPEN B$ FOR INPUT	Open the existing file named by B$. If no such file exists, an error message is printed.
OPEN$ B$ FOR INPUT	Open a .E0 file named by A$.
INPUT #	Read variables (numeric and/or string) from the input file previously opened.
PRINT #	Write data (numeric and/or string) onto the output file previously opened.
CLOSE	Close open output file.
FILELOG(FIL)	Catalog the data files stored by this user (used only in immediate mode).
FILELOG$(FIL$)	Catalog the data files stored by the System Manager with the extension ,D0.
KILL A$	Delete a DECtape data file named by A$.

Table 6-2. EduSystem 25 BASIC Function Summary

Function	Description
ABS(X)	Absolute value of x.
ATN(X)	Arctangent of x(result in radians).
COS(X)	Cosine of x(x in radians).
EXP(X)	e^x (e=2.718282).
INT(X)	Greatest integer of x.
LOG(X)	Natural logarithm of x.
RND(X)	Random number.
SGN(X)	Sign of x (+1 if positive, −1 if negative, 0 if zero).
SIN(X)	Sine of x (x in radians).
SQR(X)	Square root of x.
TAN(X)	Tangent of x (x in radians).
TAB(X)	Controls printing head position on Teletype.
FIX(X)	Truncates decimal portion of x.
CHR$(X)	Converts character code to character. Used only with the PRINT command.
MID(A$,M,N)	Returns N characters, starting at the Mth character of A$.
LEN(A$)	Returns the number of characters in A$.
CAT(A$,B$)	Returns a string of A$ concatenated with B$ (maximum of 6 characters returned).

ERROR MESSAGES

EduSystem 25 checks all statements and commands before executing them. If a statement or command cannot be executed, the system informs the user by printing an error message and the line number (if available) in which the error was found (see Table 6-3). Procedures used to correct errors are described under Program Editing.

Table 6-3. EduSystem 25 Error Messages

Message	Explanation
WHAT?	Immediate mode statement or command not understood. It does not begin with a line number and is not a valid system command.
ERROR 1	Log of negative or zero number was requested.
ERROR 2	Square root of negative number was requested.
ERROR 3	Division by zero was requested.
ERROR 4	Overflow, exponent greater than approximately +38.
ERROR 5	Underflow, exponent less than approximately −38.
ERROR 6	Line too long or program too big.
ERROR 7	Characters are being typed too fast; use TAPE command for reading paper tapes.

Table 6-3 (Cont.). EduSystem 25 Error Messages

Message	Explanation
ERROR 8	System overload caused character to be lost.
ERROR 9	Program too complex or too many variables. (GOSUB, FOR, or user defined function calls are too deeply nested.)
ERROR 10	Missing or illegal operand or double operators.
ERROR 11	Missing operator before a left parenthesis.
ERROR 12	Missing or illegal number.
ERROR 13	Too many digits in number.
ERROR 14	No DEF for function call.
ERROR 15	Missing or mismatched parentheses or illegal dummy variable in DEF.
ERROR 16	Wrong number of arguments in DEF call.
ERROR 17	Illegal character in DEF expression.
ERROR 18	Missing or illegal variable.
ERROR 19	Single and double subscripted variables with the same name.
ERROR 20	Subscript out of range.
ERROR 21	No left parenthesis in function.
ERROR 22	Illegal user defined function, not FN followed by a letter and a left parenthesis.
ERROR 23	Mismatched parentheses or missing operator after right parenthesis.
ERROR 24	Syntax error in GOTO.
ERROR 25	Syntax error in RESTORE.
ERROR 26	Syntax error in GOSUB.
ERROR 27	Syntax error in ON.
ERROR 28	Index error out of range in ON.
ERROR 29	Syntax error in RETURN.
ERROR 30	RETURN without GOSUB.
ERROR 31	Missing left parenthesis in TAB function.
ERROR 32	Syntax error in PRINT.
ERROR 33	Writing past end-of-file.
ERROR 34	Missing or illegal line number.
ERROR 35	Attempt to GOTO or GOSUB to a nonexistent line.
ERROR 36	Missing or illegal relation in IF.
ERROR 37	Syntax error in IF.
ERROR 38	Missing equal sign or improper variable left of the equal sign in LET or FOR.
ERROR 39	Subscripted index in FOR.
ERROR 40	Syntax error in FOR.
ERROR 41	FOR without NEXT.
ERROR 42	Syntax error in LET.

Table 6-3 (Cont.). EduSystem 25 Error Messages

Message	Explanation
ERROR 43	Syntax error in NEXT.
ERROR 44	NEXT without FOR.
ERROR 45	Too much data typed or illegal character in DATA or the data typed in.
ERROR 46	Illegal character or function in INPUT or READ.
ERROR 47	Out of data.
ERROR 48	Unrecognized command: RUN mode.
ERROR 49	Bad file name or file not found.
ERROR 50	Syntax error in LINPUT.
ERROR 51	String error, argument is incorrect type or out of bounds.
ERROR 52	String function error, missing or illegal argument.
ERROR 53	File not open or bad syntax after #.
ERROR 54	Reading past end of file.
ERROR 55	Bad syntax in OPEN.
ERROR 56	No room for file on DECtape.

PROGRAM EDITING

There are two times when a program may require editing procedures. The first occurs while a line is being typed but before the RETURN key is pressed. The second occurs when a line has been completely typed and the RETURN key has been pressed. Each situation has its own editing procedures.

Situation 1: Before the RETURN key is pressed.

Three keys may be used to correct typing errors: ← (SHIFT/0), RUBOUT, or ALT MODE (or ESC).

← (back arrow), SHIFT/0 on the keyboard, or RUBOUT is used to delete a character from a line. BASIC prints the back arrow, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

ALT MODE (ESC on some Teletypes) is used to delete an entire line. When this key is used, BASIC prints $DELETED, erases that line from the program, and returns the carriage so that the line may be retyped.

Situation 2: After the RETURN key is pressed.

Once a line of the program has been transmitted to computer memory via the RETURN key, several methods of correction may be used. Lines may be inserted, deleted, or changed.

INSERTION

To add a line to a program, assign a line number that falls between two existing lines, type the line number and text, and press RETURN.

DELETION

To erase a line from computer memory, type the line number *only* and press the RETURN key. The DELETE command may also be used to erase lines from memory. To erase a single line, type DELETE and the line number and press RETURN. DELETE followed by the two line numbers separated by a comma erases all lines between and including the two given. For example:

```
DELETE 10,20
```

erases lines 10 through 20.

CHANGE

Old instructions may be replaced by new ones by retyping the line. This procedure is adequate for changing simple lines. When, however, the line contains a complex formula or a long message to be printed, it may be changed with the EDIT command. The EDIT command allows the user to access a single line and search for the character or characters to be changed. To use this command, type

EDIT *line number*

and press the RETURN key. BASIC waits for a search character to be typed (BASIC does not print this search character when it is typed). This search character is one that already exists on the line to be changed. After the search character is typed, BASIC prints the contents of that line until the search character is printed. At this point, printing stops and the user has the following options:

- Type new characters; BASIC inserts them following the ones already printed.
- Type a form feed (CTRL/L) to cause the search to proceed to the next occurrence, if any, of the search character.

- Type a BELL (CTRL/G) to signal a change of search character, then type a new search character.
- Use the RUBOUT or ← key to delete one character to the left each time the key is pressed.
- Type the RETURN key to terminate editing of the line at that point, deleting any existing text to the right.
- Type the ALT MODE key to delete all existing characters to the left except the line number.
- Type the LINE FEED key to terminate editing of the line, saving all the remaining characters on that line.

When the EDIT operation is complete, BASIC prints READY. Note that line numbers cannot be changed with the EDIT command. The following example demonstrates the EDIT command. An incorrect line was typed and entered to the system as follows:

```
60 PRINT "PI=3.14146 ABOU%!"
```

The line was edited as follows:

```
EDIT 60
PRINT "PI=3.14146←←59 ABOU%←T!"
```

First 6 was entered as the search character. BASIC printed the line to the 6, and the RUBOUT key was typed twice to remove the two incorrect digits (46) and 59 was inserted in the line. CTRL/G was typed and another search character (%) was entered. BASIC printed characters to the % which was removed with a RUBOUT and replaced with a T. The LINE FEED key was typed to terminate the edit and save the remaining characters. If the line is listed, the following is printed on the Teletype.

```
LIST 60

  60 PRINT "PI=3.14159 ABOUT!"

READY
```

OPERATING INSTRUCTIONS

Loading EduSystem 25

EduSystem 25 software is supplied on a DECtape. This tape, called the system DECtape, must be used to create the system when the system is installed. Perform the following procedures to activate EduSystem 25.

INITIALIZE THE DECTAPE UNIT

Perform the following steps to prepare the DECtape unit for loading software:

1. Set the WRITE ENABLE/WRITE LOCK switch to WRITE LOCK.
2. Set the REMOTE/OFF/LOCAL switch to OFF.
3. Place the system DECtape on the left spindle with the DECtape label out.
4. Wind four turns of tape onto the right spindle.
5. Set the REMOTE/OFF/LOCAL switch to LOCAL.
6. Wind a few turns of tape onto the right spindle with the → switch to ensure that the tape is properly mounted.
7. Dial 0 on the unit selector dial.
8. Set the REMOTE/OFF/LOCAL switch to REMOTE.

INITIALIZE COMPUTER MEMORY

The system may be activated in one of two ways: with an automatic loader (hardware bootstrap) or with an OS/8 bootstrap loader. The following instructions are for an EduSystem 25 that includes a hardware bootstrap. If the EduSystem does not have a hardware bootstrap, see *Introduction to Programming 1972*, Chapter 9, for OS/8 bootstrap loading instructions.

1. Turn the key lock on the computer console to POWER.
2. Turn all Teletypes to LINE.
3. Mount the EduSystem 25 system DECtape on drive 0 as described above.
4. Set the SWITCH REGISTER (SR) to 0600 (octal).[4]
5. Press and raise the HALT switch.
6. Press and raise the SW switch.

The EduSystem-25 DECtape spins and the system indicates that it is active by printing a period (.) on the console Teletype. If the

[4] An explanation of the octal, or base 8, number system is included in *Introduction to Programming 1972*.

system is not activated, ensure that the correct DECtape is being used and repeat the above procedures.

Immediately following the period printed by the system, the user enters the characters shown below:

```
.R EDU25
```

The period is printed by the OS/8 Keyboard Monitor. The user types R to request the running of a program, in this case, EduSystem 25.[5] When the RETURN key is pressed, the system prints:

```
TO BOOTSTRAP BACK OS/8 MONITOR:
 LOAD ADDRESS 07600
 AND START
```

This message means that EduSystem 25 (or other OS/8 programs) can be started by setting the SR to 7600 and pressing ADDR LOAD, CLEAR, and CONT. The OS/8 Monitor prints another period, to allow the user to request a program. If, for example, the user types an incorrect response to the ensuing system dialog, he can restart the dialog by reloading (bootstrapping back) the OS/8 Monitor from location 7600 and again requesting that EduSystem 25 be run.

ANSWER SYSTEM DIALOG

When the system has been activated correctly, it prints the identification message:

```
EDUSYSTEM 25    BASIC
```

and begins to ask certain questions which the user must answer to establish the system configuration. The first question is:

```
NUMBER OF USERS (1 TO 5)?
```

The user responds with a single digit from 1 to 5, depending on the number of terminals to be used. If one user is specified, this ques-

---

[5] Refer to *Introduction to Programming 1972*, Chapter 9, for a complete explanation of the OS/8 Operating System.

tion ends the initial BASIC dialog. If more than one user is indicated, BASIC continues the dialog by asking:

```
PDP-8/L COMPUTER (Y OR N)?
```

The user responds Y if the EduSystem 25 computer is a PDP-8/L, N if not. An N response to this question prompts the next question:

```
STANDARD REMOTE TELETYPE CODES (Y OR N)?
```

BASIC is asking for a PT08 or KL8E device code for each Teletype to be used (excluding the console Teletype). Standard PT08 or KL8E device codes are 40, 42, 44, 46, 50, 52, and 54. When a system using PT08 or KL8E interface units is first installed, the user determines the specific device code for each Teletype and labels each Teletype with its specific device code. If device codes are standard, the user responds Y to this question and BASIC assumes the standard device codes and continues the dialog. If device codes are not standard, the user enters N; BASIC then asks:

```
TELETYPE #1 DEVICE CODE?
```

BASIC asks this question for each Teletype to be used, up to seven times for an 8-user system. The user responds with the specific 2-digit device code for each Teletype.

When the device codes have been determined, or if the computer is a PDP-8/L, BASIC asks the following question:

```
SAME AMOUNT OF STORAGE FOR ALL USERS?
```

The above question requires the user to decide whether to partition the available core equally among the users on the EduSystem 25. (Since EduSystem 25 software uses 4K of core, the available core is always 4K less than the total core on the system.) If the user responds Y to this question, BASIC determines the size of the core memory available and divides it equally among the users, then ends the dialog. If N is the response, BASIC determines the amount of available core storage and prints the highest core field according to the following:

Field 7—32K core memory
Field 6—28K core memory
Field 5—24K core memory
Field 4—20K core memory
Field 3—16K core memory
Field 2—12K core memory
Field 1— 8K core memory

For explanation purposes, the following dialog is written for a 16K, 5-user EduSystem 25. The available core is to be allocated as follows:

User 1—10 blocks (user 1 is the console terminal)
User 2— 6 blocks
User 3— 8 blocks
User 4— 4 blocks
User 5— 4 blocks

Each core field contains 16 blocks; a core field may be divided among several users, but no user may be allotted blocks in more than one core field. To determine the number of blocks, BASIC prints the following dialog and the user answers as shown:

```
FIELD 3
THERE ARE 16 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #1 WILL BE HOW MANY BLOCKS?10
THERE ARE 06 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #2 WILL BE HOW MANY BLOCKS?6
FIELD 2
THERE ARE 16 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #3 WILL BE HOW MANY BLOCKS?8
THERE ARE 08 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #4 WILL BE HOW MANY BLOCKS?4
THERE ARE 04 BLOCKS LEFT IN THIS FIELD.
 YOUR ALLOCATION FOR USER #5 WILL BE HOW MANY BLOCKS?4
```

When an invalid response is made to any of BASIC's questions, an error message is printed and the question is repeated. For example:

```
STANDARD REMOTE TELETYPE CODES (Y OR N)?4
INVALID RESPONSE

STANDARD REMOTE TELETYPE CODES (Y OR N)?
```

When all responses have been entered, BASIC prints:

```
IS THE ABOVE CORRECT (Y OR N)?
```

If an incorrect response was made, answer N and BASIC begins the dialog again. A response of Y ends the dialog and BASIC prints:

```
END OF DIALOGUE

READY
```

BASIC prints READY on each of the Teletypes associated with the specified device codes.

ESTABLISH TERMINAL EXTENSIONS

When EduSystem 25 is active, the user can write a simple, uniquely named program for each terminal (including the console terminal) and save the program from the terminal. To do this, the user:

1. Sets DECtape drive 0 to WRITE ENABLE.
2. Types the following programs, for example, on the terminals in the sequence given:

First Terminal

```
NEW
NAME--ALPHA

READY

10 PRINT "TERMINAL ALPHA"
20 END
SAVE

READY
```

Second Terminal

```
NEW
NAME--BRAVO

READY

10 PRINT "TERMINAL BRAVO"
20 END
SAVE

READY
```

Third Terminal

```
NEW
NAME--CLOVER

READY

10 PRINT "TERMINAL CLOVER"
20 END
SAVE

READY
```

The user repeats this process, using a unique name, for each remaining terminal.

When the user has saved a uniquely named program from each terminal, he performs the following actions:

1. Set DECtape drive 0 to WRITE LOCK.
2. Press and raise the HALT switch.
3. Set the SR to 7600; press ADDR LOAD switch.
4. Press the CLEAR switch, then the CONT switch.

The OS/8 Monitor responds by printing a period, and the user types:

```
.R PIP
```

PIP is the OS/8 Peripheral Interchange Program which is used to transfer files between devices, merge and delete files, and list, zero, and compress directories.[6] In this example, it is being used to obtain a directory listing. After the user types PIP and presses RETURN, PIP responds with an asterisk and the user types:

```
*/F
```

to request a directory listing. This example would produce a listing similar to the following:

```
EDU25 .SV
PIP   .SV
ALPHA .E1
BRAVO .E2
CLOVER.E3
```

[6] Refer to *Introduction to Programming 1972,* Chapter 9, for a complete explanation of OS/8 PIP.

The EDU25 .SV and PIP .SV files are executable core images of these two system software programs. ALPHA .E1, BRAVO .E2, and CLOVER .E3 are the programs the user wrote and saved on DECtape. Now that the user knows the correct extensions for each Teletype, he can use a marker, e.g., a permanent-ink felt tip marker, to write the appropriate extension on each Teletype. The Teletype extensions will thus be clearly identified for each user. To return to the OS/8 Monitor, the user types CTRL/C and the Monitor again prints a period.

CREATE DATA FILE TAPE

DECtape drive 1 is used for EduSystem 25 data files. When the system is first installed, the user should create a tape for data file storage in the following manner:

1. Mount a blank DECtape on drive 1; set the drive to WRITE ENABLE.
2. Type R PIP after the OS/8 Monitor period. When PIP responds with an asterisk, type the following:

```
*DTA1:</Z
```

   This message instructs PIP to create a zeroed OS/8 directory on the DECtape. The user should create this zeroed directory on each data DECtape file he establishes.
3. Press CTRL/C; the OS/8 Monitor responds with its period and EduSystem 25 can be restarted by typing:

```
.R EDU25
```

   pressing RETURN, and answering the dialog as before.

When the data file tape has been created, EduSystem 25 is ready for use. Turn the key lock to PANEL LOCK to prevent the system from being disturbed.

Maintaining the Public Library

As previously explained, EduSystem 25 has a public library of programs which all users can access. Public library programs are stored or deleted by the system manager under the .E8 extension. The system manager can access all stored files and insert programs

into the library using PIP. For example, assume that a user on terminal .E2 has developed a program that the system manager feels all users would like to access. This program, named PEACE, could be inserted into the public library by the system manager as follows:

1. Turn key lock to POWER.
2. Set DECtape drive 0 to WRITE LOCK.
3. Press and raise the HALT switch.
4. Set the SR to 0600.
5. Press and raise the SW switch.
6. The OS/8 Monitor prints a period on the console terminal (.EØ). The user types:

```
.R PIP
```

7. Set DECtape drive 0 to WRITE ENABLE.
8. When PIP responds with an asterisk, type :

```
*/F
```

PIP prints the directory listing of the EduSystem 25 system DECtape, for example:

```
EDU25 .SV
PIP   .SV
ALPHA .E1
TEST  .EØ
GAMBLE.E1
ROCKET.EØ
BRAVO .E2
PEACE .E2
```

9. Stop the directory printout by typing CTRL/O (press O while holding down the CTRL key). PIP responds with an asterisk. The user then types:

```
*DTAØ:PEACE.E8<DTAØ:PEACE.E2
```

and presses RETURN.

10. PIP copies the program PEACE and affixes the .E8 extension to the copy. PEACE has been placed in the public library.
11. Type CTRL/C. The OS/8 Monitor responds by typing a period. EduSystem 25 can be restarted.
12. Turn the key lock to PANEL LOCK.

Protecting DECtape Files

Programs, or data files, stored under the system manager's extension (.E8 or .D8) are protected by simply locking the computer console. This information can still be accessed by all users but cannot be erased from the DECtape or manipulated in any way. Program and data files created under user extensions can be manipulated and deleted as previously described. The system manager can protect user extensions, limiting them to read-only operations by setting the appropriate DECtape unit (drive 0 to 1) to WRITE LOCK. DECtape drives should also be set to WRITE LOCK when:

1. Starting up any DECtape system.
2. Shutting down any DECtape system.

Storing Programs on Paper Tape

If ASR-33 Teletypes (TTY) are available, programs may be saved on paper tape to save storage space on the system DECtape. Once a program has been typed correctly, it may be saved by performing the following procedures:

1. Turn the TTY control knob to LINE.
2. Type TAPE; press RETURN.
3. Turn the TTY paper tape punch ON.
4. Type LIST; press RETURN.
5. When punching is complete, turn the TTY punch OFF.
6. Type KEY; press RETURN.

Reloading Programs from Paper Tape

Programs saved on paper tape may be reloaded using the Teletype (TTY) paper tape reader. To reload programs, perform the following procedures.

1. Turn the TTY paper tape reader to FREE.
2. Turn the TTY control knob to LINE.

3. Insert tape in the reader.
4. Type TAPE; press RETURN.
5. Turn the TTY reader to START.
6. When the tape has read in, turn the TTY reader to FREE.
7. Type KEY; press RETURN.

System Reconfiguration

If the user desires to change the EduSystem 25 configuration at any time, he simply performs the procedures used for loading the system initially and answers the dialog again to reflect the new configuration. A reconfiguration would be needed, for example, if new terminals were added, thus changing the number of users. EduSystem 25 must be inactive to be reconfigured. To ensure that the system is inactive, the user types CTRL/C to stop a running program or the listing of a program and turns all the terminals OFF. He can then proceed with the reconfiguration.

System Shutdown

If power failure detection hardware is available on the system, the system can be shut down by simply turning the console key lock to OFF. Otherwise, to shut the system down, overnight or for any reason, the user ensures that the system is inactive, as explained above; he then presses the HALT switch and turns the key lock to OFF.

System Restart

If power failure detection hardware is available on the system, it can be restarted by simply setting all DECtape drives to WRITE LOCK and turning the key lock to PANEL LOCK. Otherwise, the following procedures are necessary:

1. Turn the key lock to POWER.
2. Set the SR to 7600 and press ADDR LOAD.
3. Press the CLEAR switch and the CONT switch.
4. Turn the appropriate Teletypes to LINE.
5. Turn the key lock to PANEL LOCK.

EduSystem 25 responds by printing the OS/8 Monitor period. The user can then request EduSystem 25 as described previously.

chapter 7

edusystem 30

INTRODUCTION

EduSystem 30 is a powerful BASIC-speaking system that adapts to the loner or to the crowd. The system operates in two modes: batch and interactive. In batch mode, the system reads programs prepared on specially formatted optical mark cards. Cards are marked with an ordinary pencil or keypunched. Running programs in batches, EduSystem 30 can process hundreds of student programs per day. When operated in interactive mode, EduSystem 30 allows one person at a time to interact with the computer through the Teletype.

In either mode, the system provides a powerful BASIC language with advanced features such as string variables, program chaining, and data files. Programs can be stored on a magnetic storage device (DECdisk or DECtape) and retrieved when needed —a great time-saving feature. In addition, EduSystem 30 keeps a log of all programs it runs, providing an exact record of how students are using the computer. Also available is a unique capability for automatic testing (and grading) of student programs.

System Components

EduSystem 30 is composed of a PDP-8/E computer with 4096 (4K) words of core memory, powerfail protection, automatic loader (hardware bootstrap), 32,768 (32K) word DECdisk, optical mark card reader, and computer terminal (Teletype) with paper tape reader/punch. Each EduSystem 30 includes the BASIC language processor with batch capabilities and a library of sample programs, textbooks, and curriculum guides. DECtape may be substituted for DECdisk. A punched card reader may be used in place of the marked card reader. The system also supports a high-speed paper tape reader/punch and a line printer.

System Expansion

EduSystem 30 expands easily to EduSystem 40 and, with additional memory and DECtape, may be used to run the OS/8 Operating System, DEC's programming system for the PDP-8 computer.

- To expand to EduSystem 40, add 4K words of core memory, plus up to seven additional computer terminals and their associated interfaces (12K words of core memory are recommended for more than five terminals), and an EduSystem 20 software set.
- To run OS/8, add 4K words of core memory (providing the system includes DECtape).

BASIC LANGUAGE CAPABILITIES

EduSystem 30 BASIC includes the language elements shown in Table 7-1. The system also includes many advanced features to allow the user to perform more complicated and lengthy problem-solving operations. Features which pertain to either the interactive mode or the batch mode are discussed in separate sections. This section explains the advanced features that can be used in both modes.

Table 7-1. EduSystem 30 BASIC Statement Summary

Statement	Description
Input/Output Statements	
LET	Assign a value to a variable. The word LET is optional for interactive programs.
PRINT	Print out the indicated information.
READ	Assign values from data list to variables.
DATA	Provide data for a program.
RESTORE	Restore the data list.
WRITE	Record data on DECtape storage file.
GOTO	Change order of program execution.
IF GOTO IF THEN	Conditionally change order of program execution.
FOR TO STEP	Set up a program loop.
NEXT	End of program loop.
DIM	Define subscripted variables.
GOSUB	Go to a subroutine.
RETURN	Return from a subroutine.
INPUT	Get values from the Teletype.
REMARK (REM)	Insert a program comment.

Table 7-1 (Cont.). EduSystem 30 BASIC Statement Summary

Statement	Description
RANDOMIZE	Cause RND function to randomly select new sequence of random numbers between 0 and 1.
DEF	Define a function.
CHAIN	Link to next section of a program which is stored within the system.
NOLINE	Do not print out the line numbers in which program logic errors are found. (Allow longer-than normal programs to be run without chaining.)
STOP	Stop program execution.
END	End a program.

Editing/Control Commands

LIST	List all stored program statements.
LIST n	List program starting at line n.
LISTNH	List all program statements but do not print a header line.
LISTNHn	List program starting at line n but do not print a header line.
RUN	Execute the current program.
RUN NH	Same as RUN without header line.
SCRATCH (SCR)	Delete the currently stored (in memory) program.
CTRL/C	Stop execution of a program or printing of a listing. CTRL/C is typed by pressing C while holding down the CTRL key.
TAPE	Read a program from paper tape. Ignore any line which does not begin with a line number.
ECHO	Switch from printout to non-printout mode or vice versa.
PUNCH	Punch a program on the high-speed punch.
PUNCH n	Punch a program on high-speed punch, starting at line n.
LPT	Print output on line printer.
TTY	Switch back to Teletype from line printer.
BATCH	Begin processing card programs.
RESEQUENCE	Renumber program lines.
NEW	Clear memory, request program name.
OLD	Clear memory, bring program to memory from storage area.
NAME	Same as new but does not clear memory.

Table 7-1 (Cont.). EduSystem 30 BASIC Statement Summary

Statement	Description
CATALOG (or CAT)	Print out the names of programs in storage area.
LENGTH	Print out the number of blocks needed to store the current program.
PRIVILEGE	Enable use of privileged commands. To be successful, this command must be followed by the correct password. This command is recognized only if the privileged command capability was selected at system lead time.
Privileged Commands[1]	
PASSWORD	Change the password code.
SAVE	Save the current program in the system storage area.
UNSAVE	Delete the named program from the system storage area.
HEADER	Change the system header; type new header (maximum 12 characters) for next batch run.
LOG	Print system log.
MAX	Set instruction limit n times 200 per program for next batch run.
BATCH n	Same as BATCH; limit runs (n) per program.
STACK	Start unattended batch operation.
STACK n	Same as STACK; limit runs (n) per program.
Functions	
ABS(X)	Absolute value of x.
ATN(X)	Arctangent of x (result in radians).
COS(X)	Cosine of x (x in radians).
EXP(X)	e^x (e=2.712818).
INT(X)	Greatest integer of x.
LOG(X)	Natural logarithm of x.
RND(X)	Random number.
SGN(X)	Sign of x (+1 if positive, −1 if negative, 0 if zero).
SIN(X)	Sine of x (x in radians).
SQR(X)	Square root of x.
TAN(X)	Tangent of x (x in radians).
TAB(X)	Controls printing head position on Teletype.
CHR$(X)	Converts character code to character. Used only with the PRINT command.

[1] *The privileged commands may only be used after a successful PRIVILEGE command has been executed.*

Using Random Numbers

The RND function allows the use of random numbers within a program. The RND function returns a random value between zero and one. Unlike the other functions, the value returned by RND is not a function of its argument. However, all functions in BASIC must be followed by an argument. Therefore, RND should always be followed by a dummy argument, such as zero, which is enclosed in parentheses.

Notice that it is possible to generate random numbers over any range. For example, the expression:

$$(B-A)*RND(0)+A$$

has a random value in the range A<n>B.

Repeated uses of RND in a program return different values between zero and one. The sequence of numbers is, however, the same each time the program is run. Thus, the sequence is reproducible for later checking of the program. The RANDOMIZE statement allows the user to make the random number sequence returned by the RND function different each time a program is run. That is, when executed, the RANDOMIZE statement causes the RND function to select randomly a new sequence of random numbers. If RANDOMIZE is used, it normally appears as one of the first lines in a program.

Running Long Programs

EduSystem 30 runs programs of up to 6000 characters in interactive mode and up to 5000 characters in batch mode. These limits correspond to roughly 200 and 250 lines per program, respectively. In some cases, interactive programs which are at or near the 6000 character limit and which contain many complex FOR, IF, and GOSUB sections are too large to be run. If this is the case, the NOLINE command may be used to gain more space. If NOLINE is used, program logic errors which are detected while the program is executing cause an error message to be printed, but the line number of the error is not printed. NOLINE allows substantially longer programs to be run.

If the program to be run is substantially longer than the 6000-character limit, it may still be run by means of a technique known as chaining. The program is written in sections, each of which is

less than 6000 characters. A chained program may have many of these program sections and, hence, may be indefinitely long. Each section of the total program is then stored in the system storage area with the SAVE command under a separate name. The final command to be executed in all but the last section is a CHAIN statement containing the name of the next section of the program. For example, the statement

```
950 CHAIN "PART10"
```

would cause the system to load and execute the stored program whose name is PART10.

In the CHAIN statement, the name of the next section of the program must be encoded in quotation marks and must be exactly six characters long. If the actual name of the next section is less than six characters, one or more spaces must be inserted *before* the second quotation mark to make a total of six characters. For example, if the next section of the program is named LINK2, the following CHAIN statement would be used:

```
955 CHAIN "LINK2 "
```

Execution of the CHAIN statement loads and executes the named program. The previous section of the program is deleted. Thus, the user only needs to load the first section and run it. All succeeding sections of the chained program load and execute automatically.

Using a Data File

Just as some programs may be too large to be executed in one piece, other programs may need to store and use more data than may be accommodated under normal system operation. If this is the case, data may be temporarily stored in the system storage area. Data stored in this way is referred to as a data file.

The data file is actually a part of the program data which is defined by a DATA statement. All data within a program is gathered from the DATA statements into a data list which is then read by READ statements. As items are read from the list, they are marked as already having been used. A READ statement always fetches the *next* item from the list. In fact, the data list may be considered to have a movable marker which remembers the next item of the list. The marker initially marks the first data item. As READ statements are executed, the marker moves down the list. A RESTORE statement moves the marker back to the top of the list.

The data file capability allows a program, by means of a WRITE statement, to change and add to this data list as well as to read it. The WRITE statement format is the same as the DATA statement format. Writing a variable puts the value of that variable in the next place in the data list. The data item that was there previously is replaced by the new value. If a WRITE statement follows a RESTORE, it changes the first item or items in the data list. If it follows one or more READs (or WRITEs), the WRITE statement changes data items further down in the data list. The total number of items which may be put in the data list depends on the size of the program. Maximum sized BASIC programs may have up to 1000 items; small programs have room for 2000 items.

Programs which write data on the data list must keep track of how much data has been written and the order in which it was output. If data which has been written is to be subsequently read, a RESTORE statement must be executed to return the marker to the top of the data list. If data has been written off the end of the data list, the program must remember how many items the data list contains, and be careful not to try to READ more data items than are there. The normal BASIC check for end of data does not exist for a written data list. The program must also be sure that it does not write more data than the data list can contain (1000-2000 items). Writing too much data causes part of the user's BASIC program to be destroyed.

Data files are frequently used in conjunction with chaining since data written onto the data list by one program section may be read by the next section. The program section which writes the

data must execute a RESTORE just before the CHAIN statement. The next section, which reads the data list, must not have any DATA statements since this data would destroy the data items written by the previous section.

Character Variables and String Capability

Standard BASIC statements deal only with numbers, assuming all variables to be decimal values. However, BASIC is also capable of performing many useful operations on characters or words (strings of characters) instead of numbers. The character handling capabilities of BASIC depend on the representation of individual characters as numbers. Each character has its own numeric code (or character code), as indicated in Appendix B. When a character is input, by an INPUT statement, it is converted to a numeric code. All internal processing of that character uses this code. Since the code is a number, it can be used and manipulated with any BASIC statement. When a user program is to output a character, BASIC converts the numeric code back into a character and prints the character. In short, characters stored in a BASIC program are indistinguishable from numeric values. The only difference is in the way they are used, i.e., that certain numeric values actually represent characters.

The INPUT statement is used to input characters. A dollar sign ($) is placed before a variable name to indicate that a character code is to be input rather than a decimal number, e.g., INPUT $A. When the character is typed, its character code is stored in the indicated variable. It is important not to confuse character input with numeric input. A potential confusion lies in the fact that the numeric values are themselves characters. For example, the value 234 is composed of the three characters 2, 3, and 4. If these three characters were input to a BASIC program as character variables, they would be entered as three separate numeric (character code) values rather than as the single value 234. Whether the input is character code or numeric, the physical characters typed at the terminal are identical; the difference is entirely in the way that the program interprets the input. In the following examples, each program executes an INPUT statement. In the program on the left, three characters are entered and three variables are set up. In the example on the right, a single numeric value is input.

```
10 PRINT "ENTER VALUE ";
20 INPUT $X1,$X2,$X3
30 PRINT X1;X2;X3
40 END
RUN NH

ENTER VALUE 234 50  51  52

READY
```

```
10 PRINT "ENTER VALUE";
20 INPUT X
30 PRINT X
40 END
RUN NH

ENTER VALUE?234
 234

READY
```

Note that character INPUT statements do *not* cause a question mark to be printed. Therefore, a series of characters may be typed without intervening question marks. Programs doing character input must therefore indicate, by PRINT statements, when input is expected. Also note that the $ is not part of the variable name. It is used only in INPUT statements to indicate that typed characters are to be converted to their numeric character codes before being stored in the variable.

Character codes may be converted to their respective characters by means of the special PRINT statement function CHR$. CHR$ is the inverse of the dollar sign INPUT convention. The CHR$ function takes, as its argument, a single constant or variable and prints the single character whose code corresponds to that value. For example:

```
50 PRINT CHR$(65)
```

prints the character A. CHR$ may only be used in PRINT statements.

One of the most frequent uses of the character capability is to enter words or characters into BASIC programs in response to questions. For example, a program might ask the user if he wants to run the program again with a different set of input data. The user responds by typing Y if he wants to run again or by typing N if not. The program then compares the code of the character entered with the character code of Y to determine if they are equal. If so, it branches back to the beginning of the program. If not, the program executes the remaining statements. The following program illustrates the use of character variables in making a run-time decision.

```
10 PRINT
20 PRINT "WOULD YOU LIKE TO DO THIS AGAIN (Y OR N)?";
30 INPUT $A
40 IF A=#Y THEN 10
50 IF A<>#N THEN 90
60 PRINT
70 PRINT "O.K. IT'S YOUR CHOICE."
80 STOP
90 PRINT
100 PRINT "Y OR N?";
110 GOTO 30
120 END
RUN NH


WOULD YOU LIKE TO DO THIS AGAIN (Y OR N)?Y
WOULD YOU LIKE TO DO THIS AGAIN (Y OR N)?8
Y OR N?Z
Y OR N?N
O.K. IT'S YOUR CHOICE.

READY
```

The comparisons shown in the preceding program are facilitated by a special BASIC language feature. Pound sign (#) followed by a single character may be used to indicate the character code of the single character following the pound sign. In line 40 above, using #Y relieves the programmer of the need to remember or reference the actual character code for Y.

Often, the character capability is used to input a series (or string) of characters, such as a last name. The string may be any number of characters up to a full line (72 characters on a Teletype line). In this case the program must read each character and determine whether it is the carriage return character (character code 13) which indicates the end of the line. Subscripted variables are used to store such a series of characters. The following program illustrates character string input:

```
10 DIM A(72)
20 PRINT "TYPE YOUR NAME:";
30 FOR I=1 TO 72
40 INPUT $A(I)
50 IF A(I)=13 THEN 70
60 NEXT I
70 END
```

USING THE INTERACTIVE TERMINAL

When EduSystem 30 is not processing card programs, it may be used interactively from the Teletype. In this mode of operation, programs and commands are typed directly at the keyboard without being marked on cards first. Individual program lines may be changed, deleted, or added without having to read the whole program again. Listings and program output are produced at the terminal (or line printer) as always.

The system is ready for interactive use whenever it prints READY. If a batch run has just been completed and the system has typed MORE CARDS?, the user responds by typing N to indicate that no more card programs are to be run. The system then prints READY. The word READY always indicates that the system can accept commands from the Teletype.

Entering Programs

Programs are entered in the computer by the user typing at the Teletype keyboard. Each statement in the program must begin with a line number between 1 and 4095. In interactive mode, all lines of input are terminated when the RETURN key is typed. Statements need not be entered in order. BASIC automatically arranges program lines in their proper order.

EduSystem 30 BASIC expects each program to have an assigned name. At the beginning of each interactive session, the NEW command should be typed to clear any existing program in core and define the name of the new program to be entered.[2] To use the NEW command, the user types

```
NEW
```

and the system asks for:

```
NEW FILE NAME--
```

The user then types any name of 1 to 6 characters (the first of which must be a letter), followed by the RETURN key. BASIC assigns that name to the program to be entered. The user may

---

[2] If the user does not wish to assign a program name, he can delete any existing program by typing the SCRATCH command.

change the name of the program being entered at any time by typing the NAME command. BASIC again asks for NEW FILE NAME and assigns a new name to the program being entered. The NAME command does not delete the existing program.

Using Multiple Statements per Line

EduSystem 30 allows more than one statement to be typed on a single line. Statements after the first begin with a back slash character (\) which is SHIFT/L on the keyboard. A program is often more understandable when statements, such as a series of LET's, are grouped into a single line. For example, the program:

```
10 A=3\B=8\C=13
20 PRINT (A+B+C)*5\END
```

is the same as

```
10 LET A=3
20 LET B=8
30 LET C=13
40 PRINT (A+B+C)*5
50 END
```

and will produce the same result when the RUN command is typed:

```
RUN NH

 120

READY
```

This multiple-statement capability is helpful when writing long programs since statements require less storage in the computer when they are grouped as a single statement.

Listing the Program

The LIST command may be used to list all or part of the current program. LIST prints the program statements in their proper order, regardless of the order in which they were entered. The EduSystem 30 LIST command has four different forms, as shown below.

Command	Meaning
LIST	List the entire program. Precede it by a header line[3] giving the name of the program.
LIST n	List the program starting at the given line number (n). Precede it by a header line. The line number *must* be separated from LIST by two spaces.
LISTNH	List the entire program, but do not print a header line.
LISTNHn	List the program starting at the given line number (n), but do not print a header line.

NOTE

The programmer may stop a listing at any time by typing CTRL/C on the keyboard.

Executing the Program

The programmer may execute a program at any time by typing the RUN command. The existing program is inspected for errors; if none exist, it is executed. If an error is detected, an error message (see the section on Error Messages) is printed. In either case, at the end of the run, BASIC prints READY, indicating that the program may now be changed or rerun. There are two types of RUN commands: RUN and RUN NH. RUN executes the current program, preceding it by a header line. RUN NH (NH means no header) executes the current program but does not print a header line. RUN and NH must be separated by a single space. The user may terminate program execution at any time by typing CTRL/C.

Loading a Card Program for Interactive Use

Programs previously written on cards may be loaded for use in interactive mode. The program deck is preceded by a NEW card and followed by a KEY card. The user mounts this deck in the card reader in the normal way (see Executing Card Programs for instructions) and types the BATCH command. Cards are processed in normal batch fashion until the KEY card is read. The KEY card switches the system to interactive mode. The program is then available for execution or editing from the interactive terminal.

[3] A header line consists of the program name followed, on the same line, by the system name (EDU BASIC). If no program name was assigned, the system prints "*NONE* EDU BASIC".

After two RUN commands (or after some number of RUNs if the privileged BATCH command is used), the system automatically returns to processing any additional card program in the reader.

Storing Programs on Paper Tape

Once a program has been entered or read correctly, it may be saved on paper tape for quick reloading, as follows:

1. Turn the Teletype control knob to LINE.
2. Type LISTNH but do not press the RETURN key.
3. Turn the Teletype paper tape punch ON.
4. Press SHIFT/CTRL, type "PPPPPP" to produce some leader tape.
5. Press the RETURN key.
6. When punching is complete, press SHIFT/CTRL, type "PPPPPP" to produce some trailer tape.
7. Turn the Teletype punch OFF.
8. Carefully rip tape off punch. Notice that an arrow head is at the beginning of the program and an arrow tail at the end.

Reloading Programs From Paper Tape

Programs punched on paper tape may be reloaded using the Teletype (TTY) paper tape reader. The TAPE command is used to load programs from paper tape as follows:

1. Insert the paper tape in the TTY reader, with the arrow head facing out of the reader.
2. Turn the TTY control knob to LINE.
3. Type NEW, then press the RETURN key.
4. Type the program name.
5. Type TAPE, press the RETURN key.
6. Turn the TTY paper tape reader to START.
7. When the tape has read in, turn the TTY reader to FREE. Remove the paper tape.

A special control command, ECHO, may be used with TAPE to prevent the program from being listed while it is being read. The first time it is used, ECHO inhibits all printout. A second ECHO command restores normal printout.

Privileged Control Commands

Several optional commands are available which modify and control a program run. All of these commands are considered to be

privileged in the sense that their use is restricted. The privileged commands are available only if the privileged command capability was selected when EduSystem 30 was loaded. During normal system operation these commands are disabled. If a user attempts to use a privileged command, it is ignored and the system prints WHAT?

A special command, the PRIVILEGE command, is used to make these privileged commands available. To use PRIVILEGE, the user types PRIVILEGE and the RETURN key. The system then waits for the user to type a one to six character password code. (The typed characters are not printed.) At the time the system was loaded, a password was typed into the system. The characters typed after the PRIVILEGE command are compared to this password. If they match, the PRIVILEGE command is successful and all privileged commands may then be used. If they do not match, the message INVALID PASSWORD is printed and all privileged commands continue to be unavailable.

In short, the user must know the password in order to use any privileged command. It is important that the password be kept secret. For this reason, the password is never printed when the user types it. It is possible to change the password code at any time. The command, PASSWORD, which changes the code is, of course, a privileged instruction. Certain privileged commands are used only when processing card programs; these commands are discussed under Executing Card Programs. The other privileged commands, SAVE and UNSAVE, are discussed below. All privileged commands must be typed at the terminal; they cannot be entered on cards.

Using the System Storage Capability

EduSystem 30 allows the system device (DECdisk or DECtape) to be used for permanent on-line storage of programs. Programs stored in this way may be loaded instantly, without the need to load a paper tape or card deck or type a program.

SAVE AND UNSAVE COMMANDS

Two commands, SAVE and UNSAVE, may be used to change the contents of the system storage area. Because the amount of storage space is limited, and to prevent accidental erasure of stored

programs, both SAVE and UNSAVE are privileged commands. During normal system operation they are disabled. SAVE and UNSAVE may only be used after a successful PRIVILEGE command has been executed.

The SAVE command stores the programs currently in memory on the system storage area (DECtape or DECdisk) and gives it the name specified by the last NEW, OLD, or NAME command. Any existing program stored under this name is deleted. Thus, all stored programs have names which may be used to recall them in the future. If a SAVE is attempted when the privileged commands are disabled, the system prints WHAT? and ignores the command. If a successful PRIVILEGE command has been executed, but the storage area is full, the message NO SPACE is printed and the program is not stored.

The UNSAVE command is used to delete a program already stored. UNSAVE must be preceded by a NEW, OLD, or NAME command which specifies the name of the file to be deleted. The user must be certain to use exactly the same program name as he used when he first identified the program. Like SAVE, UNSAVE is ignored unless preceded by a successful PRIVILEGE command. If the program to be deleted does not exist in the system storage area, the message FILE NOT SAVED is printed and no program is deleted.

CATALOG COMMAND

The CATALOG command may be used to obtain a list of the names of all programs available in the system storage area. The CATALOG list includes the number of storage blocks used by the program. A standard EduSystem 30 includes 116 blocks of storage space (for nonstandard systems, see Calculating Available Storage). The CATALOG command may be used to determine how many of these blocks have been used and, hence, how many are free.

LENGTH COMMAND

If the system storage space is almost full and another program is to be saved, the LENGTH command may be used to determine if there is room to store the current program. If space is not available, an existing program must be deleted. In all cases, the maximum number of stored programs, regardless of size or storage device, is 62.

OLD COMMAND

The user may load programs stored in the system storage area at any time by typing the OLD command. After the OLD command is entered, the system prints OLD PROGRAM NAME. The user then types the name of the program to be loaded. The user must be certain to use exactly the same program name as he used when he first identified the program.

Returning to Batch Mode

At the end of an interactive session, the system is ready for further batch processing. When the system prints READY, the BATCH or STACK command may be used to begin a batch run.

Program Editing

There are two times when a program may require editing procedures. The first occurs while a line is being typed but before the RETURN key is pressed. The second occurs when a line has been completely typed and the RETURN key has been pressed. Each situation has its own editing procedures.

Procedure 1: Before the RETURN key is pressed.

Three keys may be used to correct typing errors: ALT MODE (or ESCAPE), ← (back arrow), or RUBOUT.

> ALT MODE (or ESCAPE) is used to delete an entire line. When this key is used, BASIC prints DELETED, erases that line from the program, and returns the carriage so that the line may be retyped.
>
> ← (back arrow), SHIFT/O on the keyboard, or RUBOUT is used to delete a character from a line. BASIC prints the back arrow, deleting the last character from that line. More than one back arrow deletes more than one character, in reverse order.

Procedure 2: After the RETURN key is pressed.

Once a line of the program has been transmitted to computer memory via the RETURN key, several methods of correction may be used. Lines may be inserted, deleted, changed, or renumbered.

> INSERTION: To add a line to a program, assign a line number that falls between two existing lines, type the line number and text, and press RETURN.

DELETION: To erase a line from computer memory, type the line number *only* and press the RETURN key.

CHANGE: To change an individual line, simply retype it. The old instruction is replaced by the new one.

RENUMBER: Occasionally, repeated editing and insertions result in there being no more room in an area of a program to insert new lines. It is then necessary to *spread out* the line numbers so there is room for new insertions. The RESEQUENCE command is used for this purpose. To renumber a program, type the RESEQUENCE command. This command changes the first line's number to 100 and renumbers each succeeding line with an increment of 10. RESEQUENCE also automatically changes all GOTO, GOSUB, and IF statements to correspond to the new line numbers.

WRITING AND RUNNING CARD PROGRAMS

Writing a Program on Cards

Programs to be submitted to EduSystem 30 are first transcribed onto specially formatted BASIC cards. These cards are preprinted with a series of small rectangular boxes. Each card has 39 columns of boxes, with the column number printed along the lower edge. Figure 7-1 illustrates a BASIC card.

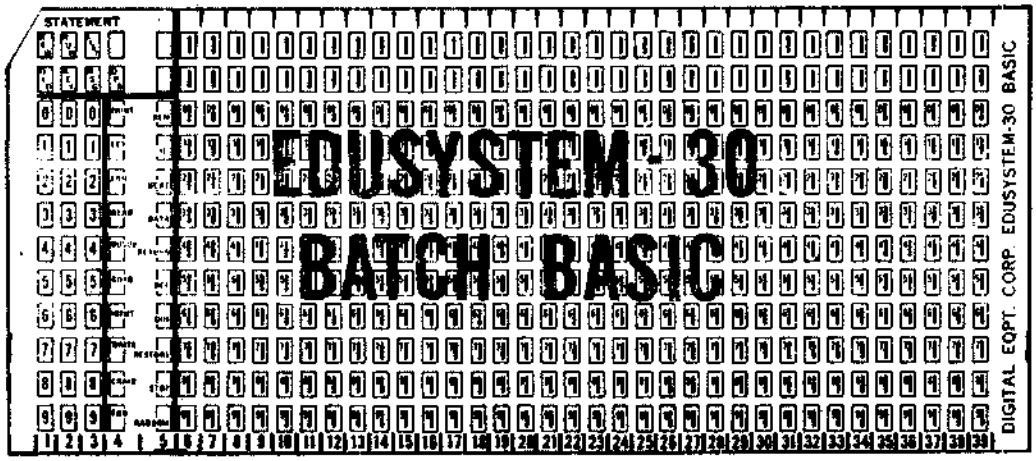


Figure 7-1. EduSystem 30 BASIC Card

Program statements are written onto the BASIC cards by marking these boxes with ordinary lead pencil. Each box is marked with a single heavy line, drawn either vertically or diagonally. It is not necessary to fill in the entire box. Avoid making any stray marks,

as the computer reads all marks on the card, whether or not they occur inside a box. Any stray marks cause the computer to misread the card. A program to be submitted to EduSystem 30 is marked on a series of these cards; each program statement is placed on a separate card.

LINE NUMBERS

The first item of each program statement to be marked on the card is the line number. All program statements must have line numbers from 1 to 999. The line number is marked in the leftmost three columns of the card. In each of these columns are 10 boxes containing the digits 0 through 9. (Above these numbered boxes are six other boxes which are *never* marked as part of a program statement. They are used as control cards and are explained later.)

The user marks an appropriate digit in each of three columns. If the line number is only one or two digits, it can be marked in any of the three columns. Blank line number columns are ignored. Figure 7-2 shows a card marked with the line number 120.

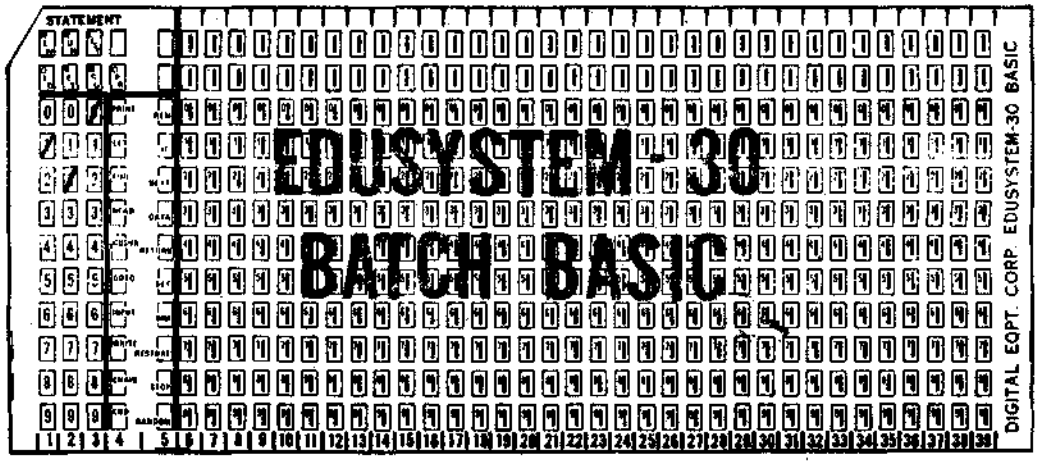


Figure 7-2. Line Number Example

Line numbers should be marked on cards in the order in which program statements are to be executed: the lowest line number on the first card, the highest line number (the END statement) on the last card. If the cards are not physically ordered by line number within the card deck, EduSystem 30 still executes the program in line-number order. The only time when the order of the cards is important is when two cards have the same line number. In that

case, only the last card in the deck which has that line number is used in the program.

BASIC STATEMENTS

BASIC statement types are marked after the line number, in column 4 or 5, within the outlined area. Each box in this area represents a single BASIC statement; thus, one mark in the appropriate box indicates the corresponding statement type. It is not necessary to spell out the statement. Since each statement is marked on a separate card, each program card should have only one mark in column 4 or 5. In Figure 7-3, the LET box has been marked. Line 120 is, therefore, a LET statement. (The implied LET cannot be used with card programs.)

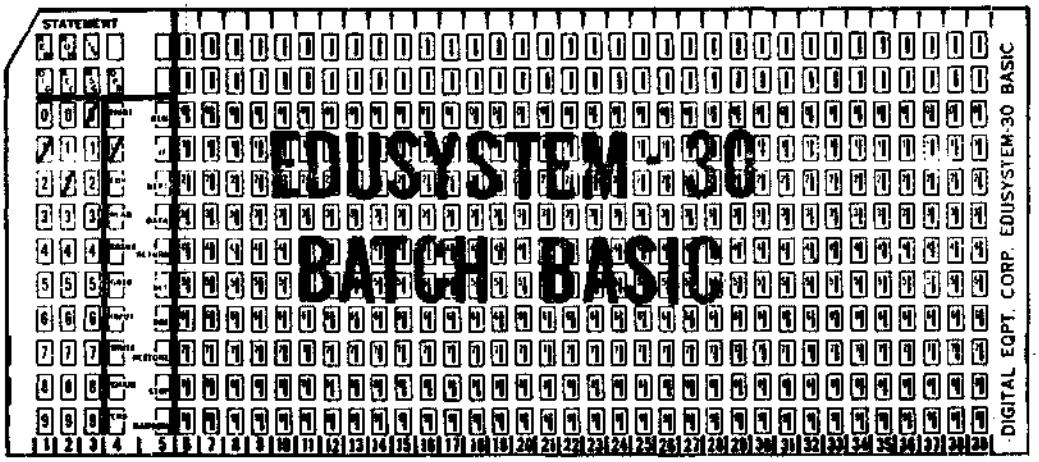


Figure 7-3. Statement Example

STATEMENT OPERAND

The line number and statement type are marked in specific sections of the card. The remainder of a statement, the statement operand, is marked in a different manner in columns 6 through 39. The first step in this process is to write the remaining characters of the line in the boxes across the top of the card, one character to a box, starting in column 6. The boxes below each of these characters are then marked with the aid of the BASIC template, as shown in Figure 7-4.

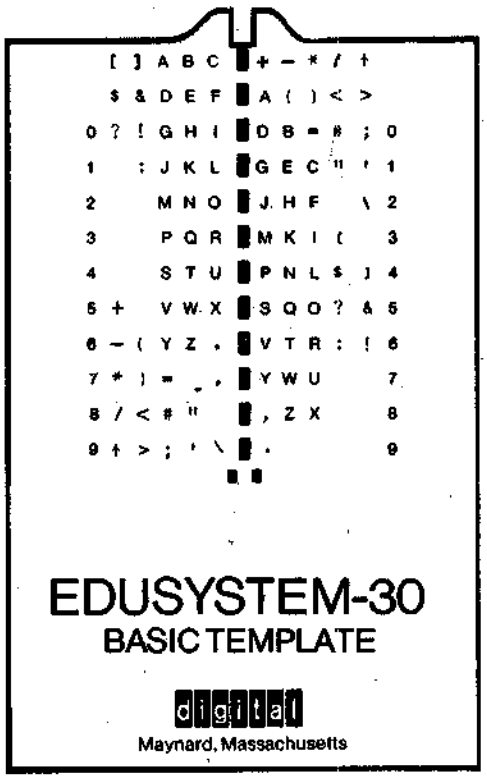


Figure 7-4. EduSystem 30 BASIC Template

The template has a column of holes down the center with the characters used in writing an operand printed on either side. All characters are printed twice on the template, once on the left and once on the right. The top hole in the template corresponds to the blank boxes at the top edge of the card where the characters of the operand were written. The bottom pair of holes correspond to the row of heavy black marks along the lower edge of each card. The twelve holes in between correspond to the twelve boxes in each card column.

The following procedure is used to mark the statement operand characters:

1. Place the template on the card so that the character to be marked appears in the top box of the template and the two black marks appear in the bottom pair of holes.
2. If the character is a digit (0 to 9), find that digit on either side of the template and mark the one box which is beside that digit.

3. If the character is not a digit, find the two occurrences of that character on the template, one on the left and one on the right. Mark the two boxes which are beside these characters.
4. If the character is a space, make no marks.

An example of the marking procedure is shown in Figure 7-5.

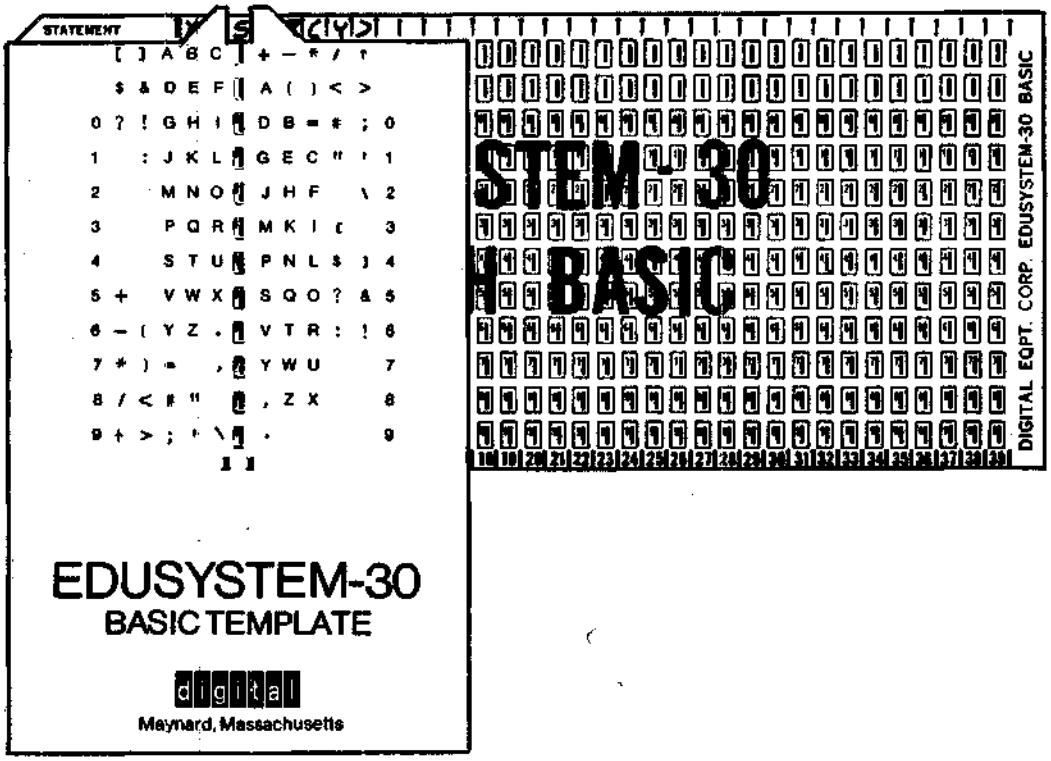


Figure 7-5. Marking the Statement Operand

SUMMARY OF CARD MARKING PROCEDURE

A completed card is marked in three places:

1. Columns 1 through 3 to mark the line number.
2. Column 4 or 5 to indicate the BASIC statement type.
3. Columns 6 through 39 to indicate the statement operand, as written across the top of the card.

Figure 7-6 shows two examples of completed cards.

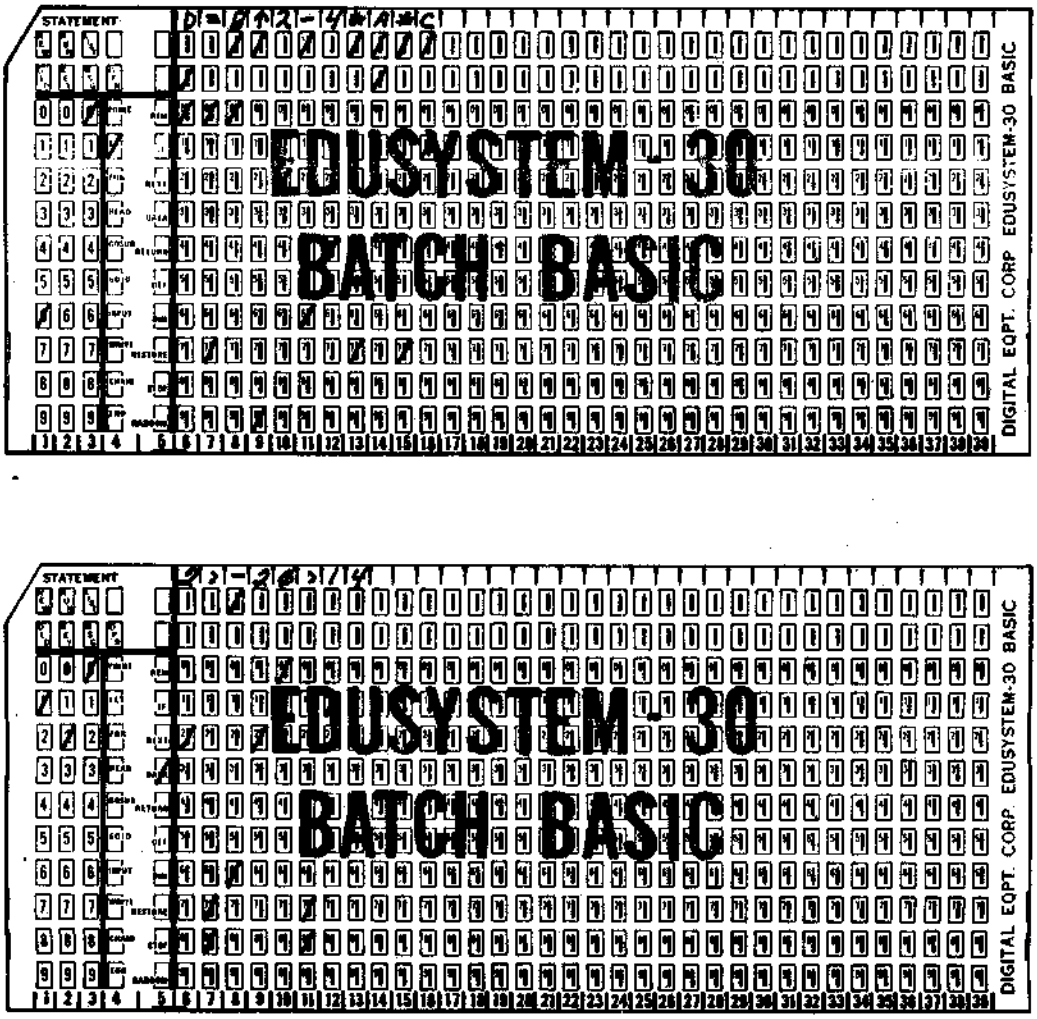


Figure 7-6. Completed BASIC Cards

Submitting a Program to be Run

When all statements of a program have been marked on cards, the cards are collected into a single deck for submission to the computer. However, in addition to the deck of program cards, the computer must be given the program name and explicitly instructed to execute the program. Special cards, called control cards, are added to program deck for this purpose. Control cards are not part of the actual program. They do not have line numbers and are placed at the beginning and end of the program deck.

THE NEW CARD

The first card of any deck submitted to the computer must be a NEW card or an OLD card[4] (see Using a Stored Program), which contains the name of the program. Any cards in a deck which come before a NEW, or OLD, card are ignored. If no NEW, or OLD, card is included in the deck, the entire program is ignored. A NEW card has a single mark in the NEW box in the upper left-hand corner of the card. It has no other marks in columns 1 through 5. The name of the program, which can be 1 to 6 characters long, is marked using the template and starting in column 6. The remaining columns of the NEW card, starting in column 12, should be used to mark identifying information such as the name of the programmer.

THE LIST CARD

When the computer reads a program, it reads the marks in the columns rather than the characters printed across the top of the card. If the columns below each character are accurately marked, the computer reads the card as intended. Erroneous marks, however, cause the computer to read the statements incorrectly. It is useful to obtain a printed listing of the program exactly as it was read by the computer. The LIST command instructs the computer to print such a listing. A LIST card contains a single mark in the LIST box, the topmost box in column 3. The LIST card should be placed after the program deck, immediately before the RUN card.

THE RUN CARD

Each program deck submitted to the computer must have a RUN card following the last program card. A RUN card has a single mark in the RUN box, the topmost box in column 2. The RUN card is the signal for the computer to execute the program. If a program deck has no RUN card, the program is loaded into the computer but never executed. The RUN card is normally the *last* card in the deck.

---

[4] Since the NEW (or OLD) card is the first card in the deck, it is often marked on a colored card to make it easy to spot the beginning of separate programs in a stack of many programs. Control cards can be easily identified if the system manager runs a felt-tip marker over the tops of a box of cards and makes these cards available to students for use as control cards.

SUMMARY

A complete program deck, ready to be run by the computer, consists of the complete set of program cards preceded by a NEW card and followed by a LIST and/or a RUN card. If other control cards are being used, they may be inserted as necessary. Control cards should be placed before or after the program deck, not inserted within it. This complete deck is then submitted to the computer operator to be run. (If there is no system operator, each student must run his own program through the computer as described under Executing Card Programs.) Figure 7-7 shows a BASIC program deck.

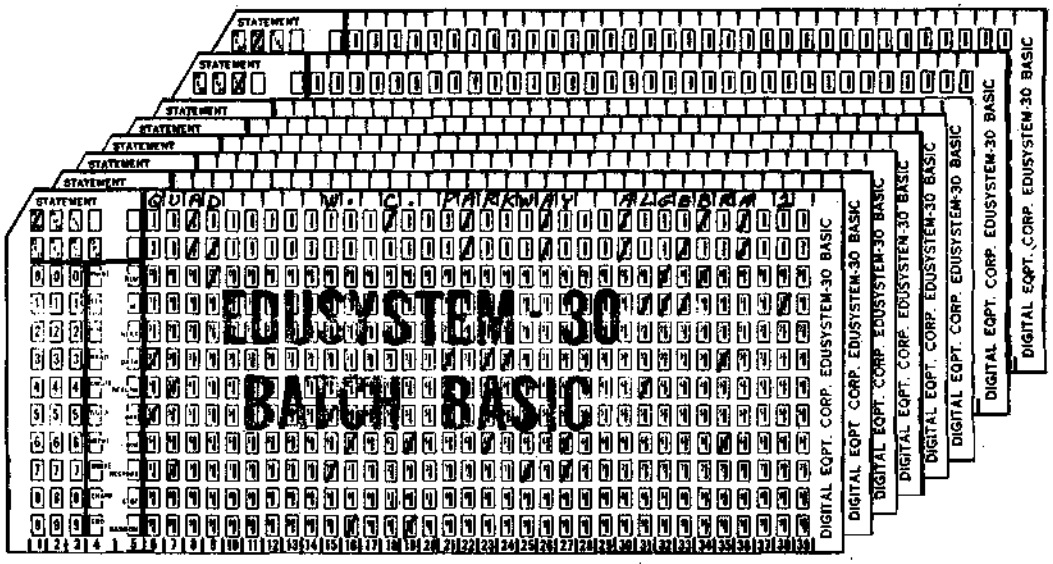


Figure 7-7. BASIC Program Deck

Getting the Results of a Computer Run

Whenever EduSystem 30 processes a card program, it prints the results of that run on the Teletype. The format of this output is as follows:

1. Beginning of Output—A row of dashes (-------) is printed to indicate the beginning of program output and thus separate each program from the output of the previous program.
2. Identification Line—The information on the NEW card is printed at the top of the first page of output. This information usually includes the name of the program and the name of the programmer.

3. Program Listing—If a LIST card is included directly after the program deck, a listing of all statements in the program is printed. It is preceded by a header line which includes the name of the program and a short message selected by the system operator.
4. Program Output—If a RUN card is included at the end of the deck, the program is executed. The results of all PRINT statements are printed as program output. The printed program output is preceded by the same header as the program listing.

Using a Stored Program

In all previous examples the program to be executed is submitted on a card deck. The NEW card at the front of the deck indicates that the program to be run is a new one (i.e., a program the computer does not already have stored away) and that it is contained in the cards which follow the NEW card. The computer is also capable of storing programs for future use. Stored programs are considered to be old programs. They do not have to be submitted on cards each time they are run.

An OLD card is used to call a program which has been stored in the system previously. The OLD card has a single mark in the OLD box in column 1 of the card. All programs stored within the system have names. The name of the old program to be used must be marked starting in column 6. The user must be certain to use exactly the same program name as he used when he first identified the program. For example, blanks which preceded the program name or appear in the name itself must be included. The OLD card is used *instead* of a NEW card; it is, therefore, the first card in the deck. Like the NEW card, it should have identifying information coded in the columns after the program name. (Like the NEW card, OLD is often marked on a colored card.)

Since the program is already stored in the system, the OLD card need not be followed by a program deck. The next card can be the RUN card. Two cards, an OLD card followed by a RUN card, are all that is needed to run a stored program. If the user wishes to supply some of his own input data to an OLD program, this data is marked as regular DATA statement cards. These DATA cards are placed between the OLD card and the

RUN card. They are merged into the stored program before it is executed.[5]

Interacting With the Operator

If card programs are being run under the supervision of a system operator, he may be requested to perform certain functions, such as entering data or making minor modifications to the program.

An OPR card is used to communicate with the operator. An OPR card has a single mark in the OPR box in column 4 of the card. Any message can be marked in columns 6 through 39. When an OPR card is encountered in a program deck, the message in columns 6 through 39 is printed and a bell rings to alert the operator.

Normally, the OPR card is used to request the operator to perform some function which a card user cannot perform directly. Operations such as saving programs in the system storage area can only be done by typing commands at the Teletype.

A KEY card is used to have the computer pause so that the operator can type one or more commands. The KEY card contains a single mark in the KEY box in column 2 of the card; it should be the last card in the deck. KEY should, of course, be preceded by one or more OPR cards, telling the operator what to do when the computer pauses. For example, BASIC programs can request data from the operator. An INPUT statement is used to request data from the Teletype, such as the current date.

Editing and Rerunning a Program

After a program has been executed by means of a RUN card, it is still available in core memory and may be modified and/or run a second time. New lines, or new data, may be added to the program, or may replace existing program statements. Any program lines to be added should *follow* the first RUN card. After these new cards, a second RUN card should be included. The

---

[5] The DATA statements submitted on cards must have line numbers which differ from the line numbers of the stored program. Otherwise, these DATA statements will replace program statements. Also a stored program which is to be used with card data should have no DATA statements of its own.

number of RUN cards allowed per deck is set by the system operator and is normally two.

Inserting Messages in the Program Printout

One line messages can be inserted into program output by using an MSG card. An MSG card has a single mark in the MSG box in column 3 of the card. Columns 6 through 39 can contain any message. This message is printed as part of the program output.

Sample Program

The following sample program demonstrates the use of Edu-System 30 as a computational tool by students who have no knowledge of computer programming. The program is already stored in the computer.

PROBLEM

Students in a physics class are performing a lab experiment whose results require very lengthy data reduction. Rather than have each student perform these calculations manually, the instructor wrote a program to do it and stored the program under the name LAB37. This program has no line numbers between 900 and 950. Each student is to prepare a deck containing his data, in some predefined order with line numbers between 900 and 950, and submit it to the computer.

PROCEDURES

Each student prepares a deck containing his data and submits it to the computer. The decks are prepared in the following manner:

1. Each student writes his lab data values across the top of one or more cards, starting in column 6. Values on any one card are separated by a comma.
2. When all data values have been obtained, the student completes the marking of the data cards. The DATA box in column 5 of each card is marked. The first card is marked with line number 900 in columns 1 to 3. Succeeding cards are given line numbers 901, 902, etc. Finally, using a template, the student fills in the boxes in columns 6 through 39.
3. An OLD card is marked with a single mark in the OLD box, the program name LAB37 in columns 6-10, and the student's name after column 11. The OLD card is placed

before the data deck. Next a RUN card, with a single mark in RUN box is placed behind the deck. These cards are then submitted to the computer.

PRINTED RESULTS

The following is a sample printout of the results of one student's input to the LAB37 program:

```
OLD LAB37     A. EINSTEIN     PHYSICS 1

RUN

LAB37     EDUBASIC


DATA SET # 1

SAMPLE SIZE = 12
SUM = 444
MEAN = 37
STD. DEVIATION = 7.51665


DATA SET # 2

SAMPLE SIZE = 8
SUM = 32.7
MEAN = 4.0875
STD. DEVIATION = .613865
```

EXECUTING CARD PROGRAMS

Once EduSystem 30 has been loaded and started, it is available for processing program decks (if the batch capability was retained at system load time). It is not necessary to reload or restart the system each time it is used.

Normal Batch Operation

Perform the following operations to process batch (multiple programs) of marked cards. Collect all programs to be run into a single deck. Any number of programs may be included so long as the deck fits into the card reader (maximum of 400 cards).

1. Arrange all cards so that they face in the same direction.
2. Place the deck in the reader, face down with the row of black marks on the bottom of each card placed inward toward the back of the reader.

3. Ensure that the reader power switch is ON, then press the START button. At this time, all red lights on the card reader control panel should be off; the single green light should be on.
4. Type BATCH. The BATCH command is used to commence processing of the entire batch of programs loaded into the reader.

Once a batch run is initiated, it continues until all cards have been processed. EduSystem 30 then asks:

```
MORE CARDS?
```

5. Type N if no more programs are to be executed during this run. EduSystem 30 concludes the batch run by typing READY. If more programs are to be executed, continue at step 6.
6. Place the additional program cards in the reader, then press START. Respond to the MORE CARDS? question by typing Y. The batch operation restarts.

All output from a batch is printed at the Teletype (or possibly on a line printer, see Using Optional Hardware). The output of individual programs is separated by rows of dashes (------).

Executing Card Programs Individually

EduSystem 30 may be used as a system in which each user loads and executes his own card program. The system is started exactly as it is for a batch run; the BATCH command is typed, even though no cards are in the reader. The system, finding no cards, asks:

```
MORE CARDS?
```

At this time the system is ready to accommodate a single user with a card deck. The single user can proceed with the operating procedures described under Normal Batch Operation. When all cards have been processed, the system again asks:

```
MORE CARDS?
```

The user may then tear off his program output and leave the system ready for the next user.

Controlling a Batch Run

A batch operator may control the execution of card programs. CTRL/C may be typed at any time to terminate a program run and automatically continue with the next program. CTRL/C does not interrupt the loading of a program, only its execution or listing. When CTRL/C is used to terminate program execution, the message STOPPED BY OPERATOR is printed.

The ALT MODE key (labeled ESC on some terminals) may be used to interrupt the system while it is reading cards (i.e., halt the batch operation) and allow the operator to type commands at the Teletype. The operator can then type BATCH to begin processing of the next program in the deck.

Several optional commands (BATCH, MAX, HEADER, STACK, and LOG) are available to modify and control a batch run. All of these commands are privileged commands (see Privileged Control Commands). They can be used only if privileged command capability was retained at system load time and if the user knows the correct password.

BATCH COMMAND

In the privileged mode, the BATCH command may be followed by a single space and an integer which is less than 4000. For example:

```
BATCH 20
```

This number is the maximum number of RUN commands which may be executed by any single program deck. If privileged commands are not enabled or if no number is specified, the BATCH command is executed and 2 is the number of RUN commands (cards) allowed per program.

MAX COMMAND

Occasionally, a program is submitted which does not terminate. It will run indefinitely (in a loop) and delay the execution of other programs until the operator stops it. The MAX command can be used to automatically terminate programs which run too long. MAX defines the maximum number of BASIC program

statements which may be executed in a program. The system counts instructions in units of 200. Thus, the limit set by the MAX command is the number of 200 instruction units which are allowed per program. If very small programs are being run, a MAX of 10, allowing execution of 2000 statements, is reasonable. This command is typed as follows:

```
MAX 10
```

More complex programs need to execute more than 2000 statements. In this case, a correspondingly higher limit should be set. If no MAX is set, or if the operand of MAX is 0, programs may be run indefinitely. A MAX limit is only in effect for the succeeding batch run. If the program being run executes a CHAIN instruction, the instruction count is restarted.

HEADER COMMAND

Before running or listing a program, the system prints a header line consisting of the name of the program followed by the message EDU BASIC. The HEADER command may be used to change this message to any new header composed of 1 to 12 characters. The new header is then used for all programs in the next batch run. Thus, this command can be used to identify the programs in that batch. For example, if a batch consists of all the programs from a given class, the name of the class or of the instructor might be used for the header. To use the HEADER command, type HEADER and press RETURN; then type the new header message and press RETURN again. For example:

```
HEADER
PHYSICS 101
```

STACK COMMAND

The KEY command and the INPUT statement may be used by a card program to interact with the operator, as described previously. When a batch is being processed unattended, these commands stop program execution. Therefore, the STACK command is available to start up an unattended batch run. The STACK command makes all INPUT and KEY instructions illegal and causes processing of card programs to begin. Like the BATCH command in privileged mode, the STACK command may be followed by a space and an integer which specifies the number of

RUN commands allowed per program. For example, if the STACK command is typed as:

```
STACK 4
```

it allows any program to be executed four times. This number is 2 by default if not explicitly specified or if the system is not in privileged mode. Execution of BATCH, STACK, NEW, or OLD commands automatically locks out the privileged instructions. Subsequent privileged instructions must be preceded by a successful PRIVILEGE command.

LOG COMMAND

As programs are run, a system log is created which contains a record of all the programs which have been run. More specifically, the log retains all information included on the NEW or OLD card which precedes each program deck. NEW or OLD cards generally contain such information as the name and class of the student who submitted the program. The privileged command LOG may be used to obtain a printed listing of this log.

If a very large number of programs are run before the log is printed, or if the amount of space reserved for the log when the system was loaded is small, the log may become full; if this happens, the log is automatically printed at that time without a LOG command being given.

Hands-On Interaction Versus Batch

If the user wishes to get maximum throughput of student programs, he should use the STACK command. However, if the user wishes to allow hands-on student interaction with a high throughput, he uses the BATCH command and implements the plan outlined below.

Most student interaction in programming occurs during debugging. The EduSystem 30 BATCH command allows the user to mix hand-on use with high throughput. Typing BATCH with no following number allows each program to be run a maximum of two times. A programming class use of EduSystem 30 is optimized by entering cards in the following order:

NEW program name student, class
card deck

LIST card
RUN card
KEY card

EduSystem 30 reads the deck, lists the program, and attempts execution. The student's error messages print at this time, if errors occur, and the KEY card allows the student to sit down at the terminal and correct the errors on-line. When errors have been corrected, the student types his second RUN, getting his second attempt at execution. Upon completion of the second RUN, EduSystem 30 automatically reads the next student's cards.

This use of BATCH with the KEY cards allows each student two successive attempts at a successful RUN and brings much of the excitement of interaction into the EduSystem 30 batch operation.

ERROR MESSAGES

Some programs execute correctly the first time they are tried. Most others, especially if they are at all complex, have errors in them. EduSystem 30 checks all statements and commands when they are entered (on cards or from the interactive terminal) and before executing them. If it cannot execute a statement or command, the system informs the user by printing one of the following types of messages.

Batch Mode Program Loading Errors

The following messages are printed if errors occur while BASIC is loading a card program.

Message	Explanation
BAD CHARACTER	One or more characters on a card were not understood. The character % replaces any unreadable character. The information on the card is processed if possible. This error *does not* cause any subsequent RUN commands to be ignored.
BAD CHARACTER—LINE NOT USED	Same as BAD CHARACTER message except that the information on the card is ignored. This error *does* cause subsequent RUN commands to be ignored.
WHAT?	Line does not make sense to the system. It does not begin with a line number and is not a valid system command or statement.

Message	Explanation
NO ROOM	The program is too large to be loaded, i.e., it is greater than 5000 characters. Larger programs should be run in interactive mode or CHAINed.
FILE NOT SAVED	The program named on an OLD control card is not available in the system storage area.
FILE TOO BIG FOR BATCH MODE	The program named on an OLD card is too large to be run in batch mode. It must be run interactively.

Interactive Mode Program Loading Errors[6]

As each line is typed, EduSystem 30 checks it for program loading errors. If it finds an error, it prints one of the following error messages immediately after the erroneous line.

Message	Explanation
WHAT?	Line does not make sense to the system. It does not begin with a line number and is not a valid system command.
LINE NO. TOO BIG	The line number of a line or the argument of a system command is greater than 4095.
LINE TOO LONG	Line just entered is longer than 80 characters.
NO ROOM	There is no room to store the line just entered.
FILE NOT SAVED	The program named as the operand of an OLD command was not previously saved on the system device.
NO SPACE	There is not enough space on the DECtape to SAVE the current program.
Bell	If an invalid character is entered, the Teletype bell rings and the character is ignored.
I/O ERROR	An input or output error occurred on the DECtape unit. Be sure that the unit is on-line, write-enabled, and the unit number is set correctly. Retry whatever was interrupted by the error. If the problem persists, there is a hardware problem. Contact the system administrator or DEC field service.
INVALID PASSWORD	The password typed after a PRIVILEGE command is not the system password. Privileged mode is not entered.

[6] To correct program loading and coding errors indicated by these messages, the appropriate line in the program must be corrected in the manner described in the Program Editing section, Procedure 2.

Coding Errors [7]

After the RUN (or RUN NH) command is entered, EduSystem 30 checks each statement before executing it for mistakes in the BASIC program coding. If the system cannot execute a statement, it informs the user by printing one of the following messages and the line number in which the error was found.

Message	Explanation
CH	There is an illegal character in the line.
EN	Program does not have an END statement as the last line in the program.
FN	Not enough NEXT statements in the program. There must be a NEXT statement for each FOR statement in the program.
FO	FOR and NEXT statements do not match. There is a NEXT statement in the program whose variable is not the same as the variable in the corresponding FOR statement.
LI	Line contains an improperly written decimal number or constant. It may, for example, have two decimal points or have an alphabetic character in it.
M1 M2	The program as a whole is too big to be run by the system. Making the program smaller, reducing the size of subscripted variables, or using the NOLINE command may help.
NE	Program has too many (more than 8) FOR-NEXT loops one within another.
PC	Line contains an improperly used parenthesis. Generally, the problem is an expression which does not have an equal number of left and right parentheses.
RO	Statement contains an invalid *relational* operator (<, =, >, <=, >=). Relational operators may *only* be used in IF statements.
S1 S2	Statement as a whole is not properly written and, as a result, does not conform to proper BASIC syntax. For example, a semicolon is allowed in a PRINT statement but not in a READ or INPUT statement.
ST	Statement word is not one of the legal BASIC statement types.
TB	The program is too big to be run. Cause is usually an extremely large number of PRINT statements.

[7] To correct the program loading and coding errors indicated by the messages, the appropriate line in the program must be corrected in the manner described in the Program Editing section, Procedure 2.

Message	Explanation
TO	Program is either too large or too complex to be run. The total number of variables, constants, functions, and line numbers should be reduced, if possible.
UL	A GOSUB, GOTO, or IF statement contains a line number which does not exist.
UQ	A quotation mark indicating the beginning of a string of text does not have a corresponding quotation mark at the end of the text string.

Program Logic Errors[8]

Some errors do not show up until the program is actually executed. An example of this type of error is an expression which uses a square root of a variable. If, when this square root is actually calculated, the variable has a negative value, a program logic error has occurred. EduSystem 30 prints the following messages if program logic errors occur.

Message	Explanation
CH	A CHAIN statement tried to chain to a program which was not available in the DECtape storage area.
CO	Program ran too long and was automatically stopped by the system.
DA	The program ran out of data. It attempted to do a READ after all data had been read.
D0	The program attempted to divide by zero. Instead of dividing by zero, BASIC divides by the smallest possible number, giving a result of about 10^{500}. This error does not cause the program to stop.
FN	An expression contains a function which was not defined in a DEF statement.
GS	The program is too complex to be executed. The problem is generally that too many subroutines have themselves executed GOSUB instructions.
LG	Program attempted to take the logarithm of a negative number or zero.
RE	A RETURN statement was used outside of a subroutine or a subroutine was entered by a GOTO instead of a GOSUB.
SP	See GS.

[8] Some program logic errors may be corrected by the method described in the Program Editing section, Procedure 2. Most, however, necessitate the rewriting of the program.

Message	Explanation
SQ	Program attempted to take the square root of a negative number. BASIC automatically takes the square root of the absolute value of the number instead. This error does not cause the program to stop.
SS	Program used a subscript which was too big for the variable. The maximum size of a subscript is specified in a DIM statement.
WR	There is no room to write data. The program attempted to do a WRITE statement when the data list was full. (Note that if this error occurs, the program text will no longer be intact. A NEW, OLD, or SCRATCH command must be used to clear the program area.)

OPERATING INSTRUCTIONS

Loading EduSystem 30

The EduSystem 30 software is supplied on a paper tape. This tape must be loaded into computer memory when the system is first installed. Once loaded, the software need not be reloaded. Perform the following procedures to load the EduSystem 30 software.

INITIALIZE THE DECDISK

If the system device is DECdisk, perform the following steps to prepare the disk for software loading:

1. Ensure that the disk unit is on (red light illuminated inside the back of the computer cabinet).
2. Ensure that all disk protect switches (located inside the front of the computer cabinet) are in the off (down) position.

INITIALIZE THE DECTAPE UNIT

If the system device is a DECtape unit, perform the following steps to prepare the unit for software loading.

1. Set the REMOTE/OFF/LOCAL switch to OFF.
2. Place a DECtape on the left spindle with the DECtape label out.
3. Wind four turns of tape onto the empty reel on the right spindle.
4. Set the REMOTE/OFF/LOCAL switch to LOCAL.
5. Wind a few turns of tape onto the right spindle reel with the → switch to ensure that the tape is properly mounted.

6. Dial 0 on the unit selector dial.
7. Set the REMOTE/OFF/LOCAL switch to REMOTE.
8. Set the WRITE ENABLE/WRITE LOCK switch to WRITE ENABLE.

INITIALIZE COMPUTER MEMORY

The EduSystem 30 software tape may be loaded into memory in one of two ways: with an automatic loader (hardware bootstraps) or with the Read-In Mode (RIM) loader program. The following loading instructions are for an EduSystem 30 that includes a hardware bootstrap. If the EduSystem does not have a hardware bootstrap, see Appendix A for instructions on using the RIM loader.

1. Turn the key lock on the computer console to POWER.
2. Turn the Teletype to LINE.
3. Place the EduSystem 30 tape in the appropriate reader (high-speed or Teletype) with the leader code (ASCII 200) over the read head.
4. Set the SWITCH REGISTER (SR) to 5356 (octal).[9]
5. Press and raise the HALT switch.
6. Turn on the appropriate paper tape reader.
7. Press and raise the SW switch.

The tape should begin to move. If it does not, ensure that the correct tape is being used and that tape is positioned over the read head. Repeat the above procedures.

System Building Dialog

When a portion of the EDUSYSTEM-30 paper tape has been read, BASIC prints a series of questions (see Figure 7-8) concerning the system configuration. The user responds by typing Y for yes and N for no, followed by the RETURN key. The first question is:

```
STANDARD SYSTEM?
```

EduSystem 30 has several optional operating modes and may be used with a variety of hardware components. A response of Y to the above question causes EduSystem 30 to build a system as it

[9] An explanation of the octal, or base 8, number system is included in *Introduction to Programming 1972*.

would if the following answers were given to the other system building questions:

```
IS SYSTEM DEVICE A DF32 DISK?Y
HOW MANY DISKS?1
DO YOU WANT BATCH CAPABILITIES?Y
HOW MANY BLOCKS FOR LOG?6
BATCH INPUT ON PUNCHED CARDS?N
DO YOU WANT PRIVILEGED COMMAND CAPABILITY?Y
TYPE INITIAL PASSWORD    (EDU-BASIC)
DO THE FOLLOWING EXIST:
    HIGH-SPEED PUNCH?N
    LP08 PRINTER?N
```

Basic then asks:

```
PROGRAM LIBRARY INITIALLY EMPTY?
```

The user responds Y if there are no programs to be saved on the system device. If programs are to be saved, the response must be N and the questions in boxes 1 through 8 of Figure 7-8 must be answered in the same way as when the system was built.

If the user types N in response to the question STANDARD SYSTEM?, EduSystem 30 recognizes that a nonstandard system is being loaded and asks the following questions:

```
IS SYSTEM DEVICE A DF32 DISK?
TC01 DECTAPE?
RF08 DISK?
LINCTAPE?
TD8E DECTAPE?
```

This series of questions is asked only until a response of Y is typed. If the device being described is not the one being used in the system, the user responds by typing N. If the system device is a disk, either DF32 or RF08, the system asks:

```
HOW MANY DISKS?
```

The user responds by typing the number of disk surfaces available in the system (1, 2, 3, or 4) and then typing the RETURN key.

If the system device is TD8E DECtape, the system asks:

```
DO YOU HAVE A TD8E ROM?
```

Respond Y if the system has a TD8E Read-Only Memory. If the response is N, the system asks:

```
8K OF CORE?
```

If the system device is DECtape, the system must include either a TD8E ROM (TD8E DECtape only) or 8K of core memory. If the response to the previous question was Y, this question is not asked.

EduSystem 30 then asks:

```
DO YOU WANT BATCH CAPABILITIES?
```

If the system is to process card programs, respond Y. If it is *only* to be used as an interactive system, respond N. If batch capability is selected, the system then asks:

```
HOW MANY BLOCKS FOR LOG?
```

The log is a stored record of system usage which records the name of all programs submitted for execution. The larger it is, the more program runs may be recorded before the log becomes full and must be printed. The log may be from 0 (no log at all) to 22 blocks of disk storage. Each block will hold the record of approximately 10 to 20 programs. Respond by typing the number of blocks to be reserved for the batch log, then typing the RETURN key. The system then asks:

```
BATCH INPUT ON PUNCHED CARDS?
```

Respond N if the batch input is to be coded with the EduSystem 30 template on the standard 39-column EduSystem 30 mark-sense cards. Respond Y if input will be on cards punched using a keypunch.

The system then asks:

```
MAXIMUM DATA COLUMNS PER CARD?
```

The maximum allowable response to this question depends upon the type of reader used: the punched-card reader has a maximum of 80 columns per card; the mark-sense reader maximum is 40 columns per card. See Using Optional Hardware for more information on preparing cards using a keypunch.

Next, the system asks:

```
DO YOU WANT PRIVILEGED COMMAND CAPABILITY?
```

The privileged command capability prevents unauthorized users from executing certain critical system commands. To establish this protection, type Y.

The system then prints:

```
TYPE INITIAL PASSWORD
```

The password is a special code which must be known in order to use privileged commands. It is up to six characters long. Respond by typing the desired system password. If it is less than six characters, type the RETURN key after the last letter of the password. To protect their secrecy, the characters typed are not printed. If no privileged capability is desired, the response to the original question is N. (A standard system keeps the privileged capability and sets the initial password to BASIC.)

Finally, the system asks if a high-speed paper tape punch and/or line printer are part of the system by printing:

```
DO THE FOLLOWING EXIST:
    HIGH-SPEED PUNCH?
    LP08 PRINTER?
```

The response to each question must be Y if the device exists, N if it does not. A standard system assumes that neither device exists. (See Using Optional Hardware for a discussion of how these devices are used.)

When all questions have been answered, the system asks:

```
IS THE ABOVE CORRECT?
```

If all questions were answered properly, type Y. The system then loads the rest of the EduSystem 30 paper tape. If any of the re-

sponses were incorrect, type N; the complete set of questions is repeated.

When the entire tape has been read, EduSystem 30 gives the user a chance to load additional DEC-supplied system update tapes by asking:

```
MORE INPUT?
```

If no DEC-supplied update tapes exist, respond N and EduSystem 30 is completely loaded. If update tapes do exist, load the first one into the tape reader and type Y to obtain loading.

Finally, when all input has been read, EduSystem 30 indicates that it is ready to process BASIC programs by printing:

```
READY
```

At this time, turn the key lock to PANEL LOCK and remove the key to prevent the system from being accidentally disturbed.

DIAGNOSTIC MESSAGES DURING SYSTEM BUILD

The following error messages are printed when errors are detected during the building of EduSystem 30.

```
TAPE READY?
```

This message is printed whenever the system is waiting for the paper tape reader to be loaded. It may appear by itself, usually due to a tape tear or reader jam, or it may appear as the last line of another diagnostic message.

ACTION:

1. The portion of the paper tape which is read after the system building dialog has distinct blocks of information about two and one-half tape fanfolds long. The start of such a block is indicated by nine blank tape frames followed by a frame with all positions punched.
   Back up the tape several fanfolds to the beginning of a previously read block. Position the tape such that the blanks at the beginning of the block are over the read station.
2. Type Y on the interactive terminal.

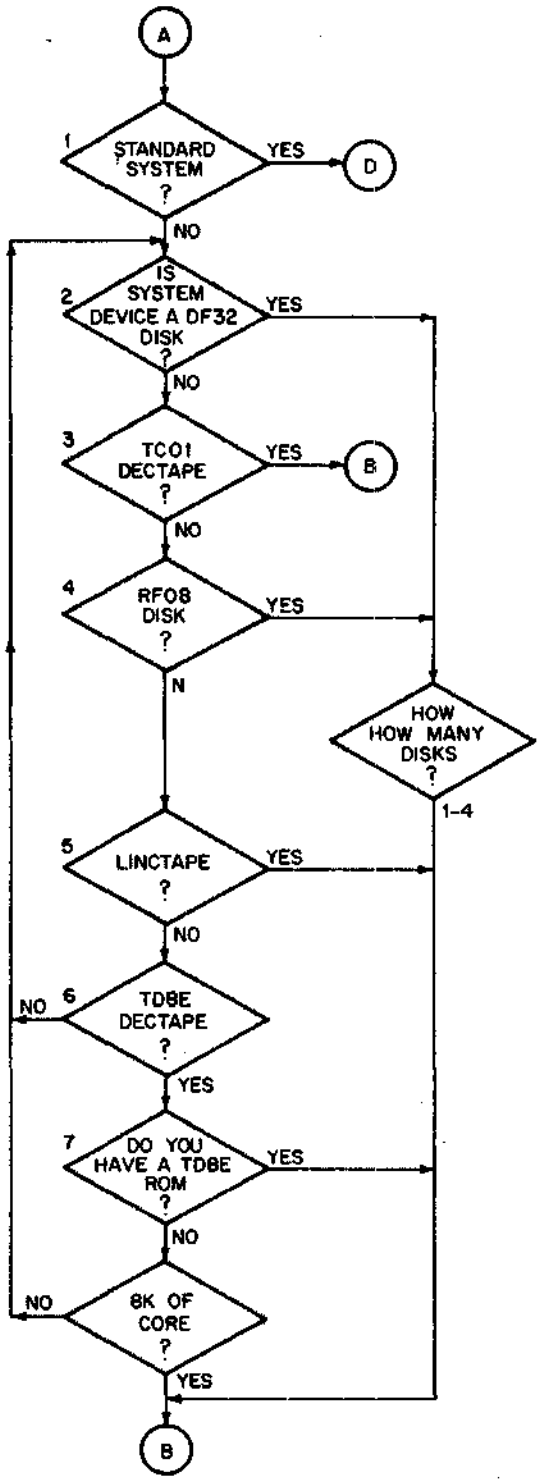


Figure 7-8. System Building Dialog

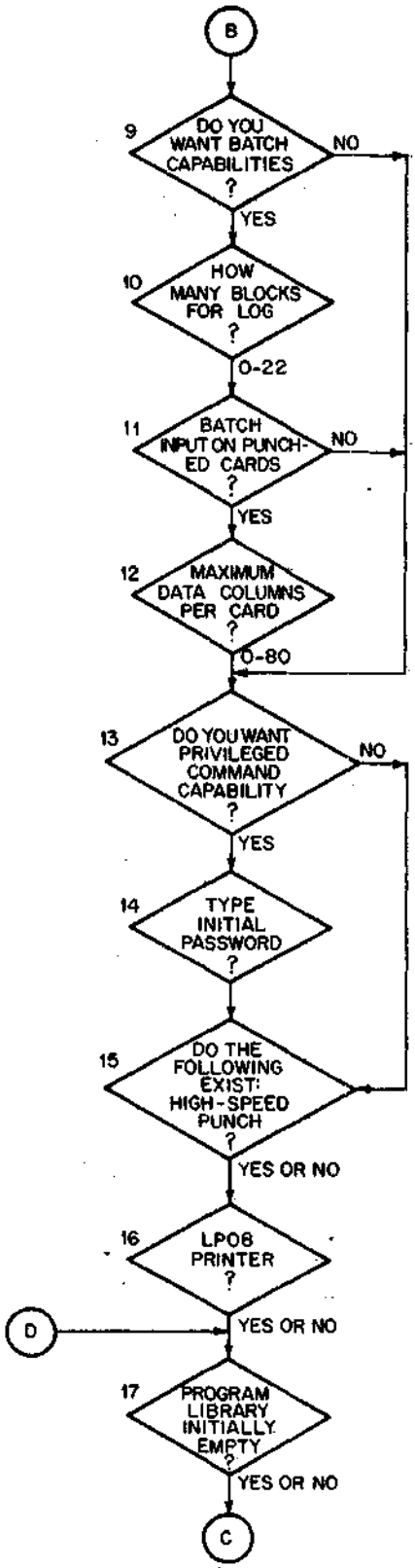


Figure 7-8 (Cont.). System Building Dialog

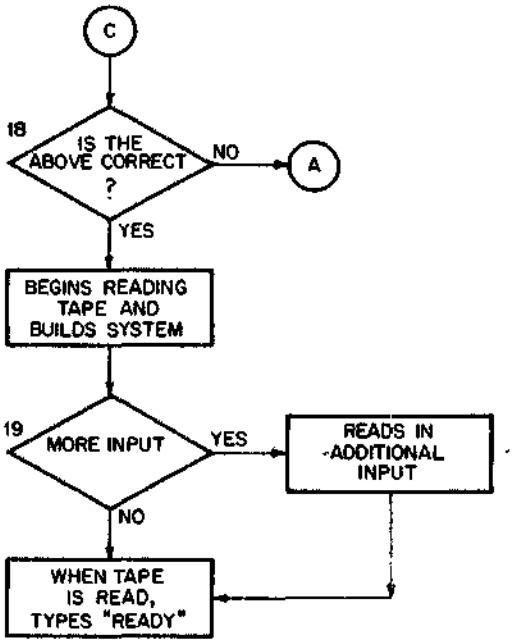


Figure 7-8 (Cont.). System Building Dialog

```
BAD PLACE TO START TAPE
TAPE READY?
```

This message means that after a previous message the user did not correctly position the tape to the beginning of a data block. (See discussion under TAPE READY? message.)

ACTION:

1. Correctly position the tape.
2. Type Y on the interactive terminal.

```
CHECKSUM ERROR
TAPE READY?
```

A checksum error occurred while the most recent data block was being used.

ACTION:

1. Back up the tape to the beginning of the block.
2. Type Y to reread the data.

```
SYSTEM DEVICE I/O ERROR
```

If this message occurs before the dialog has been completed, the dialog automatically restarts.

If an I/O error occurs after the dialog is completed, the TAPE READY message is printed.

ACTION:

1. Make sure that the system device is on-line and write-enabled and that the unit number is set correctly.
2. Respond appropriately to the question which follows the message.

Turning Off the System

If power failure detection is available on the EduSystem 30, simply turn the console key lock to OFF. Otherwise, to shut the system down, overnight or for any reason, ensure that the system is inactive by:

1. Typing CTRL/C to stop any program that is running.
2. Turning the key lock to POWER, and pressing and raising the HALT switch.
3. Turning the key lock to OFF.

NOTE

Turning off the computer does not turn off the disk unit. Disk power should *never* be turned off.

Turning On the System

If power failure detection is available on the EduSystem 30, simply turn the key lock to PANEL LOCK. Otherwise, perform the following procedures.

1. Ensure that the hardware is properly initialized as explained previously.
2. Turn the key lock to POWER.
3. Set the SR to 0000 and press EXTD ADDR LOAD.
4. Set the SR to 7600 and press ADDR LOAD.
5. Turn the Teletype to LINE.
6. Turn the key lock to PANEL LOCK.

EduSystem 30 is now ready to process BASIC programs.

Restarting EduSystem 30

If EduSystem 30 has been loaded onto the system device but the computer memory has been used to run other programs (which do not write on the system device), then the following device-dependent programs may be used to start up the system. Instructions for these programs are given in octal numbers. If unfamiliar with the octal, or base 8, number system, refer to *Introduction to Programming 1972*.

DF32 OR RF08 DISK

With hardware bootstrap:

1. Press and raise the HALT switch.
2. Set the SR to 5350.
3. Press and raise the SW switch.

Without hardware bootstrap:

1. Press and raise the HALT switch.

2. Set the SR to 7750 and press ADDR LOAD.
3. Set the SR to 7600 and lift DEP.
4. Set the SR to 6603 and lift DEP.
5. Set the SR to 6622 and lift DEP.
6. Set the SR to 5352 and lift DEP.
7. Set the SR to 5752 and lift DEP.
8. Set the SR to 7750 and press ADDR LOAD.
9. Press the CLEAR switch, then the CONT switch.

TC01 DECTAPE

With hardware bootstrap:

1. Press and raise the HALT switch.
2. Set the SR to 0600.
3. Press and raise the SW switch.

Without hardware bootstrap:

1. Press and raise the HALT switch.
2. Set the SR to 7742 and press ADDR LOAD.
3. Set the SR to 1353 and lift DEP.
4. Set the SR to 6766 and lift DEP.
5. Set the SR to 6771 and lift DEP.
6. Set the SR to 5344 and lift DEP.
7. Set the SR to 1352 and lift DEP.
8. Set the SR to 5343 and lift DEP.
9. Set the SR to 7752 and press ADDR LOAD.
10. Set the SR to 0220 and lift DEP.
11. Set the SR to 0600 and lift DEP.
12. Set the SR to 7577 and lift DEP, then lift DEP again.
13. Set the SR to 7742 and press ADDR LOAD.
14. Press the CLEAR switch, then the CONT switch.

TD8E DECTAPE

With TD8E ROM:

1. Press and raise the HALT switch.
2. Set the SR to 7470.
3. Press, in order, the ADDR LOAD, EXTD ADDR LOAD, CLEAR, and CONT switches.

Without TD8E ROM:

1. Press and raise the HALT switch.

2. Set the SR to 0011.
3. Press the EXTD ADDR LOAD switch.
4. Set the SR to 7470 and press ADDR LOAD.
5. Press the CLEAR switch, then the CONT switch.

Using Optional Hardware

LP08 LINE PRINTER

If the EduSystem 30 includes an LP08 line printer, perform the following procedures to produce output on the line printer rather than at the Teletype.

1. Turn the LP08 power switch to ON.
2. Turn the ON-LINE/OFF-LINE switch to ON-LINE.
3. Type LPT and press RETURN.
4. When printing is complete, type TTY and press RETURN.
5. Turn the LP08 power switch to OFF.

An LPT command causes all subsequent output associated with LIST, LOG, RUN, CATALOG, BATCH, or STACK commands to be printed on the LP08 printer. A TTY command returns control to the Teletype from the line printer. The user may temporarily interrupt printing by turning the ON-LINE/OFF-LINE switch to OFF-LINE; he may continue printing by turning it to ON-LINE. When batch processing terminates, the output device is automatically reset back to the Teletype.

HIGH-SPEED PAPER TAPE READER/PUNCH

If a high-speed punch is available, the PUNCH command may be used to punch the current program on paper tape. PUNCH may be used with or without an argument. It is similar to LIST in that the entire program is punched if no argument is used and, if an argument is used, only those lines numbered greater than or equal to that argument are punched. Perform the following procedures to use the high-speed punch:

1. Turn the TTY control knob to LINE.
2. Turn the high-speed punch ON.
3. Press FEED to produce some leader tape.
4. Type PUNCH and press RETURN.
5. When punching is complete, press FEED to produce some trailer tape.
6. Turn the high-speed punch OFF.

Punching may be stopped at any time by typing CTRL/C.

If a high-speed reader is available, the TAPE command may be used to load BASIC programs previously punched on tape. Perform the following procedures to use the high-speed reader:

1. Turn the high-speed reader ON.
2. Turn the TTY control knob to LINE.
3. Insert tape in the reader.
4. Type TAPE and press RETURN.
5. When the tape has read in, turn the high-speed reader OFF.
6. Type KEY; press RETURN.

A KEY command returns control to the Teletype.

PUNCHED CARD INPUT

Programs loaded and run via the card reader may be prepared using a standard DEC 029 keypunch or be marked with a lead pencil and the EduSystem 30 template. If punched cards are used, the system building dialog questions must be answered appropriately.

Punch program statements on cards just as if they were being typed at the interactive terminal. The seven control commands (NEW, RUN, LIST, OLD, KEY, MSG, and OPR) available in batch mode are used by punching the command name starting in column 1. Information normally marked starting in column 6 must begin in column 5.

The number of data columns on each card which is read by EduSystem 30 is specified in the system building dialog. Thus the system can be made to ignore the latter portion of a batch input card. These ignored columns then may be used to hold sequence numbers or other information which will not be processed by EduSystem 30.

Programs punched with a keypunch may be read by either the regular DEC punched card reader (CR8 series) or the DEC mark-sense reader (CM8 series). With the punched card reader, standard 80-column cards may contain up to 80 columns of data. Cards to be read by the mark-sense reader must be punched on cards which have the special marks along the bottom edge of the card and which conform to other rigid specifications.

The mark-sense cards may be made to contain up to forty

columns of data per card. The simplest, and recommended, method for using punched mark-sense cards involves using the standard EduSystem 30 cards. Ignore the information printed on the card. Prepare a keypunch drum card which will cause the punches to occur only in keypunch columns 1, 3, 7, 11, 13, . . . , that is in all the odd numbered columns *except* column 9. All other columns should be automatically skipped. The user may now punch EduSystem 30 cards as if they were normal cards. The drum card will correctly position the punches. Note that the BASIC commands must be punched in their entirety.

CALCULATING AVAILABLE STORAGE

The number of storage blocks initially available on the system device is calculated according to the following rules:

1.

System Device	Available Blocks
DF32 Disk	131
DECtape	1348
RF08 Disk	1922
LINCtape	1348

2. If batch capabilities are included in the system, subtract 10 blocks from the basic number of blocks. If the number of blocks specified for use by the batch log is greater than one, subtract one less than that number from the current total.
3. If the system device is a disk and the hardware configuration includes more than a single disk surface, add the appropriate number of blocks for each additional disk surface.

System Device	Available Blocks
DS32	255
RS08	2046

chapter 8

edusystem 40

INTRODUCTION

EduSystem 40 offers a combination of the language capabilities of EduSystem 20 and EduSystem 30 by allowing the user to run either system, but not both simultaneously. The purchaser of an EduSystem 40 is obtaining an extremely versatile system—one which can accommodate not only a large and varied number of users, but a tremendous number of program runs per day. Working with EduSystem 20, the beginning student, who is unfamiliar with a computer system and who may be learning a computer language for the first time, can develop his skills while as many as seven other students may also be using the system at equal or more advanced levels of learning. If the problem is complex or the student desires the use of commands not present in EduSystem 20, then EduSystem 30 is available as a stand-alone system. In addition, using the batch capabilities of EduSystem 30 permits extremely large numbers of user programs on cards to be executed every day.

With the varied services provided by EduSystem 30 and, alternately, the time-sharing capabilities of EduSystem 20, the user obtains a complete system which can be put to use continuously for maximum productivity.

System Components

EduSystem 40 is composed of a PDP-8/E computer with 8192 words of core memory (12,288 words of core memory are recommended if the system includes 5 or more terminals); automatic loader (MI8-EF or MI8-EG hardware bootstrap, MI8-EC for TD8-E DECtape with ROM); 32,768 word DECdisk (DF32 or RF08) or DECtape; optical mark sense card reader; and as many as eight Teletype terminals with low-speed paper tape reader and punch units. Each EduSystem 40 contains two BASIC Language processors (EduSystem 40-20 and EduSystem 40-30) and a library of sample programs, textbooks and curriculum guides.

Optional components include a high-speed line printer, high-speed paper tape reader and punch, and additional core memory to increase program size.

System Expansion

By adding 4096 words of core memory; a high-speed paper tape reader; 262,000 word DECdisk and control; additional computer terminals and their associated interfacing; and an EduSystem 50 software set, EduSystem 40 may be expanded to EduSystem 50. (Chapter 9 provides information concerning the EduSystem 50 TSS-8 time-sharing system.)

If the system includes DECtape, the PDP-8 Operating System (OS/8) may be run. This system contains a keyboard monitor, machine language assemblers, debugging tools, and FORTRAN, and is described in Chapter 9 of *Introduction To Programming 1972*.

BASIC LANGUAGE CAPABILITIES

Advantages and Applications

EduSystem 40 provides the user with two BASIC systems—EduSystem 40-20 and EduSystem 40-30. Both systems are fairly compatible in language capabilities—they share a large number of common statements and commands. However, there are certain features peculiar to each EduSystem which may make it advantageous for the user to choose one over the other. A short summary of these features is included here. (The user is directed to Chapters 5 and 7 for complete details concerning the use of the language capabilities of EduSystems 20 and 30).

EDUSYSTEM 20

EduSystem 20 provides an immediate mode for fast and accurate calculations of expressions and equations. Statements used in this mode are not stored in memory.

There are certain commands available in EduSystem 20 which are either not present in EduSystem 30 or are used in a different way. The more important of these are EDIT, INPUT, DIM, IF THEN, ON GOTO, and ON GOSUB.

The EDIT command speeds the editing procedure by eliminating the need to retype complete lines. The user simply searches for the character(s) he wishes to change and, using the options available, makes the necessary corrections.

The INPUT statement allows a number or numbers to be entered from the Teletype as values for variables. Using EduSystem 20, the user response may be a value, or a mathematical expression which may contain arithmetic operations and a BASIC function (see Tables 8-4 and 8-5).

The DIM statement is not necessary in EduSystem 20 as the system sets limits on subscript size (single subscript: 0-2047; double subscript: 0-63 for each subscript), and defines all variables as they occur.

The IF THEN statement, in addition to conditionally altering the order of execution by effecting transfer of control, may include another BASIC statement, thus causing an operation to be performed without changing the order of execution.

The ON GOTO and ON GOSUB statements allow conditional transfer to another statement or subroutine depending upon the integer value of the formula following ON. After execution of a subroutine, control returns to the statement following the ON GOSUB statement.

It has already been mentioned that EduSystem 20 is a multiuser BASIC, and as far as the school system is concerned, this is probably its greatest asset. As many as eight users may be working simultaneously on one computer. Terminals may be situated in different rooms, even different buildings, to serve a greater number of users and applications.

EDUSYSTEM 30

EduSystem 30 is a true batch system allowing either interactive or "hands-off" batch operation. Specially formatted mark cards provide a means by which students can easily code programs and submit a deck for later execution. An operator can gather all individual program decks into one large deck, to be run consecutively and automatically by the system. Commands are available which allow operator intervention during runs.

EduSystem 30 makes available to the user a single mass storage device—DECdisk (or DECtape). The user can store data files or sections of very long programs which will later be chained together. He can also save programs which may be used later or which may be called more than once. Several special commands called Privilege Control Commands are associated with the mass storage device. Among them are commands which will save and

delete programs, determine the amount of available core, list programs in storage, and list the number of blocks in use.

EduSystem 30 includes a CHAIN command which makes program chaining possible and allows the running of programs of any length. Each section of the entire program must be 6000 characters or less and is stored on the mass storage device. Data files may be used in a similiar manner by allowing temporary storage of data on the mass storage device. Using WRITE statements, data may be written onto files; data stored in these files is called into a program via READ statements.

EduSystem 30 includes a character-handling feature which allows words or characters to be entered into a BASIC program in response to questions. The special characters # and $ are used to facilitate this procedure. Characters are stored as their respective numeric codes and therefore may be used and manipulated with standard BASIC commands. The ASCII codes for these characters may be found in Appendix B.

For detailed information concerning the features mentioned in this section, and all other available language capabilities, the user should refer to Chapter 5 when using EduSystem 20 and Chapter 7 when using EduSystem 30.

Language Summaries and Error Message Summaries follow.

LANGUAGE SUMMARIES

BASIC Statements and Commands

Table 8-1 lists the statements available in EduSystems 20 and 30; Table 8-2 summarizes the editing and control commands. These tables are abbreviated forms of those contained in Chapters 5 and 7.

Table 8-1. Statements

Statement & Format	Description	EduSystem 20	EduSystem 30
Input/Output			
DATA $n_1, n_2, \ldots n_n$	Numbers n_1 through n_n equal variables in a READ	x	x
INPUT $v_1, v_2, \ldots v_n$	Get v_1 through v_n input from TTY	x	x

Table 8-1. (Cont.) Statements

Statement & Format	Description	EduSystem 20	EduSystem 30
PRINT $e_1, e_2, \ldots e_n$	Print values of specified text, variables or expressions; also used for format control	x	x
READ $v_1, v_2, \ldots v_n$	Read variables v_1 through v_n from DATA list	x	x
RESTORE	Reset DATA pointer to beginning value	x	x
WRITE $n_1, n_2, \ldots n_n$	Record DATA n_1 through n_n on mass storage file		x
Transfer of Control			
GO TO n	Transfer control to line number n	x	x
IF e_1 r e_2 GO TO n	If relationship r between e_1 and e_2 is true, transfer control to line number n		x
If e_1 r e_2 THEN n	Same as IF GO TO; under EduSystem 20 n may be a statement	x	x
ON e_1 GO TO n_1, n_2, n_n	Conditionally change order of program execution according to evaluation of formula e_1 (if integer of e_1=1 transfer control to n_1; if integer of e_1=2, transfer to n_2, etc.)	x	
ON e_1 GOSUB n_1, n_2, n_3	Conditionally change order of program execution according to evaluation of formula e_1 (if integer of e_1=1 transfer control to subroutine n_1; if integer of e_1=2 transfer to subroutine n_2; if integer of e_1=3 transfer to subroutine n_3)	x	
Loops and Subscripts			
DIM $v(n_1)$, $v(n_2, n_3)$	Define subscripted variables	x	x
FOR v=e_1 TO e_2 STEP e_3	Set up program loop; define v values beginning at e_1 to e_2 incremented by e_3	x	x
NEXT v	Terminate program loop; increment value of v until v>e_2 (in FOR statement)	x	x

Table 8.1 (Cont.) Statements

Statement & Format	Description	EduSystem 20	EduSystem 30
Subroutines			
GOSUB n	Enter subroutine at line number n	x	x
RETURN	Return from subroutine to statement following GOSUB	x	x
STOP	Transfer control to END statement	x	x
Others			
CHAIN 'n' $	Link to next section of a program which is stored within the system as file n		x
DEF FNA (x)=f(x)	Define a function	x	x
DEF FNA (x, y)= f(x, y)			
END	End of a program	x	x
LET v=f	Assign value of formula f to v	x	x
RANDOMIZE	Randomizes random number routine	x	x
REM text	Insert a remark or comment	x	x
NOLINE	Suppress printing of line numbers in which program logic errors are found		x

Table 8-2. Edit and Control Commands

Command & Format	Description	EduSystem 20	EduSystem 30
CAT	List names of programs in storage area		x
CTRL/C	Stop program execution: return to edit phase	x	x
DEL n	Delete line n	x	
n	Delete line n	x	x
DEL n, m	Delete lines n through m inclusive	x	
EDI n c	Search line n for character c typed following carriage return	x	
KEY	Return to keyboard mode after TAPE (EduSystem 30-HSR only)	x	x

Table 8-2. (Cont.) Edit and Control Commands

Command & Format	Description	EduSystem 20	EduSystem 30
LIST	List entire program in core (If low-speed punch is on, EduSystem 20 will cause a tape to be punched)	x	x
LIST n	List program starting at line n		x
LIST n	List line n only	x	
LIST n, m	List lines n through m inclusive	x	
LISTNH	List program; no header (If low-speed punch is on, program will be punched on paper tape)		x
LISTNH n	List program beginning at line n; no header		x
NEW	Clear core; request program name	x	x
OLD	Clear core; bring program to core from storage area		x
RUN	Compile and run program in core	x	x
RUN NH	Same as RUN; no header		x
SCR	Erase current program from core	x	x
BYE	Same as SCR	x	
TAPE	Read paper tape from low-speed reader; suppress printing on TTY (also used with high-speed reader under EduSystem 30)	x	x
BATCH	Commence batch processing		x
*BATCH n	Same as BATCH; limits runs to n per program		x
ECHO	Switch from typeout to non-typeout mode or vice versa when using low-speed reader		x
LPT	Print output on lineprinter, if available		x
LENGTH	Request number of blocks to store current program		x
NAME	Same as NEW but does not delete existing program		x
PUNCH	Punch entire program on paper tape		x
PUNCH n	Punch program starting at line n		x
PTP	Punch a program out on the high-speed paper tape punch.	x	
PTR	Read program from high-speed reader	x	
RESEQUENCE	Renumber program lines		x
TTY	Print output on TTY		x

Table 8-2. (Cont.) Edit and Control Commands

Command & Format	Description	EduSystem 20	EduSystem 30
PRIVILEGE (password)	Enable use of privileged commands Insert password, no echo		x
*HEADER (header)	Change header; type new header (max. 12 characters) for next batch run		x
*LOG	Print system log		x
*MAX n	Set instruction limit n times 200 per program for next batch run		x
*PASSWORD (new password)	Change password Type new password, no echo		x
*SAVE	Save program in storage area		x
*STACK	Start unattended batch operation		x
*STACK n	Same as STACK; limit runs per program		x
*UNSAVE	Delete program from storage area		x

* Privileged Mode Command

Batch Control Cards

The control cards in Table 8-3 are used when running an EduSystem 30 batch card deck. The user must mark the appropriate box in the upper left hand corner of the EduSystem 30 mark card. Each control card is explained in detail in Chapter 7.

Table 8-3. Batch Control Cards

Control Card	Description
NEW	Indicates a new program; contains the new program name; must be first card in the deck—any cards appearing before this card are ignored.
OLD	Indicates an old program; contains the name of the program to be called; must be first card in the deck—any cards appearing before this card are ignored.
MSG	Columns 6-39 contain a message which is printed as part of program output.
OPR	Message marked in columns 6-39; when encountered in program execution rings bell to alert operator and prints message on TTY.

Table 8-3. (Cont.) Batch Control Cards

Control Card	Description
KEY	Temporarily halts the batch operation and turns control over to the console terminal.
LIST	Instructs computer to print a listing of the program on the TTY. Should be placed at the end of the card deck before the RUN card.
RUN	Normally the last card in the deck; instructs the computer to begin execution.

BASIC Functions and Arithmetic Operations

Table 8-4 lists the functions available in EduSystem 20 and 30. Table 8-5 is true for all EduSystems.

Table 8-4. Functions

Function	Description	EduSystem 20	EduSystem 30
SQR(x)	Square root of x ($\sqrt{x}$)	x	x
SIN(x)	Sine of x (x in radians)	x	x
COS(x)	Cosine of x (x in radians)	x	x
TAN(x)	Tangent of x (x in radians)	x	x
ATN(x)	Arctangent of x (x in radians) (result in radians)		
EXP(x)	e^x (e=2.712818)	x	x
LOG(x)	Natural log of x ($\log_e x$)	x	x
ABS(x)	Absolute value of x ($\|x\|$)	x	x
INT(x)	Greatest integer of x	x	x
SGN(x)	Sign of x (+1 if positive, −1 if negative, 0 if zero)	x	x
RND(x)	Random number between 0 and 1	x	x
FIX(x)	Truncates decimal portion of x	x	
TAB(x)	Controls printing head position on TTY.	x	x
CHR$(x)	Converts character code to character. Used only with PRINT command.	x	x

Table 8-5. Arithmetic Operations

Symbols	Meaning
↑ or ^	exponentiation
*	multiplication
/	division
+	addition
−	subtraction

Order of Execution

1. parenthetical expressions
2. exponentiation
3. multiplication and division
4. addition and subtraction

ERROR MESSAGE SUMMARIES

EduSystem 20

EduSystem 20 checks all commands before execution. If an error is found, it prints one of the error messages in Table 8-6 and the number of the line in which the error occurred.

Table 8-6. EduSystem 20 Error Messages

Message	Explanation
WHAT?	Command not understood. It does not begin with a line number and is not a valid system command.
ERROR 1	Log of negative or zero number was requested.
ERROR 2	Square root of negative number was requested.
ERROR 3	Division by zero was requested.
ERROR 4	Overflow—exponent greater than approximately +38.
ERROR 5	Underflow—exponent less than approximately −38.
ERROR 6	Line too long or program too big.
ERROR 7	Characters are being typed too fast; use TAPE command for reading paper tapes.
ERROR 8	System overload caused character to be lost.
ERROR 9	Program too complex or too many variables. (GOSUB, FOR, or user-defined function calls are too deeply nested.)
ERROR 10	Missing or illegal operand or double operators.
ERROR 11	Missing operator before a left parenthesis.
ERROR 12	Missing or illegal number.
ERROR 13	Too many digits in number.
ERROR 14	No DEF for function call.

Table 8-6. (Cont.) ***EduSystem 20 Error Messages***

Message	Explanation
ERROR 15	Missing or mismatched parentheses or illegal dummy variable in DEF.
ERROR 16	Wrong number of arguments in DEF call.
ERROR 17	Illegal character in DEF expression.
ERROR 18	Missing or illegal variable.
ERROR 19	Single and double subscripted variables with the same name.
ERROR 20	Subscript out of range.
ERROR 21	No left parenthesis in function.
ERROR 22	Illegal user-defined function—not FN followed by a letter and a left parenthesis.
ERROR 23	Mismatched parentheses or missing operator after right parenthesis.
ERROR 24	Syntax error in GOTO.
ERROR 25	Syntax error in RESTORE.
ERROR 26	Syntax error in GOSUB.
ERROR 27	Syntax error in ON.
ERROR 28	Index out of range in ON.
ERROR 29	Syntax error in RETURN.
ERROR 30	RETURN without GOSUB.
ERROR 31	Missing left parenthesis in TAB function.
ERROR 32	Syntax error in PRINT.
ERROR 33	An unavailable device was requested by the user—the device is either not present in the system, or in use.
ERROR 34	Missing or illegal line number.
ERROR 35	Attempt to GOTO or GOSUB to a nonexistent line.
ERROR 36	Missing or illegal relation in IF.
ERROR 37	Syntax error in IF.
ERROR 38	Missing equal sign or improper variable left of the equal sign in LET or FOR.
ERROR 39	Subscripted index in FOR.
ERROR 40	Syntax error in FOR.
ERROR 41	FOR without NEXT.
ERROR 42	Syntax error in LET.
ERROR 43	Syntax error in NEXT.
ERROR 44	NEXT without FOR.
ERROR 45	Too much data typed or illegal character in DATA or the data typed in.
ERROR 46	Illegal character or function in INPUT or READ.
ERROR 47	Out of data.
ERROR 48	Unrecognized command during execution.

BATCH MODE PROGRAM LOADING ERRORS

EduSystem 30 also checks all commands before execution. If it is unable to execute a command, EduSystem 30 informs the user by printing one of the error messages in Table 8-7.

Table 8-7. Batch Mode Program Loading Errors

Message	Explanation
BAD CHARACTER	One or more characters on a card could not be understood. The character % replaces any unreadable character. The information on the card is processed if possible. This error *does not* cause any subsequent RUN commands to be ignored.
BAD CHARACTER—LINE NOT USED	Same as BAD CHARACTER message except that the information on the card is ignored. This error *does* cause any subsequent RUN commands to be ignored.
WHAT?	Line does not make sense to the system. It does not begin with a line number and is not a valid system command.
NO ROOM	The program is too big to be loaded, i.e. it is greater than 5000 characters. Larger programs should be run in interactive mode.
FILE NOT SAVED	The program named on an OLD control card is not available in the system storage area.
FILE TOO BIG FOR BATCH MODE	The program named on an OLD card is too big to be run in batch mode. It must be run interactively.

INTERACTIVE MODE PROGRAM LOADING ERRORS

As each line is typed, a check is made for program loading errors. If an error is found, one of the messages in Table 8-8 is printed immediately after the line containing the error.

Table 8-8. Interactive Mode Program Loading Errors

Message	Explanation
WHAT?	Line does not make sense to the system. It does not begin with a line number and is not a valid system command.
LINE NO. TOO BIG	The line number of a line or the argument of a system command is greater than 4095.
LINE TOO LONG	Line just entered is longer than 80 characters.
NO ROOM	There is no room to store the line just entered.
FILE NOT SAVED	The program named as the operand of an OLD command was not previously saved on the system device.
NO SPACE	There is not enough space to SAVE the current program (DECtape Systems).
Bell	If an invalid character is entered, the Teletype bell rings and the character is ignored.
I/O ERROR	An input or output error occurred on the DECtape unit. Be sure that the unit is on-line, write-enabled, and the unit number is set correctly. Retry whatever was interrupted by the error. If the problem persists, there is a hardware problem (DECtape Systems).
INVALID PASSWORD	The password typed after a PRIVILEGE command is not the system password. Privileged mode is not entered.

CODING ERRORS

After the RUN or RUNNH command, but before execution, EduSystem 30 checks for mistakes in the BASIC program coding. If an error is found, one of the following messages and the appropriate line number is printed.

Table 8-9. Coding Errors

Message	Explanation
CH	There is an illegal character in the line.
EN	Program does not have an END statement as the last line in the program.
FN	Not enough NEXT statements in the program. There must be a NEXT statement for each FOR statement in the program.
FO	FOR and NEXT statements do not match. There is a NEXT statement in the program whose variable is not the same as the variable in the corresponding FOR statement.
LI	Line contains an improperly written decimal number or constant. It may, for example, have two decimal points or have an alphabetic character in it.
M1 M2	The program as a whole is too big to be run by the system. Making the program smaller, reducing the size of subscripted variables, or using the NOLINE command may help.
NE	Program has too many (more than 8) nested FOR-NEXT loops.
PC	Line contains an improperly used parenthesis. Generally, the problem is an expression which does not have an equal number of left and right parentheses.
RO	Statement contains an invalid relational operator (<, =, >, <=, >=). Relational operators may only be used in IF statements.
S1 S2	Statement as a whole is not properly written and, as a result, does not conform to proper BASIC syntax. For example, a semicolon is allowed in a PRINT statement but not in a READ or INPUT statement.
ST	Statement command word is not one of the BASIC statement types.
TB	The program is too big to be run. Cause is usually an extremely large number of PRINT statements.
TO	Program is either too big or too complex to be run. The total number of variables, constants, functions, and line numbers should be reduced, if possible.

Table 8-9. (Cont.) Coding Errors

Message	Explanation
UL	A GOSUB, GOTO, or IF statement contains a line number which does not exist.
UQ	A quotation mark indicating the beginning of a string of text does not have a corresponding quotation mark at the end of the text string.

PROGRAM LOGIC ERRORS

If a program logic error occurs, EduSystem 20 prints one of the messages in Table 8-10 after execution.

Table 8-10. Program Logic Errors

Message	Explanation
CH	A CHAIN statement tried to chain to a program which was not available in the DECtape storage area.
CO	Program ran too long and was automatically stopped by the system.
DA	The program ran out of data. It attempted to do a READ after all data had been read.
D0	The program attempted to divide by zero. Instead of dividing by zero, BASIC divides by the smallest possible number, giving a result of about 10^{500}. This error does not cause the program to stop.
FN	An expression contains a function which was not defined in a DEF statement.
GS	The program is too complex to be executed. The problem is generally that too many subroutines have themselves executed GOSUB instructions.
LG	Program attempted to take the logarithm of a negative number or zero.
RE	A RETURN statement was used outside a subroutine or a subroutine was entered by a GOTO instead of a GOSUB.
SP	Same as GS.
SQ	Program attempted to take the square root of a negative number. BASIC automatically takes the square root of the absolute value of the number instead. This error does not cause the program to stop.

Table 8-10. (Cont.) Program Logic Errors

Message	Explanation
SS	Program used a subscript which was too big for the variable. The maximum size of a subscript is specified in a DIM statement.
WR	There is no more room to write data. The program attempted to execute a WRITE statement when the data list was full. (Note that if this error occurs, the program text will no longer be intact. A NEW, OLD, or SCRATCH command must be used to clear the program area.)

LOADING AND OPERATING INSTRUCTIONS

The software for EduSystem 40 is distributed on five paper tapes —EduSystem 40-20 and EduSystem 40-30 for Disk Systems, and individual EduSystem 20 and EduSystem 30 tapes for DECtape Systems. The user with a DECtape System should refer to Chapters 5 and 7 for loading and operating instructions as the following instructions refer only to the standard Disk System. The EduSystem 40-20 and 40-30 tapes are used to build the EduSystem 40 system on DECdisk (either DF32 or RF08) when the system is installed. The building procedure is outlined below.

Initializing the DECdisk

Initialize the DECdisk by ensuring that the disk unit is turned on and that all disk protect switches are in the off position.

Building EduSystem 40 on Disk

The system tapes may be loaded into memory by use of either the RIM Loader (refer to Appendix A) or with the automatic loader (hardware bootstrap MI8-EF or MI8-EG).

1. Turn the key lock on the computer console to POWER.
2. Turn the Teletype to LINE.
3. Place the EduSystem 40-20 tape in the appropriate reader (high or low-speed) with the leader code (ASCII 200) over the read head.
4. If the system contains a hardware bootstrap, set the SR = 5356; press and raise the HALT switch; press and raise the SW switch.

5. If using the RIM loader, set the SR = 7756; press and raise ADDR LOAD, CLEAR and CONT.
6. Turn the reader on.

The tape should begin to move (if it does not, be sure the correct tape is being used and repeat the above procedure). When the tape has been read completely, the system prints the following message on the Teletype:

```
IS SYSTEM DEVICE A DF32 DISK?
```

The user should respond Y or N. If the response is N, the system asks:

```
IS SYSTEM DEVICE AN RF08 DISK?
```

The answer here should be Y; otherwise the system will repeat both questions until the answer to one of them is Y. When the correct system device has been specified, the system prints:

```
EDUSYSTEM 20  BASIC

NUMBER OF USERS(1 TO 8)?
```

At this point halt the computer by pressing and raising the HALT switch. (If the above messages are not printed, the above procedure should be repeated before continuing the building process.)

7. Place the EduSystem 40-30 tape in the appropriate reader.
8. If the system contains a hardware bootstrap, set the SR = 5356; press and raise the SW switch.
9. Otherwise, set the SR = 0000; press EXTD ADDR LOAD; then set the SR = 7756; press ADDR LOAD, CLEAR and CONT.

Approximately one-third of the tape should be read before the standard EduSystem 30 dialogue begins. The system will print:

```
STANDARD SYSTEM?
```

Build EduSystem 30 as detailed in Chapter 7 and summarized below. The user is at point A on the flowchart in Figure 8-1 and should continue with the dialogue presented in the flowchart. (If Y is typed as a response to this question, the system assumes the responses contained in brackets on the flowchart.)

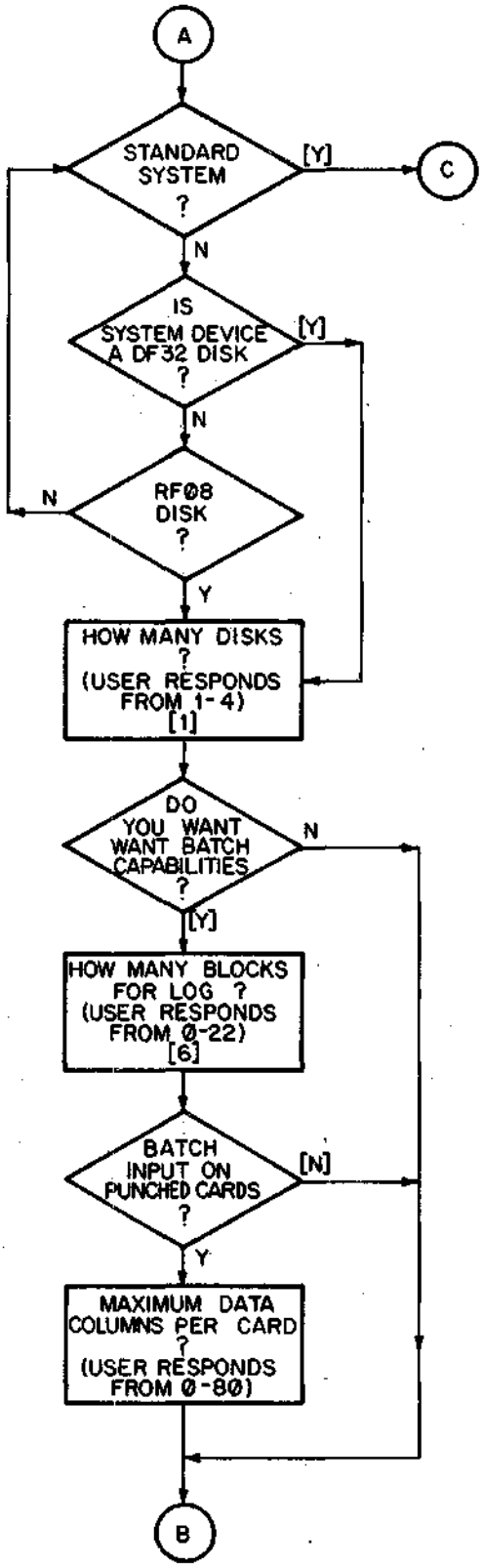


Figure 8-1. Building EduSystem 40

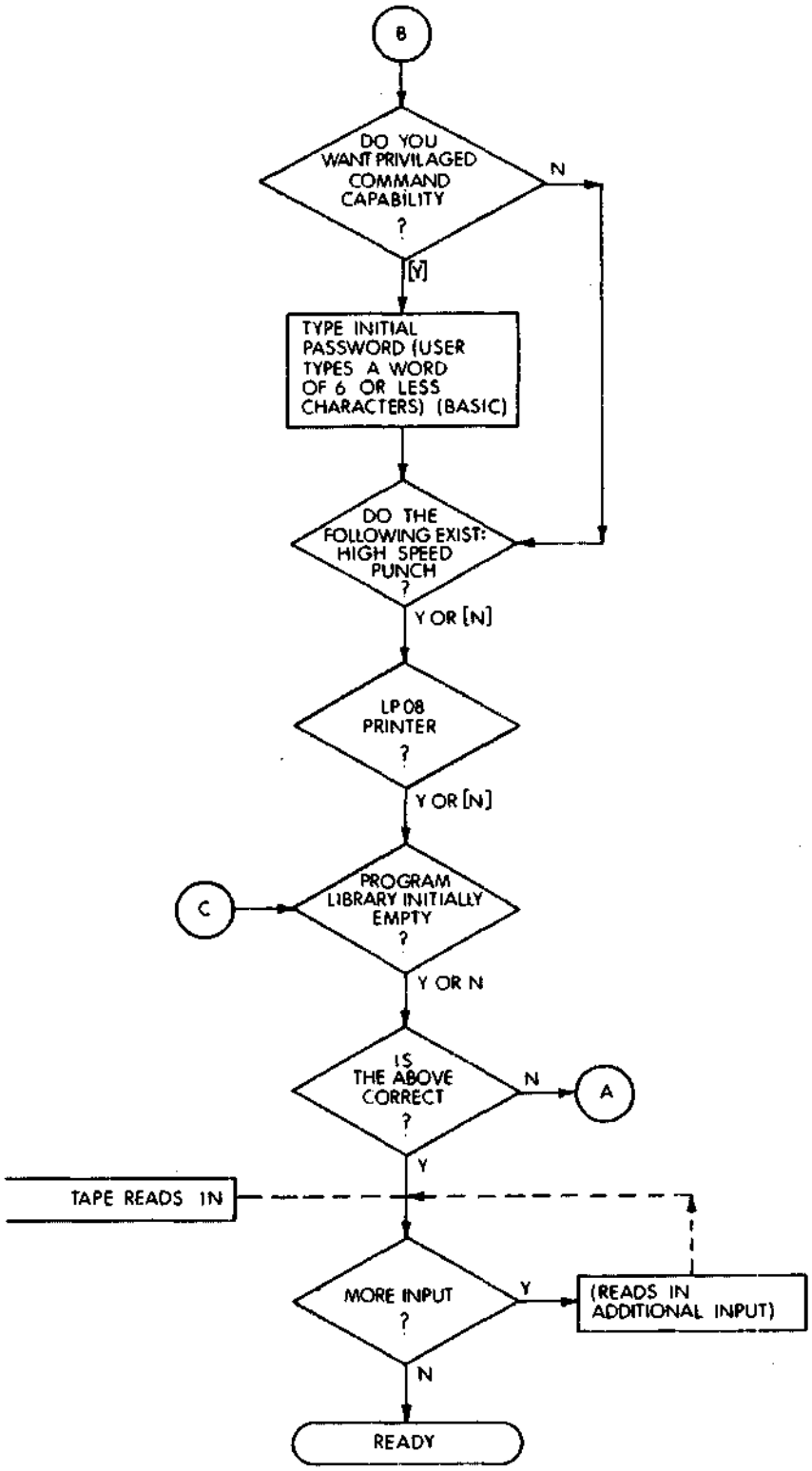


Figure 8-1. (Cont.) Building EduSystem 40

When EduSystem 40 prints READY on the Teletype, the disk is completely built and ready to be used, with EduSystem 30 running.

Starting EduSystem 40

Once EduSystem 40 has been built on the disk, it can be stopped and restarted by performing the following operations.

1. Press and raise the HALT switch.
2. A. If the system does not have a hardware bootstrap, the following instructions must be manually loaded:

   Set SR=7750 and press ADDR LOAD.
   Set SR=7600 and lift DEP.
   Set SR=6603 and lift DEP.
   Set SR=6622 and lift DEP.
   Set SR=5352 and lift DEP.
   Set SR=5752 and lift DEP.
   Set SR=7750 and press ADDR LOAD, CLEAR and CONT.

   B. If the system contains a hardware bootstrap set the SR= 5350 and press and raise the SW switch.

The system should respond by printing:

```
EDUSYSTEM 40-SELECT (1=20,2=30):
```

If the system does not respond in this manner, repeat the starting instructions; if the system does respond as it should, the user is at point A on the flowchart in Figure 8-2 and should continue with the dialogue in that flowchart.

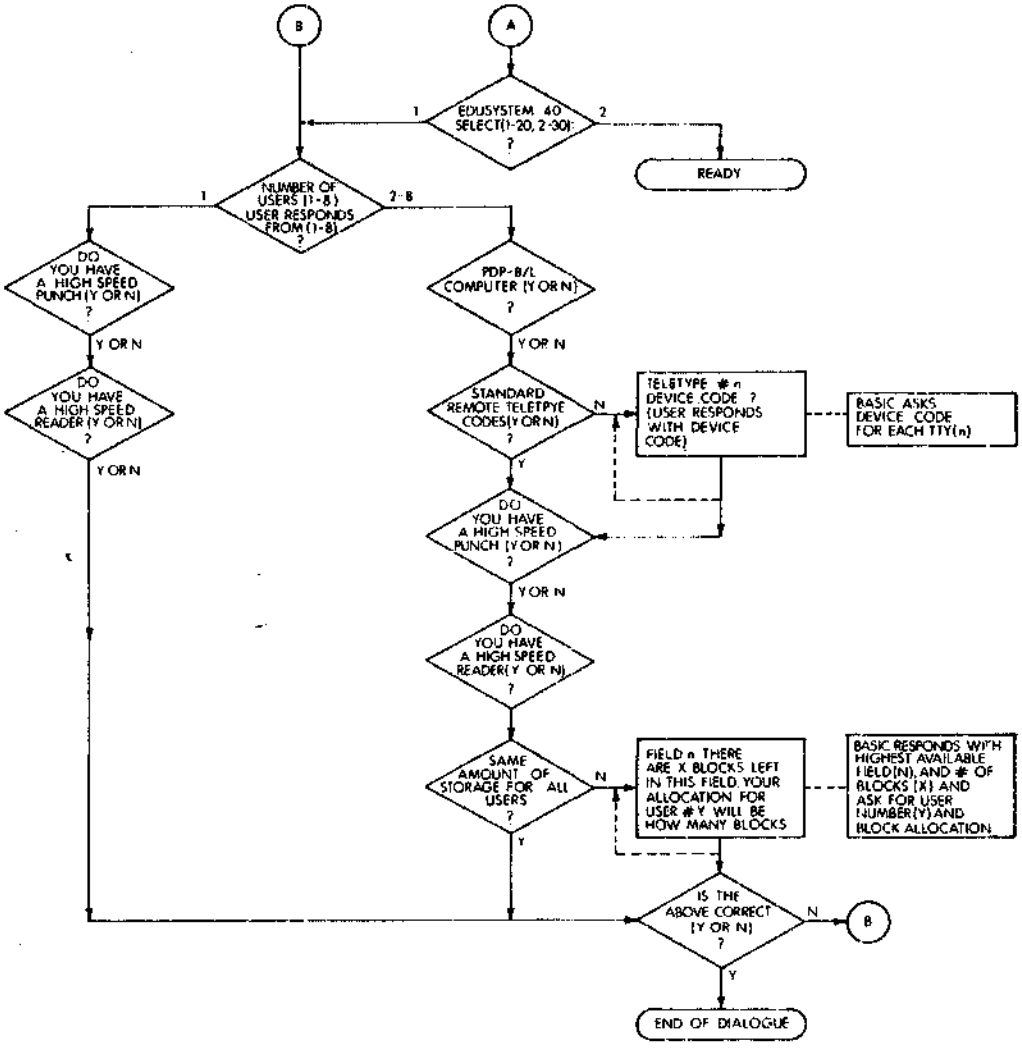


Figure 8-2. Starting Edusystem 40

EduSystem 40-20 or 40-30, whichever has been chosen, is now ready for use. To take either system down and start the other, the user need only repeat the starting instructions from step 1.

chapter 9
edusystem 50

INTRODUCTION

EduSystem 50 is a general purpose, time-sharing system for PDP-8 computers that offers up to 16 users (24 in certain applications) a comprehensive library of System Programs. These programs provide facilities for editing, assembling, compiling, debugging, loading, saving, calling, and executing user programs on-line. An extended BASIC language provides users with the ability to use strings, files, and program chaining. Two higher-level languages, FOCAL and FORTRAN, are also provided. All languages and utilities may be used simultaneously. One group of users may be working in BASIC while another is using assembly language. EduSystem 50 serves all levels of users simultaneously.

By separating the central processing operations from time-consuming interactions with human users, the computer can, in effect, work on a number of programs simultaneously. Cycling between programs and giving only a fraction of a second at a time to each program or task, the computer can deal with many users seemingly at once. The appearance is created that each user has the computer to himself. The execution of various programs is done without their interfering with each other and without lengthy delays in the response to individual users.

The heart of EduSystem 50 is a complex of subprograms called the Monitor. The Monitor coordinates the operations of the various programs and user consoles, ensuring that the user is always in contact with his program. The EduSystem 50 Monitor allocates the time and services of the computer to the various users; it grants a slice of processing time to each job, and schedules jobs in sequential order to make most efficient use of the system disk. The Monitor handles user requests for hardware operations (reader, punch, etc.), swaps (moves) programs between memory and disk, and manages the user's private files.

User Programs

When the user is working with EduSystem 50, it appears to him as though he had his own 4K (4096 word) PDP-8 computer. He then has the capability of doing anything which can be done in a 4K computer plus the capabilities of the Monitor. Several users can run different programs at virtually the same time because Monitor controls the scheduling of execution times. The Monitor brings a program into core from the disk, allows it to execute for a short time, and takes note of the state at which execution is stopped. The user is allotted a 4K block of core that contains his particular program; this 4K block is swapped (moved) from core onto a 4K area of disk when the Monitor needs to bring another user program into core to be executed.

After the user's program has been executed for a period of time, it is placed at the end of the queue (line) of user programs waiting to be run. If only one program is ready to run, it is allowed to do so without interruption until another program is ready. If a user wishes to maintain a permanent copy of his program, he can save a copy within the file area of the disk (an area separate from the swapping area) or on DECtape or paper tape.

User Files

A user is any person logged into EduSystem 50. Each user has an account number and password assigned to him by the System Manager. The account number and password allow the user to gain access to the computer. The account number is also used to identify whatever files the user may own within the EduSystem 50 file system.

The system disk is divided into logical areas called files. A user can store programs or data in files on the system disk. The user can further specify which users may access his files and for what purpose (read, write, or both). Parts of the disk are used to store system files, those programs which are accessible to anyone using the system. A major portion of this chapter deals with how to use the system files, generally called System Library Programs.

With the appropriate Monitor commands, the user can create new files and manipulate old files (extend, reduce, or delete them). These commands are summarized in Appendix C. Most individual System Library Programs are able to handle user files as input or

output with commands issued from the user's console. Such commands are described in this chapter under the section on the appropriate System Library Program.

System Configuration

Depending on the hardware configuration of a particular EduSystem 50, as many as 16 users may work on the system simultaneously. The standard configuration is designed to service 16 users and includes the following hardware components:

- PDP-8/E computer with 24,576 (24K) words of core memory, power failure protection, and an automatic loader (hardware bootstrap).
- High-speed paper tape reader.
- 262,000 word high-speed disk.
- Dual DECtape transports and controller.
- 16 computer terminals and their associated interfaces.

NOTE

EduSystem 50 is also available in an 8-user configuration with 16,384 (16K) words of core memory and without DECtape transports. All other system components and capabilities remain the same.

Software provided with the standard EduSystem 50 includes the following:

- General-purpose time-sharing Monitor system.
- Time-shared BASIC language processor.
- Time-shared assembly language package including text editor, symbolic assembler, loader, and utility debugging program.
- Time-shared FORTRAN-D and FOCAL language processors.
- System utility programs, such as PIP and DECtape COPY.
- Library of sample programs, textbooks, and curriculum guides.

EduSystem 50 may optionally include a high-speed line printer. Also optional are additional core memory, disk storage unts, or DECtape transports for added system storage capacity.

System Expansion

EduSystem 50 may be expanded to EduSystem 55 by adding a card reader to the configuration. EduSystem 55 includes all the features of EduSystem 50 and allows EduSystem 30 to be run as one of the time-shared jobs. (See Chapter 7 for a detailed explanation of EduSystem 30.)

EDUSYSTEM 50 MONITOR

EduSystem 50 Monitor controls the allocation and use of hardware resources. Many of these functions of the Monitor are invisible, and of no concern to the user, for example, the way it allows many users to run programs on a single computer. In other instances, the user explicitly tells the Monitor what he would like to do by typing one or more of the Monitor commands described in this chapter.

The Monitor commands described in the first half of this section are those needed to log into the system, to utilize the System Library Programs, and to log out of the system. All users must be familiar with these commands. The commands described in the last half of this section are not needed to run System Library Programs such as BASIC or FOCAL but are frequently useful. The Advanced Monitor commands described at the end of this chapter are primarily useful for creating assembly language programs and files.

NOTE

All Monitor commands must be terminated by typing the RETURN key. All words within a Monitor command line are separated by one or more spaces.

Calling the Monitor

The user enters commands to system programs, such as BASIC and FOCAL, in exactly the same way that he enters commands to the Monitor (i.e., by typing them at the keyboard); therefore, the system must have some way of distinguishing between the two cases. It does so by defining two modes of console operation: Monitor mode and program mode. When a user's console is in Monitor mode, all input is interpreted as being commands to the Monitor. Otherwise, all input is assumed to be to the user program or system program which is being run by the user.

A special character, CTRL/B (obtained by striking B with the

CTRL key held down and echoed on the Teletype as ↑B), is used to unconditionally place the user's console in the Monitor mode. Typing CTRL/B tells the system that the command to follow is a Monitor command, regardless of the current console mode. Generally, the command which follows the CTRL/B will be the S command.

↑B	Return to Monitor mode.
↑B↑BS	Return to Monitor mode from a program which is printing. (The two CTRL/B's stop the printout, allowing the S command to be typed.)

It is not necessary to precede each Monitor command with CTRL/B. Once in the Monitor mode, a console stays in that mode until a command is entered to start a system program. To signify that the console is in the Monitor mode, the system prints a dot (.) at the left margin of the console printer paper. This dot indicates that the characters entered next are to be treated as a Monitor command. Thus, the CTRL/B capability is important when a user is running a program and wishes to issue a Monitor command. He may, for example, be using one language (or system program) and want to change to another, as shown below:

```
.R FOCAL

SHALL I RETAIN LOG, EXP, ATN ?:NO

SHALL I RETAIN SINE, COSINE ?:NO

PROCEED.

*TYPE 6+10-3-1
=    12.0000*
*TYPE 25+5*2+5
=    40.0000*

*↑BS
.

.R BASIC

NEW OR OLD--NEW
NEW PROGRAM NAME--
```

Notice that the Monitor responds to BS by printing a dot at the left-hand margin.

Logging into EduSystem 50

To prevent unauthorized usage and to allow the Monitor to maintain a record of system usage, EduSystem 50 requires that each user identify himself to the system before using it. Before attempting to log into the system, the user should ensure that the console LINE/OFF/LOCAL knob is set to LINE and then press the RETURN key. If the console is connected to EduSystem 50 and is not already in use, the Monitor rolls the console paper up two lines and prints a dot at the left margin of the paper. The dot indicates that the system is in Monitor mode and that the Monitor is waiting for a command. The LOGIN command allows the user to gain access to EduSystem 50.

The used types LOGIN followed by an account number and password. Providing the console is free (not already logged in), the command, account number, and password are not printed on the console paper as the keys are typed. If the command name letters are being printed, stop typing the command; instead, strike the RETURN key, log out using the LOGOUT command (see Logging out of EduSystem 50). At this point, a successful LOGIN can be accomplished. The LOGIN command is formatted as shown below:

```
.LOGIN 1234 ABCD          (only the dot is printed)
```

The dot (.) is printed by the Monitor, LOGIN is the command name, 1234 represents the user account number, and ABCD represents the password.

NOTE

A command word and each parameter (except the last) is always followed by a space. Command lines are always terminated with the RETURN key. The RETURN key enters the full command line to the system.

When a user types something other than a valid LOGIN command, the Monitor responds in one of the following ways:

System Response	Explanation
. HELLO?	(user typed HELLO)
. LOGIN PLEASE?	(user typed ASSIGN D 3)
. ILLEGAL REQUEST	(user typed LOGIN ABCD ABCD)
.LOGIN 4771 DEMO ALREADY LOGGED IN	(user typed valid LOGIN on an already logged in console)
. UNAUTHORIZED ACCOUNT	(user typed an incorrect account number or password)

In the first example, HELLO is not a command, so it is repeated with a question mark by the Monitor. In the second example, ASSIGN D 3 is a valid command but is not appropriate until the user is logged into the system. In the third example, the Monitor finds that the LOGIN command is improperly formatted (the first parameter must be a 1- to 4-digit number); the console printout tells the user that he has made an ILLEGAL REQUEST. When the console is already logged in and the user types the LOGIN command, the characters typed echo at the console and the Monitor informs the user that the console is occupied with the message ALREADY LOGGED IN.

If the user attempts to use an incorrect account number or password, the Monitor replies UNAUTHORIZED ACCOUNT. Thus the Monitor can distinguish an invalid command from a valid command; it can also distinguish whether the valid command is appropriate when issued, whether the command is properly formatted, and whether the account number and password are acceptable. In all the preceding examples, Monitor ignores the command and prints another dot.

When the Monitor finds the LOGIN command properly formatted and the account number and password acceptable, it responds by identifying the version of the system being used, the job number assigned to the user, the number of the console being

used, and the time-of-day in hours, minutes, and seconds. This information is usually followed by a note from the System Manager concerning the system. For example:

```
TSS/8.22B    JOB 01    K00      17:05:40

YOU ARE NOW LOGGED INTO THE BHS EDUSYSTEM 50.
PROCEED AT YOUR OWN SPEED.
.
```

The Monitor then prints another dot and waits for the user to issue the next command. The job number assigned is an internal number by which the system identifies each on-line user; the user need not remember this number.

Logging out of EduSystem 50

The LOGOUT command indicates to the Monitor that the user is finished and ready to leave his terminal. When Monitor receives a LOGOUT command, it disconnects the user terminal from the system and records the amount of computer time used during the session and the total real time of the session. It also notes any user files deleted or saved. For example:

```
.LOGOUT

JOB 01, USER (1234,ABCD) LOGGED OFF K00 AT 19:20:19  ON 02-28-73
DELETED  1 FILES  ( 1. DISK BLOCKS)
SAVED 11 FILES  (  .38. DISK BLOCKS)
RUNTIME 00:00:24 (   7.  CPU UNITS)
ELAPSED TIME 02:14:39
PLEASE TURN OFF YOUR TTY.
```

Computer processing time used in this example was 24 seconds, while the elapsed time between LOGIN and LOGOUT was 2 hours, 14 minutes, and 39 seconds.

When typing the LOGOUT command, the user may follow it with a colon and an option to initiate some action by the system. These options and their functions are described in Table 9-1. To specify an option, the user types, for example:

```
.LOGOUT:K
```

If no option is specified, the S option is assumed; similarly if a user is simultaneously logged in at two (or more) consoles, no files will be deleted until he logs off his last job.

```
.LOCOUT:?

TYPE ↑C TO ABORT LOG-OUT; OR
TYPE ONE OF THE FOLLOWING (AND CAR RET):

K  TO KILL JOB AND DELETE ALL UNPROTECTED FILES;
L  TO LIST YOUR DISK DIRECTORY;
S  TO SAVE ALL (NON-TEMPOARY) FILES; OR
I TO INDIVIDUALLY SAVE AND DELETE FILES AS FOLLOWS:

        AFTER EACH FILE NAME IS LISTED, TYPE:
        P  TO SAVE AND PROTECT,
        S  TO SAVE WITHOUT PROTECTING, OR
        CAR RET ONLY TO DELETE.

CONFIRM: L
FIE    .BIN       <17>     1. BLOCKS
BAS00 .TMP       <17>     1. BLOCKS
BAS100.TMP       <17>     1. BLOCKS
INTER .BAS       <17>     1. BLOCKS
PROG   .FCL      <12>     2. BLOCKS

CONFIRM: I
FIE    .BIN       <17>     1. BLOCKS   :S
BAS00  .TMP       <17>     1. BLOCKS   : DELETED
INTER .TMP       <17>     1. BLOCKS   : DELETED
INTER .BAS       <17>     1. BLOCKS   :
PROG   .FCL      <12>     2. BLOCKS   : S

JOB  1, USER [3,13] LOGGED OFF K00 AT 10:46:07 ON  9 JUN 70
DELETED   3 FILES (   3. DISK BLOCKS)
SAVED  2 FILES (   3. DISK BLOCKS)
RUNTIME 00:00:25 ( 2. CPU UNITS)
ELAPSED TIME 00:06:12
```

In the previous example, the user typed a question mark to check the LOGOUT options. When LOGOUT completed the printed explanation, it printed CONFIRM: and waited for a user reply. In this case, the user requested a listing of his files. LOG-OUT followed this listing with a second CONFIRM: to which the user replied I. When using the I option, the user is advised not to type his reply to individual entries until printing stops. DELETED is printed automatically by the system to show that the temporary files are deleted without user intervention. The user saved binary file FIE and the FOCAL file PROG. The BASIC file INTER was deleted by typing the RETURN key.

Table 9-1. LOGOUT Options

Option	Function
:S	Save all nontemporary files. A temporary file is one of the following: BAS0nn BAS1nn TEMPnn where nn is the console number at which the user is logged into the system. A temporary file is created by a System Library Program and listed in CATALOG listings. A temporary file is also considered to be any file with a .TMP extension. If no option is specified in the LOGOUT command, :S is the default.
:K	Delete all unprotected files from disk.
:L	List the user's file directory. After listing the files, the system prints CONFIRM: and the user replies with one of the options.
:I	Allow the user to individually decide which files to save or delete. Temporary files are deleted automatically.
:?	Print a listing of the available options and their functions.

An optional method of logging out of the system is to type K in response to the Monitor dot or K followed by a colon and an option designation. For example:

```
.K

JOB 01, USER (1234,ABCD) LOGGED OFF KI3 AT 14:28:02 ON 02-28-73
SAVED  3 FILES (  .4. DISK BLOCKS)
RUNTIME 00:00:07 (  1. CPU UNITS)
ELAPSED TIME 00:13:57
```

System Library Program Control

Once logged into the system, the user can call any EduSystem 50 System Library Program. To call a library program, the user

types the command R (meaning run) followed by one or more spaces and the program name. For example:

```
.R BASIC

NEW OR OLD--
```

The Monitor fetches the BASIC language processor from the System Library and starts it. BASIC begins its dialog by asking if the user wishes to work on a new program or retrieve an old one from disk storage. Notice that once BASIC begins, the console is no longer in Monitor mode. Dots are no longer printed at the margin. All input is now processed by the BASIC language processor.

If the user types a program name which cannot be found in the System Library, the Monitor responds with an error message and returns the console to the Monitor mode, as follows:

```
.R BASICK
FILE NOT FOUND?
.
```

The exact contents of a System Library may vary from installation to installation. The System Manager may choose to make any number of programs available to all users.

Communication with Other Users

Although EduSystem 50 gives each system user the impression that she is the only user of the system, it is actually supporting many users at a time. Often it is useful to communicate with another user or with the system operator; this is done with the TALK command. The TALK command requests the Monitor to print a message on another system terminal. For example, a user at terminal 7 can ask the system operator to turn on the high-speed punch by typing the following command (the initial dot is printed by the Monitor):

```
.TALK Ø PLEASE TURN ON THE HIGH-SPEED PUNCH.
```

The above command causes the following to be printed at console 0.

```
**KØ7** PLEASE TURN ON THE HIGH-SPEED PUNCH.
```

K07 indicates that terminal 7 sent the message. Any terminal can initiate a message to any other terminal. However, if the destination terminal is printing at that time, the message will not be sent. The initiating terminal would, in this case, receive the message BUSY as a response.

System Status Reports

The command SYSTAT initiates a printout of the full status of EduSystem 50, how many users are on-line, what they are doing, etc. The command SYSTAT is equivalent to typing R SYSTAT. The format of the status report is described in the section on Utility Programs.

The user can obtain information on the amount of computer time used by him, the amount used by another user, or obtain the time of day with the TIME command. The TIME command can be issued in one of the following three forms:

TIME	Returns the elapsed processor time of the user issuing the TIME command since he logged into the system.
TIME 0	Returns the time of day.
TIME n	Returns the amount of processor time used by job n since logging into the system.

For example:

```
.TIME
00:00:00
.TIME 0
17:21:06
.TIME 10
00:00:34
```

Resource Sharing

All system users, when logged into the system, have access to the System Library, disk storage, a virtual 4K PDP-8, and the EduSystem 50 Monitor. The Monitor handles disk resource requests automatically. The Monitor also maintains a pool of available devices which are assigned to users upon request on a first-come, first-served basis. Devices such as the high-speed paper-

tape reader cannot, by their very nature, be assigned to several programs simultaneously. Therefore, the Monitor grants individual users exclusive access to these devices when needed. Devices such as the system disk, and sometimes DECtapes, are not assigned since they can be used by more than one user simultaneously.

All systems include a high-speed paper-tape reader in the pool of available devices. Many systems also include a high-speed paper-tape punch, high-speed line printer, and one or more DECtapes. These assignable devices are normally used with the System Library Programs PIP and COPY to store programs or data on paper tape or as DECtape files.

When a device is assignable (present on the system) and available (not being used), the ASSIGN command may be used to reserve the desired unit or units for exclusive use by the console issuing the command. The valid ASSIGN commands are formatted as shown below:

ASSIGN R	Assign the high-speed paper-tape reader.
ASSIGN P	Assign the high-speed paper-tape punch and line printer.
ASSIGN D	Assign a DECtape unit.
ASSIGN L	Assign the line printer and high-speed punch.

If other devices are assignable, the System Manager will inform the user of the appropriate device designator. The following is an example of using an invalid device designator:

```
.ASSIGN X
ILLEGAL REQUEST
.
```

The Monitor ignores the request, responds with the appropriate message, and prints another dot. When a valid ASSIGN command is issued, the Monitor checks the availability of the device and responds accordingly. For example:

```
.ASSIGN R
R ASSIGNED
.ASSIGN P
JOB 02 HAS P
```

When the system contains multiple units of a device, the user simply specifies the device; the Monitor assigns an available unit and responds with the unit number. For example:

```
.ASSIGN D
D Ø ASSIGNED
```

If all DECtape units are busy, the Monitor prints the message shown below:

```
.ASSIGN D
DEVICE NOT AVAILABLE
```

A specific unit can be requested by leaving a space between the device designator and the device number. For example:

```
.ASSIGN D 4
D 4 ASSIGNED
```

NOTE

If the user assigns a device with a nonexistent device number, that device will not be assigned; an error message does not result because that device is not busy. An error message only results when the device is already assigned.

The ASSIGN command assigns only one device at a time. Therefore, when multiple devices are to be assigned, each must be assigned separately. The following will not accomplish the desired assignments, either with or without the illegal commas.

```
.ASSIGN R, D 2, D 1
R ASSIGNED
```

The Monitor accepted the first device designator and ignored the rest of the command. If device R is unavailable, the Monitor prints

the appropriate message. The following commands complete the desired assignments (assuming available devices):

```
.ASSIGN D 2
D 2 ASSIGNED
.ASSIGN D 1
D 1 ASSIGNED
.
```

When the user has finished working with an assigned device, he should use the RELEASE command to terminate the assignment and allow other users access to the device. (When a user logs out of the system, any devices still assigned to him are automatically released.) A particular device is released when the user enters the RELEASE command, a space, and the device designator (and unit number if required), as shown below:

```
.RELEASE R
.RELEASE D 3
```

In the previous example, the high-speed reader and DECtape unit 3 are released. The Monitor prints a dot on the next line if the release is accomplished; otherwise, it prints a message. If, for example, a request is made to release a device which has not been assigned to the issuing console, the following happens:

```
.RELEASE D 2
ILLEGAL REQUEST
.
```

The Monitor printed ILLEGAL REQUEST after it checked and found that the specified device was not assigned to the console issuing the command.

NOTE

All commands must be formatted properly; ILLEGAL REQUEST is printed if the user fails to separate the device designator and unit number with a space.

When multiple device units were reserved by a user, each must be individually released. For example:

```
.RELEASE D 1
.RELEASE D 2
.RELEASE R
.
```

The Monitor does not perform checking when releasing a device as it does when assigning a device. The user may have two device units (e.g., two DECtape units) assigned and Monitor would not know which to release; therefore, device numbers are necessary with a RELEASE command. When only one unit of a specific device (one high-speed reader or punch, etc.) is on the system, the device designator alone is sufficient.

Error Messages

An appropriate error message is printed whenever: a Monitor command cannot be performed at the time it is requsted, a typing error is made, or the command is illegal (or nonexistent). Following each error message, the Monitor ignores the command and prints another dot, after which the user can issue another command. Table 9-2 is a list of the Monitor error messages.

Table 9-2. Monitor Error Messages

Message	Explanation
S1	The System Interpreter does not understand the command. S1 = command.
LOGIN PLEASE?	The user attempted to use a console which is not logged into the system.
UNAUTHORIZED ACCOUNT	The user attempted to log into the system with an invalid account number or password.
ALREADY LOGGED IN?	The user tried to log in on a console which is already in use.
FULL	The system is full. Another user cannot log in until one of the present users logs out.

Table 9-2 (Cont.). Monitor Error Messages

Message	Explanation
TYPE ↑BS FIRST	The user attempted to use a system command which cannot presently be honored due to the status of the user's program. The message may appear even after the user has typed ↑BS, since his program may continue until the I/O in progress at the time of the halt is completed. The user should wait a few seconds and then type his command a second time.
ILLEGAL REQUEST	The user requested an illegal command. This error usually results when some parameter has been given an incorrect value or the request refers to a facility not owned by the user.
BUSY	The user attempted to talk to a console which is currently printing or on which another user is typing.

SYSTEM LIBRARY PROGRAMS

The System Library contains a comprehensive set of user programs for a wide range of applications. Language processors, such as BASIC and FOCAL, allow the user to code and run programs in interactive languages. FORTRAN-D compiles and executes programs written in FORTRAN language. A complete assembly language system allows programs to be written in PAL-D, assembled, and run. Various utility programs perform special functions. The DEC-supplied System Library consists of the following programs.

- BASIC—an easily learned interactive language originally developed at Dartmouth College.
- FOCAL—DEC's own interactive language for on-line problem solving, designed especially for use on minicomputers.

- FORTRAN-D—a modified version of FORTRAN II.
- EDIT—a line-oriented text editor, used to create and modify source programs (such as FORTRAN) and data files.
- PAL-D—a 2-pass symbolic assembler.
- LOADER—a binary loader used to load assembled programs for execution.
- ODT—Octal Debugging Technique for testing and modifying assembly language programs.
- PIP—Peripheral Interchange Program for transferring files between the system disk and paper tape.
- COPY—a utility program used to transfer files between the system disk and DECtape.
- CAT—used to list all the files which a user has stored in his library.
- SYSTAT—(System Status) a utility program that prints a brief description of the system status.

A more detailed description of each of the above System Library Programs is presented in the following sections.

General File Characteristics

A fundamental feature of the Monitor is its ability to save programs or other data for each user in his own private library. These individual user libraries are maintained on the system disk. Individual entries in the library are called files, whether they contain programs or data. Within the library itself, there is no distinction between types of files by their contents. Each file is identified with a file name by which it is known and called into use.

The user does not directly create and update the files in his library. He uses the System Library Programs for this purpose. For example, he can use the SAVE command in BASIC. The SAVE command takes the BASIC program named and saves it as a file in the user's library for future use. Similarly, EDIT can be used to modify an existing file, resulting in the creation of a new file. Therefore, although the Monitor provides the actual file storage capability, most file manipulation is done while System Library Programs are being run.

The System Library Programs which operate on these files must know which file to use, when to create a new file, and what to call it. Each Library Program has its own method of determining

whether a user wishes to use an old file or create a new one; this is explained in the sections on individual library programs.

Example 1:

```
.R BASIC

NEW OR OLD--OLD
OLD PROGRAM NAME--PRIME
READY
```

Example 2:

```
.R FORT

 INPUT:TYPE
 OUTPUT:BTYPE
```

For most of his work, the user requires access to only his own library. However, it is often a useful feature to be able to obtain a program from another user's library, allowing a single file to be shared by several users. To access a program from another user's library, the user must tell the system in which individual library the file is stored. The user tells the system by entering the account number of the library's owner. (In the absence of an account number, the user's own library is the assumed source.) To get a file from the System Library, type an asterisk immediately after the file name.

Example 1:

```
.R BASIC
NEW OR OLD--OLD
OLD PROGRAM NAME--HOSSR*
READY
```

Example 2:

```
.R PALD
INPUT:NOTPIP 5440
OUTPUT:BIN!
```

NOTE

Most examples in the discussions of individual System Library Programs use file names within the user's own library. The user is free (file protect permitting) to use files from other user's libraries.

Access to another user's files is gained only with his permission. A user may "protect" his files against other users, i.e., prevent them from gaining access to his files, even though they know his program name and account number. Library Programs never permit a user to write in another user's files. Specifying a file which is protected, or specifying a nonexistent file, is an error that is detected immediately. An error message is printed and the file name is requested again.

The user places his output in a single file; however, it is often useful to input several files together. (For example, the user may wish to assemble two parts of a PAL-D program together.) To specify more than one input file, separate the file names by commas. No Library Program allows more than three input files. FORTRAN is limited to two; BASIC allows only one.

BASIC is a self-contained programming system, with an editor, compiler, and run-time system. It also has a distinctive file format. Files created by BASIC are not compatible with files created by other Library Programs. All other Library Programs depend on each other; therefore, all other Library Programs use the same format for their disk files. Consequently, files created by the Editor can be used as input to PAL-D or FORTRAN-D, and numerical files created with the use of the Editor can be read by FORTRAN programs as data files.

Up to this point, only files that exist within the time-sharing system, i.e., on the system disk, have been described; however, EduSystem 50 provides two other means of file storage: paper tape and DECtape. The Library Program PIP can be used to transfer files between paper tape and disk. The Library Program COPY allows files to be transferred between disk and DECtape.

Controlling the Execution of System Library Programs

EduSystem 50 provides the user with two options for stopping the system. CTRL/C (C with the CTRL key held down) allows the user to stop his BASIC program and return to the beginning of that program without returning to the Monitor. For example, if the user begins to run a BASIC program that has an endless loop, he can type CTRL/C to stop it. BASIC responds to ↑C with READY. All other Library Programs respond in a similar manner.

CTRL/B is used to stop the Library Program most recently

called. CTRL/B followed by S and the RETURN key unconditionally returns the user to the Monitor mode; the user can then call another Library Program. If the system is printing, two CTRL/B's and the S (↑B↑BS) are required to stop the system.

RUBOUT is another useful character that deletes the last typed character. Some Library Programs respond by printing \ or ← while others print the deleted character. If the RUBOUT key is typed while entering file names for input or output to a Library Program, RUBOUT deletes the whole line. The request for input or output is then repeated.

Returning to the Monitor

The user can stop the execution of a System Library Program at any time by typing CTRL/B followed by S and the RETURN key. The System Library Programs can also initiate a return to the Monitor. When the System Library Programs initiate a return, ↑BS is printed just as though the user had terminated the program. For example, BASIC returns to the Monitor when the user types the BYE command:

```
READY

BYE
↑BS
.
```

FORTRAN returns to the Monitor after completing execution of a program. CAT and SYSTAT return afer printing their particular data output. PAL-D returns after completion of an assembly, LOADER at the end of a normal load, and EDIT after completion of an EDIT. FOCAL, BASIC, ODT, PIP, and COPY never return to the Monitor; these programs must be terminated by the user with CTRL/B and S. Some System Library Programs return to the Monitor when a fatal error condition is detected.

BASIC

EduSystem 50 BASIC is a time-sharing version of the BASIC language. It allows even the beginning computer user to write and run meaningful programs. In addition, EduSystem 50 BASIC has advanced language features such as strings, files, and program chaining. This section describes the BASIC language capabilities not discussed in Chapter 1. Table 9-6 contains a complete summary of the EduSystem 50 BASIC language.

To call BASIC, the user types:

```
.R BASIC
```

After the user logs into EduSystem 50, and calls BASIC in the above manner, BASIC prints NEW OR OLD—. The user then types the appropriate adjective: NEW (if he wants to enter a new program) or OLD (if he wants to retrieve a program that was previously filed).

BASIC then asks NEW PROGRAM NAME— (or OLD PROGRAM NAME—) and the user types any combination of six letters or less. If the user is recalling an old program file from the disk, he must use exactly the same name as when he originally instructed BASIC to save it.

BASIC prints READY to signal the start of the editing phase; the user then begins to type the new program. If the user types a line consisting of only a line number followed by the RETURN key, that line is deleted. Each line must begin with a line number greater than 0 and less than 2047 and which contains no non-digit characters. To enter an entire line to the computer, the user must press the RETURN key.

If the user makes a typing error while typing a statement and notices it immediately, he can correct it by typing the RUBOUT key (right-hand side of the keyboard), or the back arrow key (SHIFT/O). Typing either key deletes the character in the preceding space and prints a backarrow (←) character for each character erased. The user can then type the correct characters. Typing the RUBOUT key a number of times erases one character from the current line (spaces are characters) to the left for each RUBOUT typed.

While BASIC is in the editing phase, certain additional commands (which must not have line numbers) are available. The

commands are described in Table 9-6 under Edit/Control Commands.

Truncation Function, FIX(X)

The truncation function returns the integer part of X. For example:

```
10 PRINT "FIX(10.2)=" FIX(10.2)
20 END
RUN
FIX(10.2)=10
```

FIX is like INT for positive arguments, and can be defined as:

$$FIX(X) = SGN(X) * INT(ABS(X))$$

ON GOTO Statement

The ON . . . GOTO statement may be used to provide a many-way branch. The general form of the ON . . . GOTO is:

On expression GOTO line number, line number . . . .

If the value of the integer part of the expression is 1, a GOTO is performed to the first statement. If the value of the integer part of the expression is 2, a GOTO to the second statement number is performed, etc. If the value is less than one, or greater than the number of statement numbers, the program terminates and an error message is printed. Examples of ON GOTO are shown below:

```
999 ON N GOTO 100,400,200,600,499

872 ON A+SQR(B*C) GOTO 100,200
```

SLEEP Statement

The SLEEP statement causes a BASIC program to pause for a specified interval, then continue running. SLEEP is followed by the number of seconds the program is to pause. For example:

```
222 SLEEP 30
```

or

```
220 LET N=15
222 SLEEP 2*N
```

causes a 30 second delay in the program.

The SLEEP statement is a useful way for a program to wait for a device (DECtape or line printer) which is busy. The ELSE clause in the OPEN statement can go to a routine which pauses for a while, then retries the OPEN. When the current user finishes with the device and releases it, the program may then proceed to OPEN and use it. This capability is especially useful when many users may be looking up information on a single DECtape file. It may also be used to allow two programs to communicate with each other. Each writes information on a tape file for the other, or others, to read.

SLEEP should always be used when waiting for a device. While the program is sleeping it is not using any processor time. A SLEEP time of 30 to 60 seconds is recommended. It is particularly important that the program not wait by repetitively retrying the OPEN. To do so wastes computer time and slows down other users. The integer part of the argument is used to determine the number of seconds to delay. This value must be between 0 and 4095.

Comments

An entire statement of comments may be included in the BASIC program by means of the REM statement. Often comments are easier to read if they are placed on the same line with an executable statement rather than in a separate REMARK statement. This can be accomplished by ending an executable statement with an apostrophe. Everything to the right of the apostrophe up to the statement terminator (carriage return or backslash) is ignored (unless the apostrophe occurs within a print literal or string constant.) For example:

```
10 LET X=Y 'THIS IS A COMMENT'
20 PRINT "BUT THIS IS NOT A COMMENT"
30 LET X$="A'B"
```

Thus, a comment is added to line 10 with an apostrophe, but in lines 20 and 30 the apostrophe is treated as a valid character.

Blank Lines

To make BASIC programs easier to read, blank lines can be inserted anywhere in a BASIC program. These can be used to break a program into logical sections, or (as is often done) to insert remarks with the apostrophe feature. For example:

```
10 'PROGRAM WRITTEN BY SAM JONES
100
```

Note that to insert a blank line, you must type one or more spaces after the line number; typing the line number alone will just delete that line from the program.

Multiple Statements per Line

As many statements as will fit may be typed on a single program line. Each statement must be separated by the backslash character "\" (SHIFT/L). The only statement requiring a line number is the initial one. For example:

```
10 FOR I=1 TO 10\PRINT I\NEXT I
```

Note that the backslash character acts as a statement terminator and thus cannot be included in a comment statement.

Editing BASIC Statements

If a program line is incorrect, it can be corrected by retyping it. Minor errors in statements can be corrected by using the EDIT command. The user types EDIT followed by the line number of the statement to be edited. BASIC responds by printing a left bracket ([). The user then types a search character. BASIC prints a close bracket and prints the statement through the first occurrence of the specified search character. The user may then:

1. Type new characters which are inserted at that point in the statement.
2. Type one or more back arrows (←) to delete characters to the left of the search character.
3. Type the ALT MODE key to delete the entire line up to that point (but not the line number).
4. Type CTRL/L to continue to the next occurrence of the search character.

5. Type CTRL/G to specify a new search character.
6. Type LINE FEED to finish the edit, keeping the remainder of the line unchanged.
7. Type RETURN to finish the edit, deleting the remainder of the line.

Saving Compiled Programs

BASIC compiles the current program each time it is run. If, however, a program will be used frequently without being changed, it may be stored in its compiled form. A compiled program can be retrieved and executed faster than a BASIC source program. To save a compiled program, the user types:

```
COMPILE FAME
```

The program is saved on the disk under the specified name (FAME). If a file by that name exists, BASIC prints DUPLICATE FILE NAME and does not compile that program.

Once a program has been compiled, it may be retrieved and run just like an ordinary BASIC source program. It may not, however, be listed, saved, or changed. If an attempt is made to do any of these things, the message EXECUTE ONLY is printed. The compile capability may therefore be used to protect programs from unauthorized listing or changing. Since only BASIC source programs can be edited, the user may wish to store both a source and a compiled version of a given program.

Compiled files are distinguished from regular BASIC programs by their file extensions. BASIC source programs have an extension of .BAS. Compiled files have an extension of .BAC. These extensions are printed along with the file name when a catalog is requested.

File Protection

EduSystem 50 permits a user to specify a protection code for each file. (See the section on Advanced Monitor Commands for a full description of protection codes.) The commands which write disk files (SAVE, REPLACE, COMPILE) also permit the user to specify what protection is to be given to a file. This is done

by following the file name with the protection code in angle brackets. For example:

```
SAVE DEMO <10>
```

will create and save a file named DEMO.BAS having a protection code of 10. When no protection is specified, a protection of 12 is automatically assumed.

Project-Programmer Numbers

In specifying the Account Number prior to requesting an OLD file, the user may optionally type a Project-Programmer number (giving the Account Number as two 2-digit numbers separated by commas instead of a single 4-digit number). In this way, the user may RUN files from another user's disk area. For example, both of the following are acceptable:

```
OLD PROGRAM NAME--FILE 13,3
```

where 13 is the Project Number and 3 is the Programmer Number, or:

```
OLD PROGRAM NAME--FILE 1303
```

where 1303 is the account number. The two file name indications are equivalent.

Restricted Accounts

As an added system protection, BASIC checks to see if an attempt is being made to run BASIC under Accounts 1 or 2. If so, BASIC prints the error message:

```
IMPROPER ACCOUNT #
ABORT
↑BS
```

thus preventing BASIC from interfering with the System Directories or the System Library.

Catalog Format

The CATALOG command prints file names and file extensions, file size, and file protection codes for the specified account (the account under which the user logged into the system). For example:

```
CATALOG

NAME        SIZE  PROT
TEMP00         1    12
BAS000.TMP     1    17
BAS100.TMP     1    17
IBOLD .BAC     1    12
DEMO  .BAS     1    10
```

Strings in BASIC

EduSystem 50 BASIC has the ability to manipulate alphabetic information (or strings). A string is a sequence of characters, each of which is a printing ASCII character (see Appendix B). EduSystem 50 strings consist of one to six characters; strings of more than six characters are truncated on input to six characters.

Variables can be introduced for simple strings, string arrays, and string matrices. A string variable is denoted by following the variable name with the dollar sign character ($). For example:

A1$	A simple string of up to six characters.
V$(7)	The seventh string in the array V$(n).
M$(1,1)	An element of a string matrix M$(n,m).

When string arrays or matrices are used, a DIM statement is required. For example:

```
10 DIM V$(10),M$(5,5)
```

reserves space for eleven 6-character strings for the array V$, and space for 36 6-character strings for the matrix M$.

READING STRING DATA

Strings of characters may be read into string variables from DATA statements. Each string data element is a string of one to six characters enclosed in quotation marks. The quotation marks are not part of the actual string. For example:

```
10 READ A$,B$,C$
200 DATA "JONES","SMITH","HOWE"
```

The string JONES is read into A$, SMITH into B$, and HOWE into C$. If the string contains more than six characters, the excess characters are ignored. The following program:

```
10 READ A$
20 PRINT A$
30 DATA "TIME-SHARING"
40 END
RUN
```

causes only

```
TIME-S
```

to be printed.

String and numeric elements may be intermixed in DATA statements. A READ operation always fetches the next element of the appropriate type. In the following example:

```
10 READ A,A$,B
20 DATA "YES",2.5,"NO",1
```

2.5 is read into A, YES into A$, and 1 into B.

The standard RESTORE statement (as described in Chapter 1) resets the data pointers for both string and numeric elements. Two special forms of the RESTORE command, RESTORE* and RESTORE$, may be used to reset just the numeric or string data list pointers, respectively. For example:

```
10 READ A,A$,B
20 DATA "YES",2.5,"NO",1
30 PRINT A,A$,B
40 RESTORE*
50 READ A,A$,B
60 PRINT A,A$,B
70 END

RUN
```

would print:

```
 .  2.5          YES             1
    2.5          NO              1
```

If line 40 were changed to RESTORE, this program would print:

```
2.5            YES            1
2.5            YES            1
```

since the numeric as well as string data lists would be reset.

PRINTING STRINGS

The BASIC PRINT statement may be used to print string information. If the semicolon character is used to separate string variables in a PRINT command, the strings are printed with no intervening spaces. For example, the program:

```
10 READ A$,B$,C$
20 PRINT C$;B$;A$
30 DATA ,"ING","SHAR","TIME-"
40 END
```

causes the following to be printed:

```
TIME-SHARING
```

INPUTTING STRINGS

String information may be entered into a BASIC program by means of the INPUT command. Strings typed at the keyboard may contain any of the standard ASCII characters on the user terminal except back arrow (←) and quotation mark ("). Back arrow is used in BASIC to delete the last character typed. Commas are used as terminators just as with numeric input. If a string contains a comma, the entire string must be enclosed in quotation marks. The following program demonstrates string input.

```
10 INPUT A$,B$,C$
20 PRINT C$,B$,A$
30 END
RUN

? JONES,SMITH,HOWE
HOWE           SMITH          JONES

READY
```

Strings and numeric information may be combined in the same INPUT statement as in the following example. Note that if an

input string contains more than six characters, only the first six are retained.

```
10 INPUT A,A$,B$
20 PRINT A$,B$,A
30 END
RUN
? 01754,MAYNARD, MASS.
MAYNAR          MASS.          1754
```

The numeric variable A is set to 1754 (leading zeros are deleted), the string MAYNAR is put in the string variable A$, and the string MASS. is put into the string variable B$. To print the number 01754, the number could be input and output as a character string.

LINE INPUT

Strings of more than six characters may be entered by means of the LINPUT (line input) statement. A LINPUT statement is followed by one or more string variables. For example:

```
10 LINPUT A$(1),A$(2),A$(3),A$(4),A$(5)
```

The first six characters to be typed are stored in the first string variable, the next six in the second, and so until the line of input is terminated by a carriage return.

Commas and quotes are treated as ordinary characters and hence are stored in the string variables. For example, if the following line were typed in response to the above LINPUT command:

```
?MAYNARD, MASS.  01754
```

then the values of the string variables would be as follows:

A$(1) = "MAYNAR"
A$(2) = "D, MAS"
A$(3) = "S. 017"
A$(4) = "54"
A$(6) = ""[1]

[1] Strings may consist of zero characters. Such a string is empty (or null). If printed, it causes nothing to be output. The null string is usually represented by a pair of quotes with nothing between (""). The null string should not be confused with a string of one or more spaces.

In the above example, the maximum number of characters which could be typed would be 30. Any additional characters would be ignored. In all cases, the maximum number of characters which may be typed in response to LINPUT is 50. If a longer line is typed, the message LINE TOO LONG is printed. The input line is ignored and must be reentered.

It is possible to mix numeric and string variables in a LINPUT statement, but this practice is not recommended. As an illustration of how this might be done, consider the example given earlier:

```
10 LINPUT A,A$,B$
```

where the user might type:

```
? 01754,MAYNARD, MA
```

This still sets the numeric variable A to 1754 (when used in LINPUT statements, numeric input remains unchanged). However, the string variable A$ would now be MAYNAR, and the string variable B$ would be D, MA.
When inputting strings with LINPUT, the error messages: MORE? and TOO MUCH INPUT, EXCESS IGNORED cannot occur.

WORKING WITH STRINGS

Strings may be used in both LET and IF statements. For example:

```
10 LET Y$="YES"
20 IF Z$="NO" THEN 100
```

The first statement stores the string YES in the string variable Y$. The second branches to statement 100 if Z$ contains the string NO. For two strings to be equal, they must contain the same characters in the same order and be the same length. In particular, trailing blanks are significant since they change the length of the string. "YES" is not equal to "YES ".

The relational operators < and > may also be used with string variables. When used with strings, these operators mean "earlier in alphabetic order" or "later in alphabetic order", respectively.

They may be used to alphabetize a list of strings, for example. The relation operators >=, <=, and <> may be used in a similar manner. The arithmetic operations (+, −, *, /, ↑) are not defined for strings. Thus, statements such as LET A$ = 3*5 and LET C$ = A$+B$ have no meaning, and should *not* be used in a BASIC program. They will not cause a diagnostic to be printed; however, the results of such operations are undefined.

THE CHANGE STATEMENT

The CHANGE statement may be used to access and alter individual characters within a string. Every string character has a numeric ASCII code (see Appendix B), a number which is used to indicate that particular character. The CHANGE statement converts a string into an array of numbers, or vice versa. The CHANGE statement has the form:

```
100 CHANGE A TO A$
```

or

```
100 CHANGE A$ TO A
```

where A$ is any string variable (or an element of a subscripted string variable) and A is an array variable with at least six elements. Any array variables used in CHANGE statements must have appeared in a DIM statement with a dimension of at least six.

The following program illustrates the use of the CHANGE statement by changing a string variable into an array of numbers.

```
10 DIM A(6)
20 READ A$
30 CHANGE A$ TO A
40 PRINT A(0);A(1);A(2);A(3);A(4);A(5);A(6)
50 DATA "ABCD)
50 DATA "ABCD"
60 END
RUN

 4   65   66   67   68   0   0
```

The CHANGE statement takes each character of the string and stores its corresponding numeric (ASCII) code in elements one to six of the array. Remaining array elements are set to zero. The

length of the string (0-6 characters) is stored in the zero element of the array. In the example above, the character codes for A, B, C, and D are stored in A(1) to A(4). A(5) and A(6) are set to zero. The number 4 is stored in A(0) since the string A$ is four characters long.

CHANGE may also be used to change an array of numeric codes into a character string as in the following program:

```
10 DIM A(6)
20 FOR I=0 TO 5
30 READ A(I)
40 NEXT I
50 CHANGE A TO A$
60 PRINT A$
70 DATA 5,69,68,85,53,48
80 END
RUN

EDU50
```

The length of the resulting string is determined by the zero element of the array. In the previous example, the string is five characters long. The elements of the array, starting at subscript 1, are assumed to be numeric character codes; these are converted to characters and are stored in the string. If any codes encountered are not valid character codes, or if an invalid string length is given, the message BAD VALUE IN CHANGE STATEMENT AT LINE n is printed, and execution is stopped.

A BASIC string of less than six characters always has the remaining character positions filled with zeros. For this reason, when such a string is changed to an array, the first six array elements are set to zero. The CHANGE statement always fills six array elements, even though the strings may not be six characters long. The user should be careful to dimension the array used in a CHANGE statement to at least six. If a string of characters is transformed into an array of less than six elements, an undetected error will occur.

The CHANGE statement is usable with strings not created by BASIC. It may, for example, be used to access files other than BASIC data files. Each string variable corresponds to three PDP-8 words. The CHANGE statement treats these three words as six 6-bit bytes, converts each 6-bit byte to its numeric character code equivalent and stores it in the corresponding array element. The

zero element of the array, the string length, is set equal to the number of bytes (characters) before the first zero byte. When reading unspecified data, there may be non-zero bytes following this zero byte. If so, they will be transferred to the array as well.

THE CHR$ FUNCTION

Occasionally, it is desirable to type a character other than those in the printing ASCII set, or to compute the value of a character to be printed. For this purpose, the CHR$ function can be used in a PRINT statement. The argument of the CHR$ function is sent as an ASCII character to the Teletype. For example:

```
10 FOR I=0 TO 9
20 PRINT CHR$(I+48);
30 NEXT I
40 END
```

prints 0123456789, since 48 to 57 are the ASCII values for the characters 0 to 9. The following special characters can also be printed using the CHR$ function:

Bell	CHR$(7)
Line feed	CHR$(10)
Carriage return	CHR$(13)
Quote (")	CHR$(34)
Back arrow (←)	CHR$(95)
Form feed	CHR$(12)

The Teletype will accept characters from 0 to 255 (decimal), many of which do nothing on most kinds of teletypes. Some of the special (non-printing) characters should not be used. For example, CHR$(4) causes a Dataphone to disconnect.

For each ASCII code there is a second acceptable form permitted in CHANGE and CHR$. The second code is obtained by adding 128 to the code given in the table in Appendix B. For example, CHR$ would type A in response to either 65 or 193 as an argument.

Program Chaining

Most programs are easily accommodated by EduSystem 50 BASIC. If a program becomes very long, however, it may be necessary to break it into several segments. Typically, programs of more than two to three hundred statements must be split into more

than one file. A program that has been broken into more than one piece is commonly called a chained program.

Each part of a chained program is saved on the disk as a separate file. The last statement of each part to be executed is a CHAIN statement specifying the name of the next part of the program. The next file is then loaded and executed. It may in turn chain to still another part of the program. The general form of the chain command is:

```
414 CHAIN "NAME"
```

or

```
414 CHAIN A$
```

where NAME is the name of the next segment to be executed (one to six characters enclosed in quotation marks). The name of the next segment may also be contained in a string variable. In either case, the file of that name is loaded and run. Thus, the statement:

```
999 CHAIN "SEG2"
```

is equivalent to:

```
OLD
OLD PROGRAM NAME--SEG2
RUN
```

except that it happens automatically. Each separate part of the program automatically links to the next part of the program chain.

The individual sections of a chained program may be either regular source files (.BAS) or compiled files (.BAC). If the sections are source files, they must be compiled before they are run. A chained program runs more efficiently if all its sections have been compiled. Source and compiled files cannot be mixed in program files.

If an error occurs while compiling or running a chained program, the name of the section containing the error is printed as

part of the error message. In all cases, whether a program terminates by an error or a STOP or END, BASIC returns to the first program in the chain. This is the one which is available for editing and rerunning when BASIC prints READY.

Most chained programs require that information from one section be passed to the next. The first section may, for example, accept input values and perform some preliminary calculations. The intermediate results must then be passed to the next section of the programs. This passing of values is done by means of data files which are explained in the next section. Whenever a CHAIN operation is performed, program data which has not been saved in a file is lost. Variable and array values are not automatically passed to the next program.

Disk Data Files

The standard BASIC language provides two ways of handling program data items. They may be stored within the program (in DATA statements) or they may be typed from the terminal. DATA statements, however, allow for only a limited amount of data. Also, the data is accessible only to the program in which it is embedded. Typing data from the terminal allows it to be entered into any program, but this is a time-consuming process. In either case, the data or results of calculations cannot be conveniently stored for future use. All these limitations may be overcome by the use of disk data files.

A data file is separate from the program or programs which use it. It is a file on the disk similar to a saved program, but it contains numbers or strings rather than program statements. This information may be read or written by a BASIC program. (Information in a data file is stored in a coded format; therefore, it cannot be listed by the BASIC Editor or EDIT.) (The maximum size of a data file is about 350,000 characters.) String and numeric information may be combined in a single data file. The number of data files a user may have is limited to about 100, space allowing. When a file is first created, its contents are undefined.

FILE RECORDS

A data file is made up of logical units called records. A record may be as small as a single numeric or string variable. More typically, it is a group of variables or arrays. The design of the program usually dictates the most efficient size of the record. If, for

example, the program manipulates a series of 5 by 5 matrices, each record could contain one such matrix. If the program operates on 80-character alphanumeric records, 14 string variables might comprise a record.

The size and composition of a record are defined with a RECORD statement. Like the DIM statement, RECORD is followed by a series of variables. They may, however, be unsubscripted as well as subscripted. For example:

```
10 RECORD A(5,5)
10 RECORD B$(14)
10 RECORD A,B,C$(8),D,E(5)
```

The set of variables mentioned in a RECORD statement, taken together, constitute a record. Each element within the record is a field. Numeric and string information may be mixed to comprise a more convenient record.

Variables mentioned in a RECORD statement should not appear in a DIM statement. The RECORD statement reserves variable space exactly as a DIM statements does. The difference is that the variables are also identified as being used for file input and output. Non-subscripted variables appearing in RECORD statements must not have been used previously in a program; therefore RECORD statements should always be the first statements in a program.

Records may be any length. A long record is typically more efficient since more information is transferred in a single operation. Records should, however, be only as long as necessary since excess variables lengthen the file. In particular, it is important to remember that all arrays and matrices have zero elements. The array A(5,5) has 36 elements, not 25. If A appears as part of a record, all 36 elements should be used.

It is also useful to try to make record sizes 43 variables long, or a multiple of 43. Each RECORD statement reserves program variable space in units of 43 whether or not the record is that big. Unless the record fills this area, some program variable space is wasted. It is not worthwhile, however, to make an inherently small record 43 variables long just to conform to this convention; this would make the file unnecessarily large.

OPENING A DISK FILE

Disk data files are completely separate from the programs which use them. Therefore, the program must specify which file or files it will use. The OPEN command is used for this purpose. OPENing a disk data file associates it with an internal file number, either 8 or 9. (A program may have two disk data files open at one time.) For example:

```
100 OPEN 9, "DATA10"
100 OPEN 8,A$
```

The name of the file to be opened may be explicitly stated in the OPEN command. If it is, it must be contained in quotation marks. The file name may also be contained in a string variable, allowing the program to decide which file to open, perhaps on the basis of input from the program's user. In either case, the name of the file is preceded by the internal file number, either 8 or 9. This argument may also be an expression whose value is either 8 or 9.

When a file is opened on an internal file number which has a file already open, the previously opened file is closed and the new file opened.

If no file of that name exists, the file is created. In either case, once the file is open, it is available for both reading and writing. BASIC disk data files are assigned an extension of .DAT which need not be specified as part of the file name in the OPEN statement.

READING/WRITING DISK FILES

Once open, files may be read and written, one record at a time, using the GET and PUT statements. GET statements read one record of information directly into the variable in the RECORD statement. PUT statements write the present values of the variables in the RECORD statement. Both GET and PUT statements are followed by the internal file number (8 or 9 or an expression), the line number of the RECORD statement containing the variables to be transferred, and the name of a control variable. For example:

```
100 RECORD A,B,C$(30),D(8)
110 OPEN 8,"FILE1"
120 LET I=0
130 GET 8,100,I
```

The control variable specifies the file record to be transferred. In the example above, FILE1 is opened as internal file 8. The value of the control variable, I, is zero. The GET statement in line 130 reads the first record (record 0) of FILE1 into A, B, and the arrays C$ and D. Single numeric values are read into A and B, 31 strings are read in C$, and 9 numeric values are read into D. After each transfer, whether it is a GET or a PUT, the value of the control variable is automatically incremented. Successive GET's or PUT's automatically proceed to the next record of the file.

The PUT statement has a similar format. For example, if line 130 of the preceding program had been:

```
130 PUT 8,100,I
```

the present values of A, B, C$, and D would have been written to the first record of FILE1.

File records may be accessed randomly by setting the control variable to the desired record number before doing the GET or PUT. Single records may be read, changed, and then written without processing the entire file. When reading a file, the record referenced in the GET statement must, of course, be the same as the record referenced in the PUT statement which wrote the data onto the file. The total length of the record and the relationship of string and numeric fields within the records used for the GET's and PUT's must be the same. If they are not, improper information will be read and written.

New files may be created by opening a file which does not already exist. As successive records are written onto the file, its length is extended as necessary. When a new file is created, it is useful to immediately write an end-of-file code in the last record. Writing the last record first forces the entire file to be allocated, making sure that enough disk space is available. It also provides an end-of-file mark. Programs which read this file may then check for this end-of-file mark to avoid reading past the end of the data file which results in an error.

Existing files may be enlarged by writing a new record farther out. If the program does not know how big the file will be, it may simply write records to the file in sequence.

The file will be automatically extended. When all the records have been written, one final end-of-file mark can be added.

In general, all records read or written on a specific file should be the same length, i.e., contain the same number of variables. However, if the user is careful he may intermix records of different lengths in a file. Suppose the following statement is executed:

```
40 PUT 8,100,N
```

and the value of N is n and the record specified by statement 100 is of length m. The PUT statement will write m variables in the file starting at the m*n variable. The simple rule for computing the first variable in the file to be accessed is the record length times the record number. (Remember the first record is record number zero.)

CLOSING/DELETING DISK FILES

When all work has been completed on a data file, it should be closed with a CLOSE statement. Once the file is closed, it may not be read or written unless it is reopened. The file does, however, remain on the disk and is available for future use. The CLOSE statement is followed by the internal file number to be closed (8 or 9). For example:

```
950 CLOSE 8
```

If the disk file was just created for temporary scratch use (to pass parameters during a CHAIN, for example), it should be deleted at the end of the program instead of closed. The UNSAVE statement is used to delete files. For example:

```
1000 UNSAVE 9
```

The file opened on internal file number 9 is deleted from the disk. Both CLOSE and UNSAVE may be followed by an expression equating to 8 or 9 instead of a constant.

Open disk data files are automatically closed at the end of the program, unless the program CHAINs to another program. In this case, all open files remain open and the new program may access them without executing an OPEN statement.

DECtape Data Files

Large permanent data files are best stored on DECtape rather than on disk. Each DECtape holds up to 380,000 characters of information. DECtape data files may be dismounted for safe-keeping, thereby insuring their privacy. Data files on DECtape are similar to files on disk except that they do NOT have filenames. Each reel of DECtape is treated as a discrete data file. When the tape is mounted on a DECtape drive, records may be read and written directly onto the tape.

A DECtape data file may be used by only one user at a time. Once a DECtape unit is assigned, a single user has exclusive access to it until he releases it. Each DECtape drive has a WRITE LOCK switch which physically prevents any write operations to that unit. If the WRITE LOCK switch is set, programs may not write on the tape even if the unit is assigned.

DECtape data files may be used in a variety of ways. Programs which need large data files should use DECtape to avoid consuming large disk areas. Administrative files, such as student or employee records, are best stored on DECtape. Since they are removable and can be write-locked when mounted, their use can be tightly controlled. DECtapes are also useful for information retrieval. A data tape may be kept permanently mounted but write-locked. Individual users may run programs which assign and query that file, then release it for others to use.

DECTAPE FILE RECORDS

Records for DECtape data files are specified the same way as for disk data files: with a RECORD statement. All rules for disk records apply to DECtape records. In fact, the same RECORD statement may be used for both a DECtape and disk file. (This is useful when transferring a tape file to a disk file for processing. Access to disk data files is considerably faster than to DECtape data files.)

It is possible to specify any record length for a DECtape data file, but a size of 43 variables is suggested, even more strongly than for disk data files. DECtapes are physically structured into blocks, each of which holds exactly 43 variables. If the record specified by the program is, for example, 44 variables, it requires two full blocks on the tape.

Records which are multiples of 43 variables are efficient in

utilizing DECtape space but are not efficient in speed. Such records are written in consecutive DECtape blocks. The tape unit cannot read or write consecutive blocks without stopping the tape and rewinding it slightly (rocking). This tape rocking also occurs when single block records (43 variables or less) are read or written as consecutive DECtape records. (In this case, each DECtape file record corresponds to a physical tape block.)

The most efficient way to utilize DECtape is to make records 43 variables in length and write them onto every tenth record in the file (records 0, 10, 20, etc.). When the entire length of the tape has been traversed (the last block of the tape is number 1473), write next into records 1, 11, 21, etc. In this way, every record is eventually filled. Programs which will be used repeatedly should access the tape in this manner.

OPENING A DECTAPE FILE

DECtape data files, like disk files, are completely separate from the programs which use them. Therefore, the program may specify which tape, or tapes, it will use. The OPEN statement is used for this purpose. Since DECtape files do not have names,[2] the OPEN statement specifies the DECtape unit number to be used. It is assumed that the proper tape reel has been mounted. If the file is to be updated, the unit should be write-enabled. If not, it should be write-locked. The OPEN statement is followed by the unit number to be used (0-7).

```
100 OPEN 2
100 OPEN 7
```

The unit number could be an expression. Making the unit number a variable is very useful since it is hard to predict which units will be available at the time the program is run. When the unit specification is a variable, the user may mount the file on any free unit, then INPUT the number into the program.

When the OPEN statement is executed, the indicated DECtape unit is automatically assigned to the user. It cannot subsequently be assigned to any other user. Thus, it is possible to try to open,

[2] It is important to note that BASIC data file DECtapes are not the same as the file-oriented DECtapes used by COPY. There is no directory on a BASIC DECtape file. Each tape is considered to be one file of data.

hence assign, a unit which is already assigned. If, in the above examples, units 2 and 7 were already assigned to the current user or any other user, the program would be terminated and an error message printed.

An alternative form of the OPEN statement allows the program itself to handle this situation. OPEN statements may include an ELSE clause which specifies a line number. If the OPEN statement fails, BASIC automatically performs a GOTO to this line number. For example:

```
100 OPEN 2 ELSE 900
```

If unit 2 is available, it is assigned and BASIC goes on to execute the next statement. If unit 2 is not available, statement 900 is executed next. Statement 900 could print a message and perhaps ask for an alternate unit number.

READING/WRITING DECTAPE FILES

DECtape data files are read and written using the same GET and PUT statements as are used for disk data files. The internal file number is a number between 0 and 7, or an expression. Unlike disk data files, DECtape data files are of a constant length equal to the capacity of the tape. The exact number of records per reel depends on the record size as follows:

Record Size	Tape Capacity
1-43 variables	1474 records
44-86 variables	737 records
87-129 variables	491 records

As indicated in the section on DECtape data records, a record size of 43 variables or less is recommended since it conforms to the physical blocking of the tapes themselves. It is also desirable to space the records along the tape so that the tape does not waste time rocking. The following subroutine could be used to write 1474 records on the tape in this fashion. It assumes that R is set to zero before it is called the first time and that the unit number is in U.

```
500 REM SUBROUTINE TO WRITE RECORDS ALONG TAPE
510 REM WRITES ONE RECORD EACH TIME CALLED
515 PUT U,10,R 'REMEMBER THIS INCREMENTS R
517 LET R=R+9 'SPACE OUT 10 BLOCKS
524 IF R<1474 THEN 550 'OK TO RETURN
530 IF R=1479 THEN 560 'TAPE IS FULL
540 LET R=R-1479
545 IF R>0 THEN 550
547 LET R=R+10
550 RETURN
560 STOP 'TAPE IS FULL
```

The following function may also be used to convert a logical record number (0 to 1469) to a physical record block spaced along the tape. This function does not use blocks 0-3. These blocks are, therefore, available for a header or label. Both the subroutine above and the function below assume a record length of 43 variables or less.

$$FNC(X) = (X-INT(X/147)*147)*10 + INT\ (X/147)+4$$

Once opened, any record on the tape may be read. The tape unit must, however, be write-enabled if it is to be written. Trying to PUT to a write-locked tape is an error.

CLOSING DECTAPE FILES

Once all work on a DECtape data file has been completed it may be closed. Closing a file releases the tape unit and makes it available to other users. Thus, if the tape contains important information (and especially if it is write-enabled) the CLOSE should not be done until the tape reel has been removed. If no CLOSE statement is encountered in the program, the unit remains assigned after the program has finished. The DECtape unit remains assigned until a Monitor RELEASE command is executed or the user logs out. An example of a CLOSE statement follows:

```
1100 CLOSE 6
```

USING DECTAPE DATA FILES WITH OS/8 FORTRAN

Numeric DECtape data files written by BASIC may be read by OS/8 FORTRAN with the FORTRAN RTAPE and WTAPE subroutines, and vice-versa. (String and Hollerith variables use different character codes.) Thus, it is possible to use BASIC to

prepare an input or update tape for a stand-alone FORTRAN program. This provides a convenient way to do big jobs in off-hours, without having to leave the time-sharing mode for very long.

Line Printer Output

If a line printer is available, it may be used both to list BASIC programs and to serve as an output device for the programs themselves. The line printer may only be used by one user at a time. The statements associated with line printer output are LLIST and LPRINT.

LLIST is similar to the LIST command except that the program listing is output to the line printer rather than to the Teletype. The LLIST command assumes that no other user has the line printer assigned and responds by typing WHAT? if the line printer is not available. After the listing is complete, the line printer is released and is available to any user.

BASIC programs may use the line printer as an output device during execution by means of the LPRINT statement. LPRINT is exactly like PRINT except that the information goes to the line printer rather than to the Teletype. All formatting conventions of the PRINT statement are available with LPRINT. In particular, CHR$(12) may be used to skip to the top of the next form (page).

The LPRINT statement also assumes that no other user has the line printer assigned. However, using this statement when the line printer is not available causes the program to terminate. Once LPRINT successfully assigns the line printer, it remains assigned until the program terminates.

The OPEN and CLOSE statements may be used to assign and release the line printer. An OPEN statement with a device number of 11 assigns the line printer, or if it is not available and an ELSE clause is specified, transfers control to the line number specified in the ELSE clause. CLOSE 11 releases the line printer.

Paper Tape Output

The high-speed paper tape punch may also be used as an output device. Like the line printer, the paper tape punch may only be used by one user at a time. OPEN and CLOSE statements with an internal file number of 10 will, respectively, assign and release the paper tape punch as shown in the following example:

```
10 OPEN 10 ELSE 100 'GOTO 100 IF PUNCH UNAVAILABLE
20 CLOSE 10
```

Here too, a GOTO statement in combination with an ELSE clause can be used to transfer program control if the paper tape punch is unavailable.

The LPRINT statement causes output to the paper tape punch when this device has been assigned. For example:

```
10 OPEN 10
20 LPRINT "THIS GOES TO PTP."
```

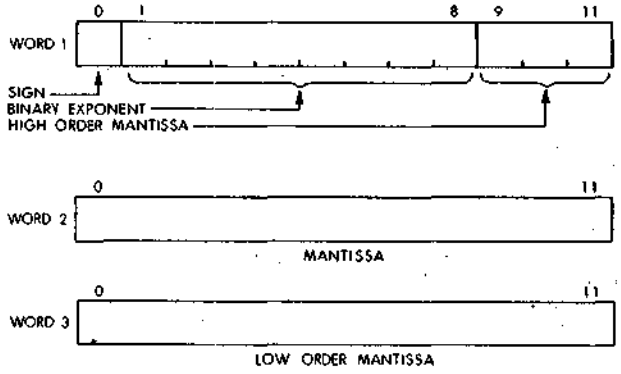
causes the statement THIS GOES TO PTP. to be punched onto paper tape. If the paper tape punch is not released with a CLOSE statement, it remains assigned after the program terminates.

Internal Data Codes

Using the file I/O capabilities and the CHANGE statement, it is possible to examine data which was written on a DECtape or disk file by a program other than BASIC. There are two data formats: numeric and string.

NUMERIC DATA

Each numeric value in BASIC is three PDP-8 words long. The storage format for numeric data is as follows:



A one in the sign bit means that the number is negative. The exponent is kept in excess 200 form where:

200_8 is 2^0
201_8 is 2^1
177_8 is 2^{-1}

The assumed decimal point is between bit 8 and bit 9 of word 1. *Also, the number is always normalized, meaning that bit 9 is always* 1 unless the number is zero. (Zero is represented by three zero words.) Note that this format is the same as the format used by FORTRAN-D.

Table 9-3. BASIC Internal Data Codes

6-Bit Byte Octal	Byte Decimal	ASCII Char.	6-Bit Byte Octal	Byte Decimal	ASCII Char.
00	0	NULL	40	32	?
01	1	SPACE	41	33	@
02	2	!	42	34	A
03	3	"	43	35	B
04	4	#	44	36	C
05	5	$	45	37	D
06	6	%	46	38	E
07	7	&	47	39	F
10	8	'	50	40	G
11	9	(	51	41	H
12	10	)	52	42	I
13	11	*	53	43	J
14	12	+	54	44	K
15	13	,	55	45	L
16	14	—	56	46	M
17	15	.	57	47	N
20	16	/	60	48	O
21	17	0	61	49	P
22	18	1	62	50	Q
23	19	2	63	51	R
24	20	3	64	52	S
25	21	4	65	53	T
26	22	5	66	54	U
27	23	6	67	55	V
30	24	7	70	56	W
31	25	8	71	57	X
32	26	9	72	58	Y
33	27	:	73	59	Z
34	28	;	74	60	[
35	29	<	75	61	\
36	30	=	76	62	]
37	31	>	77	63	↑

STRING DATA

Each string variable is three PDP-8 words long. Each word contains two 6-bit bytes or characters. If a string variable is filled by a GET from a source which was not written by a BASIC program, a BASIC program may examine the data in the variable by performing a CHANGE on that variable. The six bytes will be translated as if they were internal character codes for BASIC string characters. Table 9-3 shows how this translation interprets the 64 possible types. Note that after such a CHANGE, the 0th element of the array contains a count of the number of characters occurring before the first null.

Error Messages

Most programs, especially if they are at all complex do not execute correctly the first time they are tried. EduSystem 50 checks all BASIC statements when they are entered and before executing them. If it cannot execute a statement, it informs the user by printing one of the error messages shown in Table 9-4, followed by the line number, if present, in which the error occurred.

In addition, the system checks for non-fatal execution errors and notifies the user that he performed a computational range error. When errors of this type occur, the messages shown in Table 9-5 appear, followed by the line number in which the error occurred.

Table 9-4. BASIC Error Messages

Message	Explanation
WHAT?	The editor cannot understand the command given.
SYSTEM I-O ERROR	BASIC was unable to perform the desired disk I/O.
PROGRAM TOO LARGE	The program is too large to be executed. Make it smaller.
NO END STATEMENT IN PROGRAM	All programs must have an END statement.
ILLEGAL INSTRUCTION	A statement was used which is not one of the legal BASIC statements.

Table 9-4 (Cont.). BASIC Error Messages

Message	Explanation
ILLEGAL SYNTAX	The expression in a statement does not agree with the BASIC syntax.
DEF STATEMENT MISSING	A function needing a DEF statement exists in the program.
FOR WITHOUT NEXT	There is an unmatched FOR statement in the program.
NEXT WITHOUT FOR	The NEXT statement indicated has no preceding FOR statement.
ILLEGAL CHARACTER	The user attempted to use an illegal character in the statement being processed.
ILLEGAL LINE NUMBER	The format of the line number being used in a GOTO or IF statement is not acceptable.
ILLEGAL OPERATION	The expression being processed does not agree with the BASIC rules (this is probably due to unmatched parentheses).
STACK OVERFLOW	The user programmed a situation in which the expression is too complicated to be executed.
ILLEGAL CONSTANT	The format of a constant in the statement being processed is not valid.
OUT OF DATA	An attempt was made to READ more data than was supplied by the user.
ILLEGAL FORMAT	The structure of the statement does not agree with BASIC syntax.
DIMENSION TOO LARGE	Too large an array to fit in the available core.
UNDEFINED LINE NUMBER	The line number appearing in a GOTO or an IF-THEN statement does not appear in the program.
BAD FILE NAME	The file name used is not valid, e.g., it does not begin with a letter.

Table 9-4 (Cont.). BASIC Error Messages

Message	Explanation
SUBSCRIPT ERROR	A negative subscript was used for an array.
MISUSED TAB	The TAB function was used in an invalid manner. TAB can appear only in PRINT statements.
GOSUB—RETURN ERROR	Subroutines are too deeply nested or a RETURN statement exists outside a subroutine.
ILLEGAL FOR NESTING	FOR-NEXT loops are too deeply nested or NEXT appears before FOR.
ILLEGAL VARIABLE	An illegal variable was used in an array.
DISK FULL	There is no more storage space on the system disk.
IMPROPER ACCOUNT # ABORT ↑BS	A user logged in under account numbers 1 (system account) or 2 (system library) and tried to run BASIC. This is prohibited.
BAD FILE FORMAT	The program specified in response to OLD PROGRAM NAME was not acceptable to BASIC. This is generally caused by: (1) trying to load an obsolete compiled (.BAC) file, or (2) trying to load a non-BASIC (FORTRAN or PAL-D) program.
MISUSE OF CHR$	The CHR$ function was used in an invalid manner. CHR$, like TAB, can appear only in PRINT statements.
BAD VALUE IN CHANGE STATEMENT	While performing CHANGE A TO A$, one of the elements of the array A was found to contain an illegal value.
TIME LIMIT EXCEEDED	The number of statements executed by a job has exceeded the maximum established by the system manager. Generally, some error was made and the program is caught in a loop.

Table 9-4 (Cont.). BASIC Error Messages

Message	Explanation
PROGRAM IS "progname"	This message may immediately follow an error message, to identify the current program in a series of CHAINed programs. If there is no CHAIN, this message will not occur.
PROGRAM NOT FOUND	The file which the user tried to access with a CHAIN statement does not exist in his disk area. The PROGRAM IS message will also occur.
BAD SLEEP ARGUMENT	The argument of the SLEEP statement must have a number greater than or equal to 0, and less than or equal to 4095.
ARRAY OR RECORD USED BEFORE DEFINITION	The RECORD statement must occur before any reference to it is made. A DIM statement must occur before an array is used. (RECORD and DIM are placed at the beginning of a program.)
IMPROPER DIM OR RECORD STATEMENT	Syntax error in DIM or RECORD statement, or an array name that was previously dimensioned is reused. (Replaces IMPROPER DIM STATEMENT IN LINE n).
CAN'T CREATE FILE	An OPEN statement tried to create a file, but there is: (a) no disk space available, (b) no file name specified, or (c) a null string has been given as the file name.
CAN'T DELETE FILE	UNSAVE cannot delete a file. This is usually due to the fact that another user has the file open, or the file is protected with a code ≥20.
UNOPEN DISK UNIT	The user tried to do a GET, PUT, or UNSAVE to device 8 or 9, without a file being previously opened on the device.

Table 9-4 (Cont.). BASIC Error Messages

Message	Explanation
DEVICE BUSY	The user tried to OPEN DECtapes 0-7, line printer, or paper tape punch, but the device was unavailable, and there was no ELSE clause in the OPEN statement.
INVALID RECORD NO.	The record number must be a number which is greater than or equal to 0 and less than or equal to 4095. For DECtape I/O the maximum record number is limited further by the DECtape size.
ON INDEX OUT OF RANGE	The value of the index is less than one, or greater than the number of statement numbers.
INVALID DEVICE NO.	The device number in the file I/O statement is not between 0 and 11 inclusive, (or X and 11 inclusive where X is a number set by the system manager).
GET BEYOND END OF FILE	Disk data file is too small to have a record with the number specified in the GET statement at line n.
GET/PUT ERROR	A hardware error occupied in GET or PUT. (This is usually due to a DECtape unit being write-locked.)
CHAIN TO BAD FILE	The file specified by the CHAIN has an invalid format; it is not a BASIC format file. The "PROGRAM IS..." message will follow this error message. The program name will be the name of the bad file.

Table 9-5. Non-Fatal Execution Errors

Message	Explanation
/0	Zero divide—an attempt was made to divide by zero. The largest possible number is used for the result.
OV	Overflow—the result of a calculation was too large for the computer to handle. The largest possible number is used for the result.
UN	Underflow—the result of a calculation was too small for the computer to handle. Zero is used for the result.
LN	An attempt was made to compute the logarithm of zero or a negative number. Zero is used for the result.
SQ	An attempt was made to compute the square root of a negative number. The square root of the absolute value is used for the result.
PW	An attempt was made to raise a negative number to a fractional power. The absolute value of that number raised to the fractional power is used.

Table 9-6. EduSystem 50 BASIC Language Summary

Statement	Format	Explanation
Input/Output Statements		
CLOSE	CLOSE f	Close file f.
DATA	DATA $n_1,n_2,...n_n$	Numbers n_1 through n_n = variables in READ statement.
GET	GET f,l,r	Read record r from file f into the variables in line 1.
INPUT	INPUT $v_{1,2},...v_n$	Get v_1 through v_n input from the Teletype.
LINPUT	LINPUT $v\$_1,v\$_2,...v\$_n$	Get long character string input from Teletype (up to 50 characters).
LPRINT	LPRINT $e_1,e_2,...e_n$	Print values of specified text or expressions on line printer or high-speed paper tape punch.
OPEN	OPEN f,n$	Open a file named n$ as file number f.
OPEN-ELSE	OPEN f ELSE n	Open a file, go to line n if unavailable.
PRINT	PRINT $e_1,e_2,...e_n$	Print values of specified text, variables, or expressions.
PUT	PUT f,l,r	Write record r, formatted as in line 1, into file f.
READ	READ $v_1,v_2,...v_n$	Read variables v through v from DATA list.
RESTORE	RESTORE	Reset DATA pointer to beginning value.
RESTORE*	RESTORE*	Reset DATA pointer for numeric data only.
RESTORE$	RESTORE$	Reset DATA pointer for character string data only.
UNSAVE	UNSAVE f	Delete file f from disk storage.

Table 9-6 (Cont.). EduSystem 50 BASIC Language Summary

Statement	Format	Explanation
Transfer of Controls		
GOTO	GOTO 1	Transfer control to line number 1.
IF-GOTO	IF e_1 r e_2 GOTO 1	If relationship r between e_1 and e_2 is true, transfer control to line number 1.
IF-THEN	IF e_1 r e_2 THEN 1	Same as IF-GOTO.
ON-GOTO	ON e GOTO $1_1, 1_2, \ldots 1_s$	Computed GOTO.
Loops and Subscripts		
DIM	DIM $v(d_1)$, $v(d_1,d_2)$	Dimensions, variables subscripted.
FOR-TO-STEP	FOR $v=e_1$ TO e_2 STEP e_3	Set up program loop. Define v values beginning at e_1 to e_2, incremented by e_3.
NEXT	NEXT v	Terminate program loop. (Increment value of v until $v>e_2$.)
Subroutines		
GOSUB	GOSUB 1	Enter subroutine at line 1.
RETURN	RETURN	Return from subroutine to statement following GOSUB statement.
STOP	STOP	Transfer control to END statement.
Others		
CHAIN	CHAIN n$	Link to next program.
CHANGE	CHANGE v_1 TO v_2	Change character string to array of character codes or vice versa.

Table 9-6 (Cont.). EduSystem 50 BASIC Language Summary

Statement	Format	Explanation
DEF	DEF FNA(x)=f(x)	Define a function.
END	END	End of program.
LET	LET v=f	Assign value of formula f to variable v.
RANDOMIZE	RANDOMIZE	Randomize random number routine.
REMARK	REM text	Insert a remark or comment.
SLEEP	SLEEP n	Cause program to pause for n seconds.
Edit/Control Commands		
BYE	BYE	Leave BASIC Monitor.
CATALOG	CAT	List names of programs in storage area.
COMPILE	COM name	Compile program in core and save it on disk.
CTRL/C	CTRL/C	Stop program execution; return to edit phase.
DELETE	DEL n	Delete line n.
	n	Delete line n.
	DEL n,m	Delete lines n through m, inclusive.
EDIT	EDI n (c)	Search line n for character c.
KEY	KEY	Return to keyboard mode after TAPE.
LIST	LIST	List entire program in core.
	LIST n	List line n only.
	LIST n,m	List lines n through m, inclusive.

Table 9-6 (Cont.). EduSystem 50 BASIC Language Summary

Statement	Format	Explanation
LLIST	LLIST	List program to line printer.
NEW	NEW	Clear core, request program name.
OLD	OLD	Clear core, bring program to core from storage area.
REPLACE	REP REP name	Replace old file on disk with version in core. If name is not specified, old name is retained.
RUN	RUN	Compile and run program in core.
SAVE	SAVE name	Store program named on storage device.
SCRATCH	SCR	Erase current program from core.
TAPE	TAP	Read paper tape; suppress printing on Teletype.
UNSAVE	UNSAVE name	Delete program named from storage device.
Functions		
ABS	ABS(x)	Absolute value of x.
ATN	ATN(x)	Arctangent of x (result in radians).
COS	COS(x)	Cosine of x (x in radians)
EXP	EXP(x)	e (e=2.712818).
INT	INT(x)	Greatest integer of x.
LOG	LOG(x)	Natural logarithm of x.
RND	RND(x)	Random number.
SGN	SGN(x)	Sign of x (+1 if positive, −1 if negative, 0 if zero).
SIN	SIN(x)	Sine of x (x in radians).

Table 9-6 (Cont.). EduSystem 50 BASIC Language Summary

Statement	Format	Explanation
SQR	SQR(x)	Square root of x.
TAN	TAN(x)	Tangent of x (x in radians).
TAB	TAB(x)	Controls printing head position on Teletype.
FIX	FIX(x)	Truncates decimal portion of x.
CHR$	CHR$(x)	Converts character code to character. Used only with the PRINT statement.

FOCAL

FOCAL (FOrmula CALculator) is an on-line, interactive, service program for the PDP-8 family of computers, designed to help scientists, engineers, and students solve numerical problems. The language consists of short imperative English statements which are easy to learn. FOCAL is used for simulating mathematical models, for curve plotting, for handling sets of simultaneous equations, and for many other kinds of problems.

To call FOCAL, type:

```
.R FOCAL
```

FOCAL enters its initial dialogue, and asks if its extended functions are to be retained. The extended functions are exponential, sine, cosine, arctangent, and logarithm. If the FOCAL program to be run uses any of these functions, the user responds YES. If not, the user responds NO to free more space for the user program. Without the extended functions, there is room for approximately 1800 characters of program. If the extended functions are retained, there is room for approximately 1100 characters.

Using FOCAL Commands

Whenever FOCAL prints any asterisk, it is in command mode, and the user may type any of the FOCAL commands in response to the asterisk. FOCAL commands may be direct or indirect. A direct command is typed directly after the asterisk and is executed immediately. The format for direct commands is:

**COMMAND*

An indirect command is always identified by a line number. Indirect commands are not executed until program control passes to the line number associated with the command. The format for indirect commands is:

**GG.ss COMMAND*

When the user is typing indirect commands, he may use any line number in the range 1.01 to 31.99, except those ending in .00. Numbers such as 1.00 or 31.00 are illegal as line numbers;

they are used to identify an entire group of line numbers. Line numbers are typed in the format:

GG.ss

where *GG* is the group number and *ss* is the step number. It is not necessary to type two digits after the decimal; e.g., 2.1 is equivalent to 2.10.

All FOCAL commands must be followed immediately with a space. All FOCAL command lines must be terminated with the RETURN key.

FOCAL Overview

FOCAL consists of 12 commands which are all the beginner needs to write programs. FOCAL commands may be typed in their entirety or abbreviated. The FOCAL commands are:

Command	Explanation
TYPE	Used to print text, results of calculations, and values of variables.
ASK	Used to assign values to variables from the keyboard.
SET	Used to define variables and evaluate expressions.
GO or GOTO	Used to direct program control to the lowest line number, or to some specific line number.
IF	Used to direct program control conditionally after a comparison.
DO	Used to cause a specific line or group of lines to be executed.
RETURN	Used to terminate DO routines.
QUIT	Used to halt program execution and return control to user.
FOR	Used to increment a number and execute a user-specified command for each value of the number incremented.

Command	Explanation
COMMENT or CONTINUE	Used for comments or non-executable program steps.
ERASE or ERASE ALL	Used to erase part of a program or an entire program.
MODIFY	Used to edit words or characters on a program line.

These commands are explained in detail with actual computer output in this section. For the convenience of the user, a detailed FOCAL command summary is included in Table 9-7.

Numbers

A FOCAL number may be any decimal number between -10^{615} and 10^{615}. Numbers may be written signed (+ or −) or unsigned, either with a decimal point and a fractional part or in exponential format (see Data Formats) with a mantissa and exponent. In FOCAL, all numbers are internally represented in exponential format, retaining up to six significant digits. If more than six digits are specified, the number will be rounded to six digits. The following numbers are identical in FOCAL:

60
60.00
6E10
600E-10
60.00003

Variable Names

FOCAL variable names may consist of either one or two characters. The first character must always be alphabetic; however, it cannot be an F because FOCAL reserves that character for function names (see FOCAL Functions). The second character may be either alphabetic or numeric. The user may write variable names consisting of more than two characters, but FOCAL uses only the first two characters to identify the variable.[3] Therefore, the first two characters must be unique.

[3] A variable is represented internally as a binary fraction with an exponent. See Data Formats.

```
*SET A=56789
*SET B=123456
*SET C1=15
*SET C2=30
*SET DEPTH=10
*SET DISTANCE=C1+C2
```

Variables may also be subscripted. For a discussion of what subscripted variables are and how they are used, see Subscripted Variables.

Arithmetic Operations

To print the results of arithmetic calculations, the user types the FOCAL command TYPE followed by a space and the data to be calculated. Then he presses the RETURN key, and FOCAL prints the answer. For example:

```
*TYPE 6+10-3-1
=    12.0000*
```

The above example shows two of the arithmetic operations (+ and −) performed by FOCAL. Arithmetic operations are performed from left to right except when the operation to the right has priority or when enclosures are used. (See Enclosures.)

```
*TYPE 6+5; TYPE 5+2-3; TYPE 10-6
=   11.0000=     4.0000=     4.0000*
```

NOTE

Several commands may be typed on the same line if they are separated by semicolons (;). This is true for all FOCAL commands .except the LIBRARY commands.

Unless indicated otherwise, FOCAL mathematical computations retain an accuracy of six significant digits.

PRIORITY OF ARITHMETIC OPERATIONS

The FOCAL arithmetic operations priorities are:

First priority —exponentiation (↑)[4]
Second priority —multiplication (*)

[4] When exponentiation is performed by FOCAL, the power to which a number is raised must be a positive integer. If a calculated exponent exceeds the limits of size, no error message is given. The result will go to zero.

Third priority	—division (/)
Last priority	addition (+) subtraction (−)

When FOCAL evaluates an expression which includes several arithmetic operations, the above order of priority is followed. Therefore, FOCAL evaluates

```
*TYPE 25+5*2+5
```

to have a value of

```
=    40.0000*
```

because multiplication (*) has a higher priority than addition (+).

ENCLOSURES

The order of executing arithmetic operations is also influenced by enclosures. Three kinds of enclosures may be used with FOCAL: parentheses (), square brackets []⁵, and angle brackets <>. FOCAL treats them all the same. For example, the result of the expression:

```
*TYPE (A+B)*<C+D>*(E+F)
```

is the same as the result of:

```
*TYPE (A+B)*(C+D)*(E+F)
```

If the expression contains enclosures within enclosures (called nesting), FOCAL executes the contents of the innermost enclosures first and works outward.

```
*TYPE (5*<2+3>-5)↑2
=   400.0000*
```

[5] [and] are typed by SHIFT/K and SHIFT/M, respectively.

TYPE COMMAND

The TYPE command is used to print results of calculations, values of variables, text or character strings, and variable tables. TYPE may also be used to print combinations of text and variables.

Example 1—Result of a Calculation:

```
*TYPE 1+1
=     2.0000*
```

Example 2—Value of a Variable or Variables:

```
*SET N=5+5; SET M=30
*TYPE N,M
=    10.0000=    30.0000*
```

Example 3—Text:

```
*TYPE "THIS IS A LINE OF TEXT",!6
THIS IS A LINE OF TEXT
*
```

Example 4—Variable Tables:

```
*TYPE $
N@(00)=    10.0000
M@(00)=    30.0000
*
```

The user may command FOCAL to print all of the user-defined variables (variable table) by using the TYPE command and a dollar sign ($).

If a variable consists of only one letter, an at sign (@) is inserted as a second character in the variable table printout, as shown in the example above.

```
*SET N=25
*TYPE "N IS",N
N IS=    25.0000*
```

NOTE

Any variable, constant, or expression in a TYPE or ASK command must be followed by a comma, semicolon, or carriage return.

---

[6] The exclamation mark (!) causes a carriage return and line feed.

ASK COMMAND

The ASK command is normally used in indirect commands to enable the user to input numerical data during the execution of his program. The ASK command is similar to the TYPE command in form, but only single variable names, not expressions, are used; and the user types the values in response to a colon (:) printed by the ASK command.

```
*11.99 ASK X,Y,Z
```

When FOCAL encounters line 11.99 in the above example, it prints a colon and waits for the user to type a value (in any format) and a terminator[7] for the first variable. This process continues until all the variables in the ASK command have been given values.

The value is assigned to the variable when the user types a terminator; so any time before the terminator is typed, the value can be changed. If the user types back arrow (← or SHIFT/O) immediately after the value and before he types a terminator, he can then type the correct value and a terminator.

```
*ASK X,Y,Z
:5                          User typed 5 and RETURN key.
:6                          User typed 6 and RETURN key.
:8←7                        User typed 8, ←, 7 and RETURN
*TYPE X,!,Y,!,Z             key.
=     5.0000
=     6.0000
=     7.0000*
```

The ALT MODE key is a special non-spacing terminator which enables the user to have a previously assigned value unchanged.

```
*ASK X,Y,Z
:3
::12                        User typed ALT MODE because
*TYPE X,!,Y,!,Z             he did not want to change the
=     3.0000                value of Y.
=     6.0000
=    12.0000*
```

[7] Terminators are SPACE, comma, ALT MODE, and RETURN keys. If the user types the RUBOUT key, it is ignored.

Text Output with ASK

The ASK command, just as the TYPE command, may be used to print text. Carriage return and line spacing are controlled the same as with the TYPE command (see Data Formats).

```
*ASK "WHAT IS YOUR AGE?" AGE
WHAT IS YOUR AGE?:19
*
```

The word following the text in the command line (AGE, in this example) is the variable.

Computational Command

SET COMMAND

The SET command enables the user to assign a numerical value to a variable and store both the value and the variable. Then, when he uses the variable in an expression,[8] FOCAL automatically substitutes the numerical value that the user previously specified:

```
*SET E=2.71828
*SET PI=3.14159
*TYPE "PI TIMES E", PI*E
PI TIMES E=     8.5397*
```

The value of a variable may be changed at any time by another SET command.

```
*SET A1=3↑2
*SET A1=A1+1
*TYPE A1
=    10.0000*
```

Control Commands

GO OR GOTO COMMAND

The GO command causes FOCAL to go to the lowest numbered line in the program and begin executing the indirect commands.

```
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE A,B
*GO
=     1.0000=     2.0000:
```

[8] An expression is a combination of arithmetic operations or functions which may be reduced to a single number by FOCAL.

In the above example the GO command caused execution to begin at line 1.1.

The GOTO command causes FOCAL to go to a specific line in the program and begin executing the indirect commands in ascending line number order.

```
*ERASE
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE A,B
*GOTO 1.3
=     0.0000=     2.0000:
:
:ERASE
 *
```

In the preceding example, A and B are equal to zero at the start of the program. The ERASE and ERASE ALL commands are used to ensure that all variables are equal to zero until they are assigned a specific value. Since the GOTO command causes program execution to begin at line 1.3, line 1.1 is never executed and A is not set to 1.

IF COMMAND

The IF command is a conditional command used to transfer program control after a comparison. The normal IF command format is:

IF *space (expression) line1, line2, line3*

The expression or variable is evaluated, and program control is transferred to the first line number if the value of the expression is less than zero, to the second line number if the value is zero, or to the third line number if the value is greater than zero.

The program below transfers control to line number 2.1, 2.3, or 2.5, according to the value of the expression in the IF command.

```
*2.1 TYPE "LESS THAN ZERO"; QUIT
*2.3 TYPE "EQUAL TO ZERO"; QUIT
*2.5 TYPE "GREATER THAN ZERO"; QUIT
*IF (25-25)2.1,2.3,2.5
EQUAL TO ZERO*
```

IF with Less Than Three Line Numbers

The IF command format can be altered to transfer program control to one or two lines. For example, if a semicolon or a carriage

return is immediately after the first line number, control goes to the first line number if the value of the expression is less than zero. If the value is not less than zero, control goes to the next sequential command. For example:

```
*2.20 IF (X)1.8;TYPE"Q"
```

When line 2.20 is executed, program control goes to line 1.8 if X is less than zero. If Xis not less than zero, Q is typed.

If a semicolon or a carriage return follows the second line number, control goes to the first or second line number, depending upon whether the value of the expression is less than zero or equal to zero. If the value is greater than zero, control goes to the next sequential command. For example:

```
*3.19 IF (B)1,1.9
*3.20 TYPE B
```

If B is less than zero, control goes to line 1.8; if B equals zero, control goes to 1.9; and if it is greater than zero, control goes to the next sequential command, in this case line 3.20, and the value of B is printed.

Arithmetic Comparison with IF Command

The IF command can be used with all the arithmetic operations of FOCAL.

Example 1—Addition:

```
*1.10 IF (A+B)2.1,2.5,2.10
```

Example 2—Subtraction:

```
*5.16 IF (A-B)1.6; TYPE "X"
```

Example 3—Division:

```
*3.10 IF (M/N)5.5,5.6;
*3.15 TYPE "GREATER THAN ZERO"
```

Example 4—Multiplication:

```
*7.20 IF (P*I)8.1;
*7.25 TYPE P*I
```

Example 5—Exponentiation:

```
*4.15 IF (X↑N)6.1,6.2,6.3
```

DO COMMAND

The DO command is used to make subroutines of single lines or groups of lines. Control is returned to the line following the DO command after the subroutine is executed.

```
*1.1 SET A=1; SET B=2
*1.2 TYPE "STARTING"
*1.3 DO 3.2
*2.1 TYPE "FINISHED"
*3.1 SET A=3; SET B=4
*3.2 TYPE A+B
*GO
STARTING=     3.0000FINISHED=     7.0000*
*
```

If the user types a command such as DO 3, the DO command treats the group of program lines beginning with 3 as a subroutine. Control proceeds in ascending order through the group numbers until the end of the group is reached, or until a RETURN command is executed.

Nested DO

DO commands may be nested as shown in the following example:

```
*1.1 TYPE "BEGIN",!
*1.2 DO 2
*1.3 TYPE "END",!; QUIT
*
*2.1 DO 5.1; TYPE A,!
*2.2 DO 5.2; TYPE A,!
*2.3 DO 7.5; TYPE A,!
*
*5.1 SET A=1
*5.2 SET A=2
*
*7.5 SET A=3
*GO
BEGIN
=     1.0000
=     2.0000
=     3.0000
END
*
```

The number of nested DO commands is limited only by the amount of core memory available after storage is allocated for program and variables.

RETURN COMMAND

The RETURN command is used to exit from a DO subroutine. When a RETURN command is encountered during execution of a DO subroutine, the program exists from its subroutine status and returns to the command following the DO command that initiated the subroutine status.

QUIT COMMAND

When the QUIT command is executed, FOCAL prints an asterisk and returns to command mode.

FOR COMMAND

The FOR command is used to set up program loops and iterative procedures. The general command format is:

*FOR *A* = *B,C,D; commands*
*FOR *A* = *B,D; commands*

The variable A is initialized to the value B, then the command or commands following the semicolon are executed. When the commands have been executed, the value of A is incremented by C and compared to the value of D. If A is less than or equal to D, the command after the semicolon is executed again. This process is repeated until A is greater than D, at which time FOCAL goes to the next sequential line. The command or commands will always be executed at least once.

A must be a single variable. B, C, and D may be expressions, variables, or numbers. If the value C is omitted, it is assumed that the increment is one. If C and D are omitted, the FOR command is handled like a SET command and no iteration is performed. A FOR command may be used with a DO command and nested:

FOR with a DO

```
*ERASE ALL
*1.1 FOR X=1,1,5; DO 2
*1.2 QUIT
*
*2.1 TYPE !"         X",X
*2.2 SET A=X+100
*2.3 TYPE !"         A",A
*GO

         X=     1.0000
         A=   101.0000
         X=     2.0000
         A=   102.0000
         X=     3.0000
         A=   103.0000
         X=     4.0000
         A=   104.0000
         X=     5.0000
         A=   105.0000*
```

Nested FOR and DO

```
*1.1 FOR Z=1,3; TYPE "  A  B  C  "
*1.2 TYPE !
*1.5 FOR A=1,3; DO 3
*1.7 QUIT
*
*3.1 FOR B=1,3; DO 4
*
*4.1 FOR C=1,3; TYPE %1,A,B,C,"  "
*4.2 TYPE !
*GO
  A  B  C     A  B  C     A  B  C
= 1= 1= 1   = 1= 1= 2   = 1= 1= 3
= 1= 2= 1   = 1= 2= 2   = 1= 2= 3
= 1= 3= 1   = 1= 3= 2   = 1= 3= 3
= 2= 1= 1   = 2= 1= 2   = 2= 1= 3
= 2= 2= 1   = 2= 2= 2   = 2= 2= 3
= 2= 3= 1   = 2= 3= 2   = 2= 3= 3
= 3= 1= 1   = 3= 1= 2   = 3= 1= 3
= 3= 2= 1   = 3= 2= 2   = 3= 2= 3
= 3= 3= 1   = 3= 3= 2   = 3= 3= 3
*
```

Another way of handling the same program is:

```
*1.1 FOR Z=1,3;TYPE "  A  B  C  "
*1.2 FOR A=1,3;FOR B=1,3;TYPE !;FOR C=1,3;TYPE %!,A,B,C,"  "
*GO
```

Subscripted Variables

Variables may be further identified by subscripts which are enclosed in parentheses immediately following the variables. For example:

```
*SET AB(0)=5
*SET AB(1)=10
*SET AB(2)=15
*SET AB(3)=20
*SET AB(4)=25
*SET AB(5)=30
*FOR X=0,5; TYPE AB(X),!
=  5
=  0.1E+02
=  0.2E+02
=  0.2E+02
=  0.3E+02
=  0.3E+02
*
```

In the above example, subscripts are used to set up an array called AR. Any element in the array can be represented by a subscript in the range 0 to 5. The first element in an array always has a subscript of 0. A subscript may be a number, another variable, or an expression. If it is a number, it must be in the range ±2047. In order to be properly represented by the TYPE $ command, subscript numbers must be positive integers in the range from 0 to 99. The TYPE $ command will print subscripts greater than 99 as two random characters, although their contents will be correct as assigned by the program.

COMMENT OR CONTINUE COMMAND

The COMMENT or CONTINUE command (abbreviated as C) causes the program line to be ignored by FOCAL. The user may use the C command to insert comments into the program, or he may use the C command line as a non-executable program step. In either case, program lines beginning with C are skipped when the program is executed. However, comments are printed in response to a WRITE command.

```
*ERASE ALL
*1.1 C INITIALIZE VARIABLES
*1.2 SET A=5
*1.3 SET B=6
*1.4 SET C=7
*
*2.1 C PERFORM CALCULATION
*2.2 TYPE A+B+C
*
```

Edit Commands

WRITE OR WRITE ALL COMMAND

The WRITE or WRITE ALL command causes FOCAL to print a program line, a group of lines, or an entire indirect program on the Teletype.

*WRITE 2.1	Print a single line.
*WRITE 2	Print a group of lines.
*WRITE	Print an entire program.

Once the program is completed, the user may want to save it by putting it on paper tape. The procedure for saving a FOCAL program on-line is as follows:

1. Make sure FOCAL is in command mode.
2. Type WRITE
3. Set low-speed punch to ON position.
4. Type RETURN key.

FOCAL will punch the entire program onto the paper tape and simultaneously print it on the Teletype.

Whenever the user wants to run the program he has saved on paper tape, he does the following:

1. Makes sure FOCAL is in command mode.
2. Puts the program tape in low-speed reader.
3. Switches the low-speed reader to START.

The program will be put into core the same as it would if the user were typing it on the Teletype keyboard. When the entire program has been read into core, the user should type CTRL/C since the asterisk printed when the WRITE command is finished is also punched and may be interpreted as a command. CTRL/C ensures that FOCAL returns to command mode.

ERASE AND ERASE ALL COMMANDS

The ERASE command deletes symbolic names, lines, or groups of lines.

`*ERASE`	Delete all names defined in symbol table.
`*ERASE 2.2`	Delete line 2.2
`*ERASE 2`	Delete all lines in group 2.

The user can check to see if the line(s) has been deleted by typing the WRITE command after the ERASE command. This is a useful procedure for checking commands and also for obtaining a clean printout of the current program.

The ERASE ALL command deletes the entire indirect program. It is good programming practice to type ERASE ALL before starting to type a new program. The ERASE ALL command is generally used only with direct commands because it returns control to command mode upon completion.

MODIFY COMMAND

The MODIFY command is used to change characters within a line without changing the entire line. The format for MODIFY is:

MODIFY *line number RETURN key Search character*

The search character is not printed. After the user has typed the line number, RETURN key, and search character, FOCAL prints the contents of the specified line until it encounters the search character. When the search character is read and printing stops, the user has any one or more of the following options:

1. Type new characters in addition to those already printed.
2. Type a form feed (CTRL/L). This causes the search to proceed to the next occurrence, if any, of the search character.
3. Type CTRL/BELL. The user can then change the search character he specified in the MODIFY command.
4. Type the RUBOUT key. This causes FOCAL to delete a character, starting with the last character printed and deleting one character to the left each time RUBOUT is typed.

5. Type the back arrow (←) key. This causes FOCAL to delete everything between the back arrow and the line number.
6. Type the RETURN key. This causes FOCAL to terminate the line at that point, deleting everything to the right.
7. Type the LINE FEED key. This is normally done after the user has exercised one or more of the above options. After the user has modified the line, he may type LINE FEED and cause the remainder of the line from the search character to the end to be printed and saved.

```
*7.01 JACK AND BILL W$NT UP THE HILL
*MODIFY 7.01
JACK AND B\JILL W$\ENT UP THE HILL
```

In the above example, B was typed as the search character for line 7.01. (Note that the search character did not print.) FOCAL stopped printing when it encountered the search character (B), and the user typed the RUBOUT key (\) to delete the B. Then he typed the correct letter J. Next he typed CTRL/BELL and the $ key to change the search character. FOCAL continued to print the line until the new search character was encountered. The user typed RUBOUT to delete the $ and then typed the correct character (E). He then typed the LINE FEED key and the remainder of the line was printed.

CAUTION

When any text editing is done. the values in the user's symbol table are reset to zero.

If the user defines his symbols in direct statements and then uses a MODIFY command, the values of his symbols are erased and must be redefined. However, if the user defines his symbols by means of indirect statements prior to using a MODIFY command, the values will not be erased because these symbols are not entered in the symbol table until the statements defining them are executed. Notice in the example below that the values of A and B were set using direct statements. The use of the MODIFY command reset their values to zero and listed them after the defined symbols.

```
*ERASE ALL
*SET A=1
*SET B=2
*1.1 SET C=3
*1.2 SET D=4
*1.3 TYPE A+B+C+D; TYPE !; TYPE $
*MODIFY 1.1
SET C=3\5
*GO
?07.22 @ 01.10
*
*MODIFY 1.1
SET C=3\5
*GO
= 9
C@(00)= 5
D@(00)= 4
A@(00)= 0
B@(00)= 0
*
```

Library Commands

In addition to the basic FOCAL commands, there are four special commands to perform the library functions: storing and retrieving data from the system disk. All commands following the LIBRARY command on the same line are ignored. Names of files may be from 1 to 4 characters long. Only alphanumeric characters, letters, and numbers should be used in file names. The library commands, like other FOCAL commands, may be abbreviated.

LIBRARY SAVE

This command copies the current program into the user's area on disk and gives it the name specified. For example, the command:

```
*L S TRUE
```

will cause the current program to be stored on disk under the name TRUE. The program will also remain in the user's area.

LIBRARY CALL

This command copies the named program from disk into the user's area. For example, the commands:

```
*L C TRUE
*W
```

will cause the program TRUE to be recalled from the disk into the user's area and listed on the Teletype. Any program currently in

the user's area will be erased. The program TRUE can then be executed with a GO command.

If the LIBRARY CALL command is given a line number and stored, it must have at least one numbered command following it. In this case, the LIBRARY CALL command will cause the named program to be called into the user's area and executed as if GO had been typed. For example:

```
*3.10 L C TRUE
*3.11 C
```

If the above commands are included in a program, they will cause TRUE to be brought in and executed at that point. If there are lines beyond 3.11 remaining to be executed, they will be deleted. This procedure allows the user to chain FOCAL programs as in BASIC.

LIBRARY DELETE

This command deletes the named program from the disk directory. For example, the command:

```
*L D TRUE
```

will remove TRUE from the disk.

LIBRARY LIST

This command causes the names of all the FOCAL programs stored on the disk to be listed on the Teletype, ten names to a line, followed by the total number of free blocks remaining on the disk. The LIBRARY LIST command removes the contents of the user's area with an ERASE ALL command.

ERROR MESSAGES WITH LIBRARY COMMANDS

When the LIBRARY commands are used, five errors are possible. These are also listed in the error code summary in Table 9-9 at the end of this section.

Command	Explanation
?30.71	The command appeared to be a LIBRARY command but was not, for example: LIBRARY OPEN No action is taken; the command should be retyped.
?30.<0	Either an unacceptable file name was specified or no name was specified where one was required.
?31.42	The file name specified does not match any name currently in the user's disk directory. This error will only occur with the LIBRARY CALL and LIBRARY DELETE commands.
?31.43	There is already a program with the specified name in the directory. This error will only occur with the LIBRARY SAVE command.
?31.44	This user's disk directory is full. The current program cannot be saved until others have been deleted. This error will only occur with the LIBRARY SAVE command.

Estimating Program Length

FOCAL permits approximately 1060 (decimal) locations to be used for the user program and variables without the extended math functions, and approximately 670 locations with the extended functions (sine, cosine, log, exponential, etc.). Since FOCAL requires five locations for each variable stored in the variable table and one location for each two characters of stored program, the approximate length of a program may be determined by the formula:

$$\textit{Length of program} = 5S + \frac{C}{2} + 2L$$

where

S = number of variables
C = number of characters in program
L = number of lines

If the total program area or variable table area becomes too large, FOCAL prints an error message (06.54 or 10.:5). The following routine allows the user to find out how many core locations are left for his use:

```
*ERASE
*FOR I=1,300; SET A(I)=1
?06.54                                  (Disregard error code.)
*TYPE %4,I*5,"LOCATIONS LEFT"
= 1075LOCATIONS LEFT*
*
*
```

Debugging

USING THE ERROR DIAGNOSTICS

Whenever FOCAL detects an illegal command or improbable condition within a user's program, the execution of the program stops and an error message is printed in the form *?XX.XX@GG.ss*, where *?XX.XX* is the error message and *GG.ss* is the line at which the error occurred. (See Table 9-9 for the complete list of error messages.)

Depending upon the type of error detected, the user may ignore the error message or make program changes before continuing. For example, if the user types CTRL/C to terminate a loop, the error message ?01.00 is printed and program control goes to command mode; so, in this case, the user ignores the message and types his next command. If a program stops and the message ?03.05 is printed, the user must examine his program to determine which command line is wrong. In the following program, line 1.3 contains the instruction to transfer to a nonexistent line number.

```
*ERASE ALL
*1.10 SET A=2; TYPE "A",A,!
*1.20 SET B=4; TYPE "B",B,!
*1.30 GOTO 1.01
*1.40 TYPE "A+B",A+B
*GO
A=     2
B=     4
?03.05 @ 01.30
*
```

USING THE TRACE FEATURE

The trace feature is used to check the logic in part of a FOCAL program. To implement the trace feature, the user inserts a question mark (?) into a command string at any point. FOCAL prints each succeeding character as it is executed until another question mark is encountered or until the program returns to command mode. For example, the trace feature is used to print parts of 3 lines in the following program:

```
*ERASE ALL
*1.1 SET A=1
*1.2 SET B=5
*1.3 SET C=3
*1.4 TYPE ?A+B-C?,!
*1.5 TYPE ?B+A/C?,!
*1.6 TYPE ?B=C/A?
*GO
A+B-C=  3
B+A/C=  5
B=C/A=  2*
```

NOTE

The WRITE command disables the trace feature.

FOCAL Functions

The FOCAL functions improve and simplify arithmetic capabilities. In general, the FOCAL functions may be used anywhere a number or a variable is legal in a mathematical expression. A standard function call consists of three or four letters beginning with the letter F and followed by an expression in parentheses. The FOCAL functions are summarized in Table 9-8.

The functions must be used with a legal FOCAL command. They cannot be used alone as commands. For example:

```
*SET Z=A+FSQT(FSIN(X))
```

Within a normal range of arguments, at least five significant digits of accuracy may be expected for the trigonometric, exponential, and logarithmic functions. The following functions are available to FOCAL users.

SINE FUNCTION (FSIN)

The sine function (FSIN) is used to calculate the sine of a user-specified angle in radians. The format for FSIN is:

FSIN (*angle*)

```
*TYPE FSIN(3.14159/4)
=     0.7071*
```

The format for calculation the sine of an angle in degrees is:

FSIN *(degrees * 3.14159/180)*

```
*TYPE FSIN(30*3.14159/180)
=     0.50000*
```

COSINE FUNCTION (FCOS)

The cosine function is used to calculate the cosine of a user-specified angle in radians. The format for FCOS is:

FCOS *(angle)*

```
*TYPE FCOS(2*3.141592)
=     1.0000*
*TYPE FCOS(.50000)
=     0.8776*
*TYPE FCOS(45*3.141592/180)
=     0.7071*
```

EXPONENTIAL FUNCTION (FEXP)

The exponential function (FEXP) is used to compute e (e=2.71828) to a power specified by the user. The format for FEXP is:

FEXP *(power)*

```
*TYPE FEXP(1)
=     2.7183*
*TYPE FEXP(0)
=     1.0000*
*TYPE FEXP(-1-23456)
=     0.0000*
```

LOGARITHM FUNCTION (FLOG)

The logarithm function (FLOG) is used to compute the natural logarithm ($\log_e$) of a number specified by the user. The format for FLOG is:

FLOG *(number)*

```
*TYPE FLOG(1.00000)
=     0.0000*
*TYPE FLOG(1.98765)
=     0.6870*
*TYPE FLOG(2.065)
=     0.7251*
```

The following formulas are useful for finding logarithms to base 10:

$$\log_{10}X=\log_eX/\log_e10=0.434294\ \log_eX$$

ARCTANGENT FUNCTION (FATN)

The arctangent function (FATIN) is used to calculate the angle in radians of a user-specified tangent. The format for FATN is:

FATN *(tangent)*

```
*TYPE FATN(1.)
=     0.7854*
*TYPE FATN(.31305)
=     0.3034*
*TYPE FATN(3.141592)
=     1.2626*
```

SQUARE ROOT FUNCTION (FSQT)

The square root function (FSQT) is used to compute the square root of an expression. The format for FSQT is:

FSQT *(expression)*

```
*TYPE FSQT(4)
=     2.0000*
*TYPE FSQT(9)
=     3.0000*
*SET Z=FSQT(144);TYPE Z
=    12.0000*
```

ABSOLUTE VALUE FUNCTION (FABS)

The absolute value function (FABS) is used to indicate the absolute (positive) value of an expression. The format for FABS is:

FABS *(expression)*

```
*TYPE FABS(-66)
=    66.0000*
*TYPE FABS(+23)
=    23.0000*
*TYPE FABS(-99.05)
=    99.0500*
```

SIGN PART FUNCTION (FSGN)

The sign part function (FSGN) is used to output the sign part (+ or −) of a number with a value of 1. FSGN (0)=1. The format for FSGN is:

FSGN *(expression)*

```
*TYPE FSGN(6-4)
=     1.0000*
*TYPE FSGN(0)
=     1.0000*
*TYPE FSGN(-7)
=-    1.0000*
```

INTEGER PART FUNCTION (FITR)

The integer part function (FITR) is used to output the integer part of a number. The format for FITR is:

FITR *(expression)*

For positive numbers, FITR(X) is the greatest integer function. For negative numbers, FITR(−X) is the integer part of −X. The greatest integer function for negative numbers is obtained by FITR(−X)−1. For example:

```
*TYPE FITR(5.2)
=    5.0000*
*TYPE FITR(55.66)
=   55.0000*
*TPE
*TYPE FITR(-4.1)
=-   4.0000*
```

RANDOM NUMBER FUNCTION (FRAN)

The random number function (FRAN) is used to generate non-statistical pseudo-random numbers in the range 0.5000 to 0.9999. No argument is used with the FRAN function. The format for FRAN is:

FRAN( )

```
*TYPE FRAN()
=    0.7376*
*TYPE FRAN()
=    0.8959*
```

FRAN can be used to produce a less biased number. For example:

```
*SET A=FRAN()
*SET B=A=FITR(A)
```

The value assigned to B is a random number in the range 0.0000 to 0.9999.

Focal Output Operations

The following is a description of symbols used in FOCAL output operations:

	Symbol	Explanation
To set output format,	TYPE %x.y	Where x is the total number of digits, and y is the number of digits to the right of the decimal point.
	TYPE %6.3, 123.456	FOCAL prints: =+123.456
	TYPE %	Resets output format to floating point.
To type symbol table,	TYPE $	Other statements may not follow on this line.

Control Characters

FOCAL control characters and their explanations are shown below:

%	Output format delimiter	
!	Carriage return and line feed	
#	Carriage return	
$	Type symbol table contents	
( )	Parentheses	(mathematics)
[ ]	Square brackets	(mathematics)
<>	Angle brackets	(mathematics)
" "	Quotation marks	(text string)
??	Question marks	(trace feature)
*	Asterisk	(high-speed reader input)
SPACE key (names)		(nonprinting)
RETURN key (lines)		(nonprinting)
ALT MODE key (with ASK statement)		(nonprinting)
Comma (expressions)		
Semicolon (commands and statements)		

Reading FOCAL Paper Tapes

To ensure that FOCAL paper tapes are read without error, they should be read silently. To do this, type ↑B (CTRL/B) followed by UNDUPLEX just prior to reading the tape. This Monitor command suppresses the printing of the program as it is read. As each line is read, a line feed and FOCAL's asterisk are printed, indicating that the line is properly stored. After the tape has been completely read, type ↑B DUPLEX to restore FOCAL to its normal mode. An example is shown below:

```
*ERASE ALL
*↑BUNDUPLEX

*
 *
  *
   *
    *
     *
      *
       *
        *↑BDUPLEX

*WRITE ALL
C-FOCAL,1969

01.05 C PROGRAM TO CALCULATE THE HYPOTENUSE OF A
01.06 C RIGHT TRIANGLE GIVEN THE TWO SIDES
01.10 ASK "SIDES OF TRIANGLE ARE" A,B
01.20 SET C=FSQT(A↑2+B↑2)
01.30 TYPE "HYPOTNEUSE
01.30 TYPE "HYPOTENUSE IS" C,!
01.40 GOTO 1.1
*
```

Table 9-7. FOCAL Command Summary

Command	Abbreviation	Example of Form	Explanation
ASK	A	ASK X, Y, Z	FOCAL prints a colon for each variable; the user types a value to define each variable.
COMMENT	C	COMMENT	If a line begins with the letter C, the remainder of the line will be ignored.
CONTINUE	C	C	Dummy lines.
DO	D	DO 4.1	Execute line 4.1; return to command following DO command.
	D	DO 4.0	Execute all group 4 lines; return to command following DO command, or when a RETURN is encountered.
	DA	DO ALL	Execute all program lines until a RETURN is encountered.
ERASE	E	ERASE	Erases the symbol table.
		ERASE 2.0	Erases all group 2 lines.
		ERASE 2.1	Erases line 2.1.
		ERASE ALL	Erases all user input.
FOR	F	FOR i=x,y,z; (commands)	Where the command following is executed at each new value. x=initial value of i. y=value added to i until i is greater than z.

Table 9-7 (Cont.). FOCAL Command Summary

Command	Abbreviation	Example of Form	Explanation
GO	G	GO	Starts indirect program at lowest numbered line number.
GO?	G?	GO?	Starts at lowest numbered line number and traces entire indirect program until another ? is encountered, until an error is encountered, or until completion of program.
GOTO	G	GOTO 3.4	Starts indirect program (transfers control to line 3.4). Must have argument.

IF	I	IF (X)Ln, Ln, Ln
		IF (X)Ln, Ln; (commands)
		IF (X)Ln; (commands)
			Where X is a defined identifier, a value, or an expression, followed by three line numbers. If X is less than zero, control is transferred to the first line number. If X is equal to zero, control is to the second line number. If X is greater than zero, control is to the third line number.

LIBRARY CALL	LC	LIBRARY CALL name	Calls stored program from the disk.
LIBRARY DELETE	LD	LIBRARY DELETE name	Removes program from the disk.
LIBRARY LIST	LL	LIBRARY LIST	Prints directory of stored program names.
LIBRARY SAVE	LS	LIBRARY SAVE name	Saves program on the disk.

Table 9-7 (Cont.). FOCAL Command Summary

Command	Abbreviation	Example of Form	Explanation
MODIFY	M	MODIFY n	Enables editing of any character on line n.
QUIT	Q	QUIT	Returns control to the user.
RETURN	R	RETURN	Terminates DO subroutines, returning to the original sequence.
SET	S	SET A=5/B*C;	Defines identifiers in the symbol table.
TYPE	T	TYPE A+B−C;	Evaluates expression and prints = and result in current output format.
		TYPE A−B, C/E;	Computes and prints each expression separated by commas.
		TYPE "TEXT STRING"	Prints text. May be followed by ! to generate carriage return-line feed, or # to generate carriage return.
WRITE	W	WRITE	FOCAL prints the entire indirect program.
	WA	WRITE ALL	
		WRITE 1.0	FOCAL prints all group 1 lines.
		WRITE 1.1	FOCAL prints line 1.1.

Table 9-8. FOCAL Functions

Function	Format	Interpretation
Square Root	FSQT(x)	Where x is a positive number or expression greater than zero.
Absolute Value	FABS(x)	FOCAL ignores the sign of x.
Sign Part	FSGN(x)	FOCAL evaluates the sign part only, with 1.0000 as integer.
Integer Part	FITR(x)	FOCAL operates on the integer part of x, ignoring any fractional part.
Random Number Generator	FRAN(x)	FOCAL generates a random number. The value of x is ignored.
[9]Exponential	FEXP(x)	FOCAL generates e to the power x. (2.71828^x)
[9]Sine	FSIN(x)	FOCAL generates the sine of x in radians.
[9]Cosine	FCOS(x)	FOCAL generates the cosine of x in radians.
[9]Arctangent	FATN(x)	FOCAL generates the arctangent of x in radians.
[9]Logarithm	FLOG(x)	FOCAL generates $\log_e (x)$.

[9] These are extended functions and may be chosen or deleted when FOCAL is loaded.

Table 9-9. FOCAL Error Messages

Message	Explanation
?00.00	Manual start given from console.
?00.00	Interrupt from keyboard via CTRL/C.
?01.40	Illegal step or line number used.
?01.78	Group number is too large.
?01.96	Double periods found in a line number.
?01.:5	Line number is too large.
?01.;4	Group zero is an illegal line number.
?02.32	Nonexistent group referenced by DO.
?02.52	Nonexistent line referenced by DO.
?02.79	Storage was filled by push-down list.
?03.05	Nonexistent line used after GOTO or IF.
?03.28	Illegal command used.
?04.39	Left of = in error in FOR or SET.
?04.52	Excess right terminators encountered.
?04.60	Illegal terminator in FOR command.
?04.:3	Missing argument in display command.
?05.48	Bad argument to MODIFY.
?06.06	Illegal use of function or number.
?06.54	Storage is filled by variables.
?07.22	Operator missing in expression or double E.
?07.38	No operator used before parenthesis.
?07.:9	No argument given after function call.
?07.;6	Illegal function name or double operators.
?08.47	Parentheses do not match.
?09.11	Bad argument in ERASE.
?10.:5	Storage was filled by text.
?11.35	Input buffer has overflowed.
?20.34	Logarithm of zero requested.
?23.36	Literal number is too large.
?26.99	Exponent is too large or negative.
?28.73	Division by zero requested.
?30.05	Imaginary square roots required.
?31.<7	Illegal character, unavailable command, or unavailable function used.
?30.71	Undefined library command.
?30.<0	Bad argument or missing argument to library command.
?31.42	No such name in library directory.
?31.43	Attempt to enter a duplicate name in the directory.
?31.44	Library directory is full.

FORTRAN-D

FORTRAN-D compiles and runs programs written in the PDP-8 version of FORTRAN II. Programs (usually created and stored with the Symbolic Editor) are compiled in a single pass and executed (automatically) immediately following compilation.

Calling FORTRAN-D

To use FORTRAN-D, type:

```
.R FORT
```

FORTRAN requests the name of the input file, i.e., the file containing the FORTRAN program to be compiled and run. The user responds with the file name and the RETURN key. FORTRAN then requests the name of an output file in which to store the compiled version of the program. For normal usage, just the RETURN key need by typed. FORTRAN places the compiled code in a file of its own, then proceeds to run the program.

If a file name is entered for output, FORTRAN creates a permanent file in which the compiled binary program is saved. It is then possible to rerun this program without recompiling it. To run an already compiled program, call the FORTRAN operating system directly by typing:

```
.R FOSL
```

FOSL requests the name of an input file. Enter the name of the file containing the compiled binary. For example, if the user types:

```
.R FORT

INPUT:MTRIX
OUTPUT:
```

FORTRAN compiles and executes the program MTRIX but does not save the compiled binary.

FORTRAN compiles and executes the program MTRIX and

then leaves the compiled binary in the file named BMTRIX when the user types:

```
.R FORT

INPUT:MTRIX
OUTPUT:BMTRIX
```

The FORTRAN binary program BMTRIX is executed without first being compiled when the user types:

```
.R FOSL

INPUT:BMTRIX
```

All FORTRAN programs return to the Monitor when they have completed execution.

Using FORTRAN-D

Differing versions of PDP-8 FORTRAN offer slightly different features. FORTRAN-D differs in the way it is called into use (described above), and in its more powerful I/O capability (described below). FORTRAN-D allows three data formats:

I Integer format
E Exponential format
A Alpha format, ASCII value of a character is stored as an integer value.

The standard device for READ and WRITE statements is the Teletype, which is assigned device code 1. Because the Teletype is so frequently used, FORTRAN-D includes two special input/output instructions, ACCEPT and TYPE. These instructions imply use of the Teletype; therefore, the device code need not be specified. ACCEPT is especially convenient if data is to be entered at the keyboard because this instruction automatically supplies a line feed when the RETURN key is typed. Also, the user can correct an erroneously typed value by striking the RUBOUT key.

A FORTRAN-D program can also utilize the high-speed reader and punch for I/O. These devices are assigned code 2. Because the high-speed reader and punch are shared by all users, it is necessary to assign them if they are to be used. Assign the appropriate devices and mount tapes in the reader before running FORTRAN-D. An

automatic DEASSIGN is performed by FORTRAN before it returns to the Monitor; therefore, the user must reassign the devices before each run.

FORTRAN-D also allows programs to read and write data files on the disk. These data files are completely separate from the program files. Data files are read and written by standard READ and WRITE statements within the FORTRAN-D program. The device code for the disk is 3. Because programs using the disk are treated differently by FORT (the FORTRAN-D compiler), it is necessary to identify programs which use the disk. These programs are identified by a DEFINE DISK statement as the first statement in any such FORTRAN-D program including a READ or WRITE statement with device code 3.

Just as FORT itself must ask for the name of its input and output files, so must a FORTRAN program ask for the names of its disk files. FORTRAN-D programs do this by typing INPUT: and OUTPUT: a second time. The user responds by typing the name of the files to be read or written by the program. FORTRAN-D asks for both input and output for all programs which include a DEFINE DISK statement. If only input (or output) is to be used, the user responds to the other by typing the RETURN key.

Line Format

FORTRAN programs consist of a series of lines, each a string of 72 characters or less (the width of the Teletype paper from margin to margin). Each line contains two fields: the first, which begins at the left margin, is an identification field; the second contains the statement field. Termination of a line is indicated to the computer by a carriage return, accomplished by typing the RETURN key.

The identification field can be blank and extends from the left-hand margin up to and including a semicolon character. This field can contain one of the following types of identification:

1. A Statement number. This number, which can be any positive integer from 1 to 2047 inclusive, identifies the statement on that line for reference by other parts of the program. Statement numbers are used for program control or to assist the programmer in identifying segments of his program.

2. The letter, C. This identifies the remainder of the line as a comment.

Although the identification field may be left blank, it cannot be omitted entirely. The statement field begins immediately after a blank space and extends through the next carriage return. A sample FORTRAN program is shown below:

```
C       THIS PROGRAM CALCULATES FACTORIALS
5       TYPE 200
10      ACCEPT 300,N
        IFACT=1
30      IF (N-1) 5,32,33
32      TYPE 400,N,IFACT
        GO TO 10
33      DO 35 I=1,N
        IFACT=IFACT*I
35      CONTINUE
        GO TO 32
200     FORMAT (/,"PLEASE TYPE A POSITIVE NUMBER",/)
300     FORMAT (I)
400     FORMAT (/,I," FACTORIAL IS",IFACT)
        END
```

FORTRAN source programs are generated using the Symbolic Editor Program. The Editor will facilitate formatting lines by use of a tab character, permitting automatic movement to an indented second field.

STATEMENT NUMBERS

Each statement can have a positive, nonzero integer (0-2047) as its number. The statement number is used to reference that particular statement elsewhere in a program. A statement number consists of one to four digits beginning at the left hand margin and is followed by a semicolon. Statement numbers can be assigned nonsequentially; however, no two statements can have the same number. There must be no more than 40 statement numbers in a given program, and they must have a value of 2047 (decimal) or less.

STATEMENT CONTINUATION CHARACTER

Frequently, a statement is too long to fit on one line. If the character single quote (') appears as the last character of a line before the carriage return, the next line is treated as a continuation of the preceding statement. A statement may be continued on as many lines as necessary to complete it, but the maximum number of

characters in the statement cannot exceed 128 (about two formatted lines). For example:

```
10      A=B**2-(4.*A*C/(B**2+1.5*A*C))*4.3'
        +B**2+((SQTF(C)*SQTF(D))/(B**2+1.5*A*C))
```

is equivalent to the formula:

$$A = B^2 - \left(\frac{4 \cdot A \cdot C}{B^2 + 1.5 \cdot A \cdot C}\right) \cdot 4 \cdot 3 + B^2 + \left(\frac{\sqrt{C} \cdot \sqrt{D}}{B^2 + 1.5 \cdot A \cdot C}\right)$$

Although the continuation character, ('), allows a single statement to extend over two or more lines, no more than one statement can be written on one line.

FORTRAN Statements

FORTRAN statements are of several types with various functions distinguished as follows:

1. Comment statements allow a programmer to insert notes within the program.
2. Arithmetic statements resemble algebraic formulas. They define calculations to be performed.
3. Control statements govern the sequence of statement execution within a program. These statements reference program line numbers.
4. Specification statements allocate data storage and specify input/output formats.
5. Input/output statements control the transfer of information into and out of the computer.

COMMENT STATEMENTS

The character C, at the left margin of a line, designates that line as a comment line. A comment has no effect upon the compilation process but can serve as a guide to program logic for later debugging, etc. There is no limit to the number of comment lines which can appear in a given program. A comment cannot be continued by use of the continuation character, ('), but must be continued in a second comment statement. For example:

```
C       THIS IS AN EXAMPLE
C       OF A COMMENT
```

CHARACTER SET

The following characters are used in the FORTRAN language:

1. The alphabetic characters, A through Z.
2. The numeric characters, 0 through 9.
3. The control characters:

;	semicolon	CR	carriage return
.	period	LF	line feed
‘	single quote	(	left parenthesis
“	double quote	)	right parenthesis
,	comma		

4. The operators:

**	exponentiation	/	division
+	addition	*	multiplication
−	subtraction	=	replacement

All other characters are ignored by the Compiler except as Hollerith information found in FORMAT statements (where all Teletype characters are legal). The SPACE character has no grammatical function (it is not a delimiter) except in FORMAT statements and can be used freely to make a program easier to read.

CONSTANTS

Constants are explicit numeric values appearing in statements. Two types of constants, integer and real, are permitted in FORTRAN.

Integer Constants

Integer (fixed-point) constants are represented by a string of one to four decimal digits, written with an optional sign and without a decimal point. An integer constant must fall within the range ±2047. For example:

47	
+47	(+ sign is optional)
−2	
0434	(leading zeros are ignored)
−0	(same as zero)

Real Constants

Real constants are represented by a digit string, an explicit decimal point, and are written with an optional sign.[10] Real constants can also be written in exponential notation with an integer exponent to denote a power of ten (i.e., 7.2×10^3 is written 7.2E+3). A real constant may consist of any number of digits but only the leftmost six digits appear in the compiled program. Real constants must fall within the range $0.14 \times 10_{-38}$ to 1.7×10^{38}. For example:

+4.50	(plus sign is optional)
4.50	
−23.09	
−3.0E14	(same as -3.0×10^{14})
7	(saved as 7.00000, not the same as the integer 7)

Fixed and Floating-Point Representation

The difference between integers and real numbers in FORTRAN is the way in which each is represented in core memory. Both types of numbers are converted to binary to be stored in the computer.

A FORTRAN integer is stored in one 12-bit computer word. The sign of the number is kept in the high-order bit and the magnitude (the integer value) in the remaining 11 bits. This representation, shown schematically in Figure 2-1, is called fixed point, because the decimal point is always considered to be to the right of the rightmost digit. A FORTRAN integer may not exceed the range of ±2047. All integers greater than ±2047 are taken modulo 2048 (i.e., 2049 is considered to be 1; 4099, to be 3).

The floating-point format consists of two parts: an exponent (or characteristic) and a mantissa. The mantissa is a binary fraction with the radix point assumed to be to the left of bit one of the mantissa. The mantissa is always normalized; meaning it is stored with leading zeros eliminated so that the leftmost bit is always significant. The exponent represents the power of two by which the mantissa is multiplied to obtain the true value of the

[10] Where a number is to be identified as being negative, a minus sign (−) must be used. A plus sign (+) is optional; with no sign, a number is considered positive.

number for use in computation. The exponent and mantissa are both stored in two's complement form.

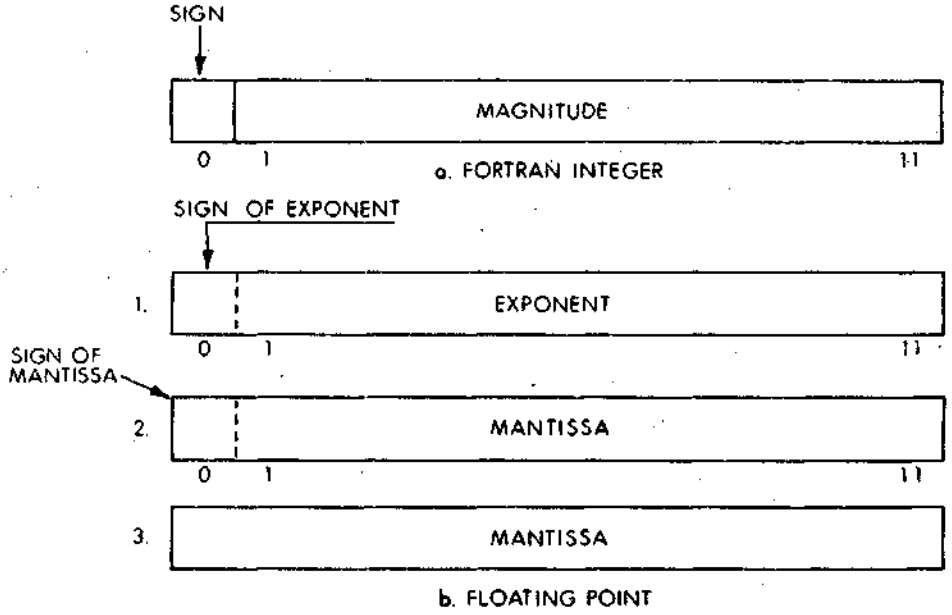


Figure 9-1. Number Representation

Users should not attempt to input floating point constants of more than six decimal digits, either in the FORTRAN source program or via the run-time ACCEPT statement.

Integers cannot appear in floating-point expressions except as exponent or subscripts. Some examples of illegal and legal expressions are as follows:

Expression	Legal	Mode
A(I)*B(J)**2	Yes	Floating
I(M)*K(N)	Yes	Fixed
16.*B	Yes	Floating
(K+16)*3	Yes	Fixed
A**(1+2)/B	Yes	Floating
8*A	No	—
4.*J	No	—
I+D	No	—

VARIABLES

A variable is a symbol whose value may change during execution of a program. The name of a variable consists of one or more alphanumeric characters, the first of which must be alphabetic.

Only the first four characters are interpreted as defining the variable name; the rest are ignored. For example, the name EPSILON would be interpreted by the Compiler as EPSI. Since only the first four characters are meaningful, the two names XSUM1 and XSUM2 would be considered identical.

Spaces, as mentioned earlier, are ignored. The name EX IT represents one variable, not two. Thus, EX IT, EXIT, or even EXI T are identical names as far as the Compiler is concerned, and they all refer to the same numerical quantity.

The type (or mode) of a variable (integer or real) is determined by the first letter of the variable name.

1. Integer variables begin with the letters:
   I, J, K, L, M, or N
2. Real variables begin with any letter other than those above.

Variables of each type may be either scalar or array variables, as explained below.

Integer Variables

Integer variables undergo arithmetic calculations with automatic truncation of any fractional part. For example, if the current value of K is 5 and the current value of J is 9, then J/K would yield 1 as a result.

Real Variables

A variable is real when its name begins with any character other than I, J, K, L, M, or N. Real variables undergo no truncation in arithmetic calculations. Real variables may be converted to integer variables, and vice-versa, across an equal sign.

Scalar Variables

A scalar variable, which may be either integer or real, represents a single number, as opposed to an array (below) representing a collection of numbers. For example, the following are scalar variables:

```
LM
G2
A
TOTAL    (considered to be TOTA by the computer)
J
```

Array Variables

An array variable represents a single element of a one-dimensional array of quantities. The variable is denoted by the array name followed by a subscript enclosed in parentheses. The subscript may be any combination of integer variables and integer constants forming a valid expression, as follows:

(V) (V+C) (V−C) (C)

where V is a fixed point (integer) variable, and C is a fixed-point constant (not equal to 0).

The value of the expression in parentheses determines the referenced array element. For example, the row matrix, A_1, would be represented by the subscripted variable A(1), and the second element in the row would be represented by A(2). Examples of array variables are:

Legal Forms	Illegal Forms
Y(1)	A(2+1)
A(K+2)	B(C)

DIMENSION STATEMENT

Array names must be identified as such to the FORTRAN Compiler. Two items of information must be provided in any program using arrays:

1. Which are the subscripted variables?
2. What is the maximum value of the subscript? (When an array is used, a certain amount of storage space must be set aside by the Compiler for the array elements.)

This information is provided by the DIMENSION statement:

```
DIMENSION A(20),B(15)
```

where A and B are array names, and the integer constants 20 and 15 are the maximum dimensions of each subscript.

The rules governing the use of array variables and the DIMENSION statement are as follows:

1. All array names must appear in a DIMENSION statement.

2. DIMENSION statements may be used more than once and may appear anywhere in the FORTRAN program, provided that the DIMENSION of an array appears before any statement which references the array.
3. Any number of arrays can be defined in a single DIMENSION statement.
4. For notes on how to implement double subscripts (i.e., A(I,J)), see the section on Implementation Notes.

Array variables may be either integer or real, depending upon the initial letter of the array name.

```
DIMENSION LIST(30),MAT(100),REGR(20)
```

In the statement above, the names LIST and MAT designate integer arrays; that is, all elements of both arrays are integers. The third name, REGR, designates a floating point, or real array. The first array is a list containing a maximum of 30 elements; the second array has a maximum of 100 elements.

The third array is a floating-point array and there are a maximum of 20 elements in it. Not all elements of an array need be used in the course of a program; but, if using the DIMENSION statement the variable LIST (3.1) could not be referenced without the occurrence of an error message.

FORTRAN Arithmetic

ARITHMETIC OPERATORS

The arithmetic operators are symbols representing the common arithmetic operations. The important rule about operators in the FORTRAN arithmetic expressions is that: every operation must be explicitly represented by an operator. In particular, the multiplication sign must never be omitted. A symbol for exponentiation is also provided since superscript notation is not available on a Teletype.

Normally, a FORTRAN expression is evaluated from left to right. like an algebraic formula. There are exceptions to this rule; certain operations are always performed before others, regardless of order. This priority of evaluation is as follows:

1.	Expressions within parentheses	( )
2.	Unary minus	−
3.	Exponentiation	**
4.	Multiplication or Division	* or /
5.	Addition or Subtraction	+ or −

The term "binding strength" is frequently used to refer to the relative position of an operator in a table such as the one above, which is in order of descending binding strength. Thus, exponentiation has a greater binding strength than addition, and multiplication and division have equal binding strength.

The unary minus is the arithmetic operator which indicates that a quantity is less than zero, such as −53, −K, −12.3. It refers only to the constant or variable which it precedes as opposed to a binary operator, which refers to operands on either side of itself as in the expression A−B. A unary minus is recognized by the fact that it is preceded by another operator, not by an operand. For example:

A + B**−2 / C−D

The first minus sign (indicating a negative exponent) is unary; the second (indicating subtraction) is binary. At present it is not possible to raise an integer variable or integer constant to an integer value with FORTRAN-D. Only real values can be raised to integer powers.

The left-to-right rule can be stated more precisely: A sequence of operations of equal binding strength is evaluated from left to right. To change the order of evaluation, parentheses are required. Thus, the expression A−B*C is evaluated as A−(B*C), not (A−B)*C. Examples of the left-to-right rule follow:

The expression:	Is evaluated as:
A/B*C	(A/B)*C
A/B/A	(A/B)/C
A**B**C	(A**B)**C

Use of Parentheses

Note the use of parentheses in the example below. They are used to enclose the subscript of the dimensional variable, D; to

specify the order of operations of the expression involving A, B, and C; and to enclose the argument of the function SINF.

```
D(I) + (A+B)**C + SINF (X)
```

In algebra there are several devices, such as square brackets [ ], rococo brackets { }, etc., for distinguishing between levels when expressions are nested. In FORTRAN, only the parentheses are available, so the programmer must be especially careful to pair parentheses properly. In any given expression, the number of left parentheses must be equal to the number of right parentheses.

An easy way to check the proper pairing of parentheses is by counting out, illustrated in the following example:

```
(Z+AM* (AM+1.))/( (X**2+C**2)*P)
 1      2       10 12          1  0
```

The procedure is this: Reading the expression from left to right, assign the number, 1, to the first left parenthesis (if you encounter a right parenthesis first, the expression is already wrong). Increase the count by one each time a left parenthesis is read, and decrease the count by one when a right parenthesis is found. When the expression has been completely scanned, the count should be zero. If it becomes less than zero during the scanning, there are too many right parentheses. If it is greater than zero at the end of an expression, there are excess left parentheses.

ARITHMETIC EXPRESSIONS

An algebraic formula such as:

$$5a + 4b(x^2 - x_0)$$

represents a relationship between symbols (a, b, x, x_0) and constants (5, 4, 2) indicated by mathematical functions and arithmetic operators (+, −, multiplication, exponentiation). This same formula can be written as a FORTRAN arithmetic expression with very little change in appearance:

```
(5.*A + 4.*B* (X**2 - XZRO)
```

The construction of both expressions is the same; the differences are notational.

Elements of an arithmetic expression are of four types: constants, variables, operators, and functions. An expression may consist of a single constant or variable or a string of constants, variables, and functions connected by operators.

Examples of arithmetic expressions follow; each expression is shown with its corresponding algebraic form.

Algebraic Expression	FORTRAN Expression
$\frac{2\sqrt{x}}{3}$	2.*SQTF(X)/3.
$\frac{3x\pi - 2(x+y)}{4.25}$	(3.*X*PI−2.* (X+Y)) /4.25
$a \cdot \sin\theta + 2a\cos(\theta - 45)$	A*SINF(THTA)+2.*A*COSF-(THTA−0.78540)
$\frac{(a^2 - b^2)}{(a+b)^2}$	(A**2−B**2) / (A+B)**2

ARITHMETIC STATEMENTS

The arithmetic statement relates a variable, V, to an arithmetic expression, E, by means of the replacement operator, (=):

V=E

Such a statement looks like a mathematical equation, but is treated differently. The equal sign is interpreted in a special sense; it does not represent a relationship between left and right members, but rather specifies an operation to be performed.

In an arithmetic statement, the value of the expression to the right of the equal sign replaces the value of the variable on the left. This means that the value of the left-hand variable will change after the execution of an arithmetic statement. A few illustrations of arithmetic statements are given below:

1. VMAX = VO + AXT
2. T = 2.*PI*SQTF(1./G)
3. PI = 3.14159
4. THTA = OMGA + ALPH*T**2/2.
5. MIN = MIN0
6. INDX = INDX + 2

With the interpretation of the equal sign stated above, Example 6 becomes meaningful as an arithmetic statement. If, for example, the value of INDX is 40 before the statement is executed, its value will be 42 after execution.

In arithmetic expressions, a binary operator requires an operand on its left and right. The equal sign of an arithmetic statement is also considered to be a binary operator, as demonstrated in the following revised table of operators:

Operator	Use	Interpretation
− (Unary)	−A	negate A
**	A**B	raise A to the Bth power
*	A*B	multiply A by B
/	A/B	divide A by B
+	A+B	add B to A
− (Binary)	A−B	subtract B from A
=	A=B	replace the value of A with the value of B

The replacement operator is considered to have the lowest binding strength of all the operators; therefore, the expression on the right is evaluated before the operation indicated by the equal sign is performed.

Multiple Replacement

An important result of treating the equal sign as an operator is that operations can be performed in sequence. Just as there can be a series of additions, A+B+C, there can also be a series of replacements:

A=B=C=D

Notice that because the operand to the left of an equal sign must be a variable, only the rightmost operand, represented by D in the example, may be an arithmetic expression. The statement is interpreted as follows: "Let the value of the expression D replace the value of the variable C, which then replaces the value of the variable B" and so on. In other words, the value of the rightmost expression is given to each of the variables in the string to the left. A common use for this construction is in setting up initial values:

```
XZRO=SZRO=AXRO=0
T=T1=T2=T3=60
P=FP=4. *ATM−AK
```

Only simple variables will compile correctly in this manner. For example, statements of the type A(1)=A(2)=R(1)=0.123 are not allowed and will not compile properly (subscripted variables may not be used in multiple replacement statements).

Multiple replacement done in a single statement must not contain mixed mode variables. That is:

A=B=C=10	compiles correctly
I=J−7	compiles correctly
A=J=7	does not compile correctly

Mode Conversion

Another useful result in treating the equal sign as an operator is that the value of an expression on the right of an equal sign is converted to the mode of the left-hand variable, if necessary, before storage. For example:

A=M Stores the value of M as a floating-point number in A
K=B Stores the value of B (truncated) as an integer number in K

If B = 4.75 and M = 7, the conversion above will result in the following values being assigned:

A = 7.00000
K = 4

FUNCTIONS

Functions are used in FORTRAN just as they are in ordinary mathematics, acting as variables in arithmetic expressions. The function name represents a call to a special subprogram which performs the calculations to evaluate the function; the result is used in the computation of the expression in which the function occurs. FORTRAN-D provides several mathematical functions: square root, sine, cosine, arctangent, exponentiation, and natural logarithm.

The argument of a function can be a simple variable, a subscripted variable, or an expression. The argument must be in a floating-point format. FORTRAN recognizes a symbol as a function when it is a predefined symbol ending in F and followed by an argument enclosed in parentheses (if the F is missing from the term, the symbol is treated as a subscripted variable). The argument of a function can consist of another function or groups of functions. For example, the expression:

LOGF(SINF(X/2)/COSF(X/2))

is equivalent to log $\tan\left(\frac{X}{2}\right)$

FORTRAN-D contains the following functions:

Function Name	Meaning
SQTF(X)	Square root of X
SINF(X)	Sine of X, where X is expressed in radians
COSF(X)	Cosine of X, where X is expressed in radians
ATNF(X)	Arctangent X, where X is expressed in radians
EXPF(X)	Exponential of X
LOGF(X)	Logarithm of X

Program Control

In this section, FORTRAN statements are discussed in the context of program sequences. FORTRAN statements are executed in the order in which they are written unless instructions are given to the contrary by use of the program control statements. These statements allow the programmer to alter sequence, repeat sections, suspend operations, or bring the program to a complete halt.

END STATEMENT

END occurs alone on a line and indicates the physical end of the program to the FORTRAN Compiler. It can be preceded by a line number. Every program must contain an END statement.

STOP STATEMENT

A program arranged so that the last written statement is the final and only stopping place needs no other terminating indication; the END statement automatically determines the final halt. Many programs, however, contain loops and branches so that the last executed statement can be somewhere in the middle of the written program. Frequently there is more than one stopping point. Such terminations are indicated by the STOP statement. This causes a final, complete halt; no further computation is possible, although the program may be completely restarted from the beginning.

When a STOP is encountered during program execution, the system signifies that a STOP has occurred by outputting an exclamation point (!) to the Teletype or high-speed punch, whichever is being used as the output device.

PAUSE STATEMENT

The STOP statement prevents further computation after it has been executed. There is a way, however, to suspend operation for a time and then restart the program. This procedure is frequently necessary when the user must do such tasks as loading and unloading paper tapes in the middle of a program. This kind of temporary halt is provided by the PAUSE statement. The PAUSE statement halts the program and returns control to the EduSystem 50 Monitor. The user may then perform any necessary manipulations and restart the program by typing the Monitor command START.

GO TO STATEMENT

There are various ways in which program flow may be directed. As shown schematically in Figure 9-2, a program may be a straight-line sequence (1), or it may branch to an entirely different sequence (2), return to an earlier point (3), or skip to a later point (4). The blocks represent sections of FORTRAN code. The lines indicate the path which control takes as the program executes.

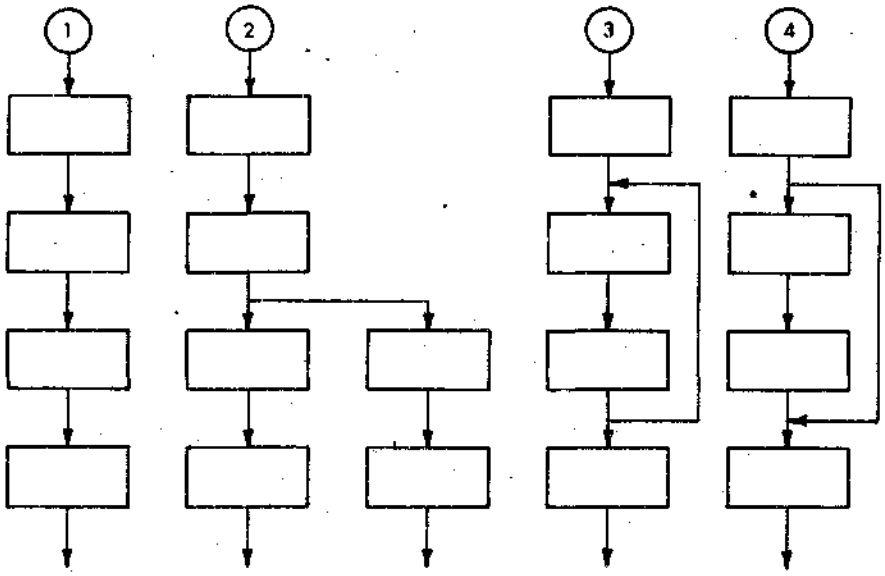


Figure 9-2. Program Flow

All of these branches can be performed in several ways, the simplest of which uses the statement:

GO TO n

where n is a statement number in the program. The use of this statement is described in the following example, which also illustrates the construction of a loop, the name given to program branches of the type shown in the example above.

EXAMPLE OF INTEGER SUMMATION

In the following example, the sum of successive integers is accumulated by repeated addition. The main computation is provided by the three-instruction loop beginning with Statement 2. The statements preceding this loop provide the starting conditions, called the initialization. The partial sum is set to zero, and the first integer is given the value of one. The loop then adds the integer value to the partial sum, increments the integer, and repeats the operation.

```
C       SUM OF FIRST N INTEGERS BY ITERATION
        KSUM=0
        INUM=1
2       KSUM=INUM+KSUM
        INUM=INUM+1
        GO TO 2
        END
```

IF STATEMENT

The program shown in the preceding example performs the required computation, but note that the loop is endless. To get out of the loop the user must know when to stop the iteration and what to do afterwards.

The IF statement fills both requirements. It has the following form:

IF(E) K,L,M

where E is any variable name, arithmetic expression, or arithmetic statement, and K, L. and M are statement numbers. The statement is interpreted in this way:

If the value of $E < 0$, GO TO statement K
$E = 0$, GO TO statement L
$E > 0$, GO TO statement M

Thus, the IF statement decides when to stop a loop by evaluating an expression. It also provides program branch choices with the transfer of control, depending on the results of the evaluation of E. For example:

```
C       SUM OF THE FIRST 50 INTEGERS
        KSUM=0
        INUM=1
2       KSUM=INUM+KSUM
        INUM=INUM+1
        IF (INUM-50)2,2,3
3       STOP
        END
```

In the foregoing example, the initialization and main loop are the same as for the example in Figure 9-2 except that the GO TO statement of the earlier program has been replaced by an IF statement. The IF statement says, "if the value of the variable INUM is less than, or equal to, 50 (which is the same as saying that if the value of the expression INUM-50 is less than or equal to zero), transfer control to Statement 2 and continue the computation. If the value is greater than 50, stop." (See the section on Implementation Notes for an alternate solution.)

A loop may also be used to compute a series of values. The following illustration is an example of a program to generate terms in the Fibonacci series of integers, in which each succeeding member of the series is the sum of the two members preceding it:

```
C       FIBONACCI SERIES, 100 TERMS
        DIMENSION FIB(100)
        FIB(1)=1.0
        FIB(2)=1.0
        K=3
5       FIB(K)=FIB(K-1)+FIB(K-2)
6       K=K+1
        IF (K-100)5,5,10
10      STOP
        END
```

In this program, the initialization includes a DIMENSION statement which reserves space in storage, and two statements which provide the starting values necessary to generate the series. Each time a term is computed, the subscript is incremented so

that each succeeding term is stored in the next location of the table. As soon as the subscript is greater than 100, the calculation stops.

DO LOOPS

Iterative procedures such as the program loop are so common that a more concise way of presenting them is warranted. Three statements are required to initialize the subscript, increment it, and test for termination. The following type of statement combines all these functions:

DO n l=K1, K2, K3

here n is a statement number, l is a simple (non-subscripted) integer variable, and K1, K2, and K3 are simple integer variables or integer constants which provide, in order, the initial value to which l is set, the maximum value of l for which the loop will be executed, and the amount by which l is incremented at each return to the beginning of the loop. If K3 is omitted from the statement it is assumed to be one (1). Statement n must be a CONTINUE statement.

```
C       FIBONACCI SERIES, 100 TERMS
        DIMENSION FIB(100)
        FIB(1)=1.0
        FIB(2)=1.0
        DO 5 K=3, 100
        FIB(K)=FIB(K-1)+FIB(K-2)
5       CONTINUE
        STOP
        END
```

In words, the DO statement says "Execute all statements through Statement 5 with K=3; when Statement 5 is encountered, perform the following test: If K+1 is less than or equal to 100, set K=K+1 and continue the program by executing the first statement after the DO statement. If K+1 is greater than 100, the next sequential statement following Statement 5 is executed."

DO loops are commonly used in computations with subscripted variables. In such cases, it is usually necessary to perform the loops within loops. Such nesting of DO loops is permitted in FORTRAN.

```
C       FIRST LOOP
        DO 10 I=1,20
        X(I)=0
C       NESTED LOOP FOLLOWS
        DO 5 K=2,40,2
        X(I)=X(I)+B(K)-Z(K))**2
5       CONTINUE
C       END OF NESTED LOOP
        A(I)=X(I)**2+C(I)
10      CONTINUE
```

Sequential elements in the array X(1) are formed by summing the square of the difference of every second element in the B and Z arrays. Then the array A(1) is formed by summing every element in the array C(1) and the square of every element in the array X(1). The algebraic expression for the loop is as follows:

$$A_i = x_i^2 - C_i \qquad \text{for } i=1,2,3,\ldots 20$$

where

$$x_i = \sum_{k=2}^{40} (b_k - z_k)^2 \qquad \text{for } k=2,4,6,\ldots 40$$

The following three rules about DO loops must be observed:

1. DO loops may be nested, but they may not overlap. Nested loops may end on the same statement, but an inner loop may not extend beyond the last statement of an outer loop. Figures 9-3 schematically illustrates permitted and forbidden arrangements.

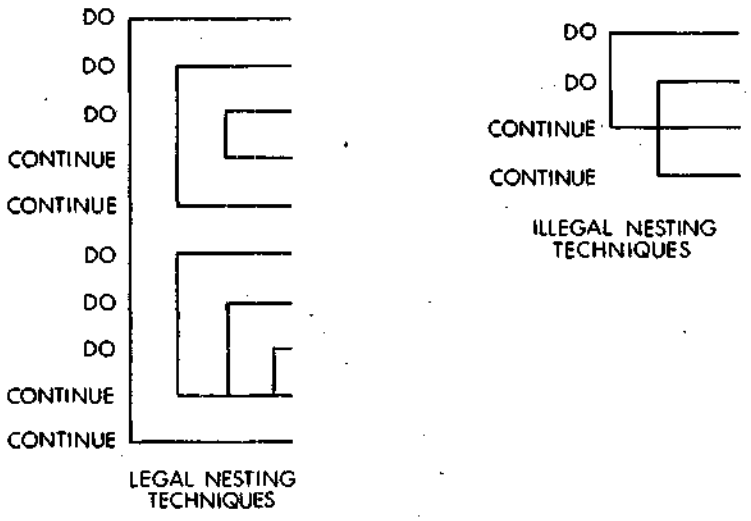


Figure 9-3. Legal and Illegal Nesting Techniques

2. If the user transfers into the range of a DO loop, the value to be incremented (I, for example) is not automatically initialized as specified in the DO statement. Transferring into the range of a DO loop is allowed as long as:
   a. Control was originally transferred out of the DO loop by some means other than by completing it.
   b. Incrementing and testing start with the current value of I at the time control returns to the loop.
3. A DO loop must end on a CONTINUE statement.

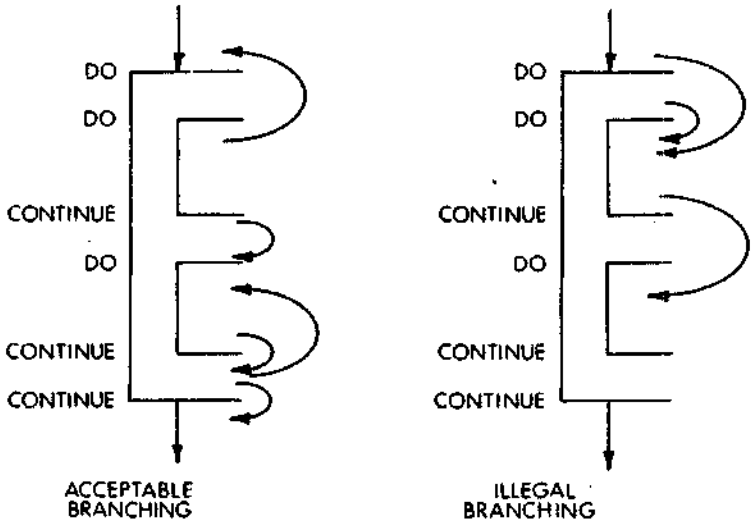


Figure 9-4. Program Branching in DO Loops

CONTINUE Statement

A special statement (CONTINUE) is provided which is not an executable statement itself, but provides a termination for all DO loops. DO loops must be terminated on a CONTINUE statement. The CONTINUE statement is identified with the line number given in the DO statement. For example:

```
        DO 37 MM=1,10
        IF (X(MM)-100.) 37,42,37
37      CONTINUE
        GO TO 102
42      STOP
```

A single CONTINUE statement can be referenced more than once in a single DO loop or can serve as the terminating line for two or more nested DO loops.

COMPUTED GO TO

The GO TO statement previously described is unconditional and provides no alternatives. The IF statement offers a maximum of three branch points. One way of providing a greater number of alternatives is by using the computed GO TO, which has the following form:

GO TO (K1,K2,K3,...,Kn),J

where K_n is statement number, and J is a simple integer variable, which takes on values of 1,2,3,...n according to the results of some previous computation. For example:

```
IVAR=I4*J/2+K
GO TO (5,7,5,7,5,7,10),IVAR
```

causes a branch to Statement 5 when IVAR=1, 3, or 5; to Statement 7 when IVAR=2, 4, or 6; and to Statement 10 when IVAR=7. When IVAR is less than 1 or greater than 7, the next sequential statement after the GO TO is executed.

FORTRAN Input/Output

So far, we have assumed that all information (programs, data, and sub-programs) is in memory, without regard to how it is put there. Programs are read by a special loader, but the programmer is responsible for the input of data and the output of results by including directions for I/O operations in his program.

For any input/output procedure, several questions must be answered:

1. In which direction is the data going? The data coming in is being read into memory; information going out is being written on whatever medium is specified.
2. Which device is being used? Information may be transferred between core and whatever input/output devices are available; each I/O operation must specify the device involved.

3. Where in core memory is the data coming from or going to? The amount of data and its location in the computer storage must be specified.
4. In what mode is the data represented? In addition to floating and fixed-point modes for numeric data, there is the Hollerith mode for transferring alphanumeric or text information.
5. What is the arrangement of the data? The format of incoming or outgoing data must be specified.

For every data transfer between core memory and an external device, two statements are required to provide all of the information listed above. The first three items are specified by the input/output statement, and the last two items are determined by the FORMAT statement.

DATA FORMATS

FORTRAN-D provides for communication of data to and from a program in the following ways:

ASCII Coded Data

The Teletype can be used to transfer data to the program either via the keyboard (in which case the user types the data) or from previously punched paper tape (read via the Teletype tape reader). Data can be output from a program to the Teletype, producing a printed copy with or without the corresponding punched paper tape. The high-speed reader and punch can also be used for data transfer via punched paper tape. No printed copy is made when output is to the high-speed punch.

Binary Coded Data

Disk and DECtape can also be used for data transfer, in which case the data is stored as a core image on tape in 128-word blocks of 12-bit binary words. Integers are read and written as single 12-bit words, floating-point numbers as three words. Alphanumeric information is transmitted as 8-bit ASCII coded characters right-justified in 12-bit words (one character per word).

INPUT/OUTPUT STATEMENTS

Input/Output statements control the transfer of information. As illustrated below, I/O statements consist of three basic items of information: the device being accessed and the direction of transfer,

the number of the FORMAT statement controlling the arrangement of data, and the list of variable names whose values are to be output or changed by new input.

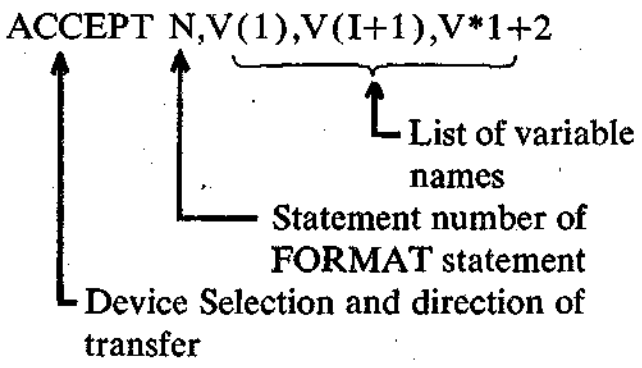
NOTE

There is a restriction on subscripted variables when used with I/O statements. Subscripts to be used with I/O statements must be of the form: LL, where each L is a letter, and not of the form LD, where D is a digit. For example:

```
DO 10 L1=1,4
ACCEPT 7,A(L1)
```

will not store information correctly. The statement should read:

```
DO 10 LL=1,4
  ACCEPT 7,A(LL)
```



ACCEPT and TYPE Statements

ACCEPT and TYPE transfer information between the Teletype and EduSystem 50. ACCEPT causes information to be read into core memory from either the keyboard, the Teletype paper tape reader, or the high-speed reader if it is assigned before calling FORTRAN-D. ACCEPT is especially convenient if data is to be entered at the keyboard since it automatically supplies line feed when the RETURN key is typed. Also, the user may correct an erroneously typed value by typing the RUBOUT key.

TYPE causes information to be transferred from core memory to the Teletype printer, the Teletype paper tape punch, or the high-speed punch if it is assigned before calling FORTRAN-D.

If the user needs the high-speed reader and punch for I/O, he must assign the devices for his use before calling the FORTRAN compiler (FORT) or operating system (FOSL). Once logged into EduSystem 50, he replies to Monitor's dot with the appropriate assign statements. For example:

```
.ASSIGN P
P ASSIGNED

.ASSIGN R
R ASSIGNED
```

The Monitor replies with P ASSIGNED and R ASSIGNED in response to the user. If the device requested is not available (being used by someone else), Monitor replies JOB XX HAS X, where XX is the job number assigned the device X. The high-speed reader/punch is device code 2. If running several programs, the user should reassign the devices before each run.

READ and WRITE Statements

EduSystem 50 FORTRAN also allows programs to read and write data files on disk. These data files are completely separate from program files. Data files are read and written by standard READ and WRITE statements within the FORTRAN program. The device code for the disk is 3. Since programs which use disk are treated differently by FORT from those which do not use disk, it is necessary to identify programs which do. This is done by placing a DEFINE DISK statement as the first statement in any FORTRAN program which includes a READ or WRITE statement with a device code of 3.

Just as FORT must ask for the names of its input and output files, so must a FORTRAN program ask for the names of its disk files. FORTRAN programs do this by typing INPUT: and OUTPUT: a second time. The user responds by typing the names of the data files to be read or written by the program. FORTRAN will ask for both INPUT and OUTPUT for all programs which include a DEFINE DISK statement. If only one is to be used, respond to the other by typing the RETURN key.

VARIABLE SPECIFICATION IN I/O STATEMENTS

Following the instruction that selects the device and direction of transfer is the statement number of the FORMAT statement that

controls the arrangement of the information being transferred. For example:

```
        ACCEPT 10,A
10      FORMAT (E)
```

Every I/O statement must have a reference to a FORMAT statement.

The final item specified in the I/O statement is the list of variables. This is a sequential list of the names of variables and array elements whose values are to be transferred in the order indicated. There is no restriction on the number of names which may appear in the list of an I/O statement, as long as the total statement length does not exceed 128 characters. The modes of the variables named need not agree with the corresponding FORMAT statement; however, the modes specified in the FORMAT statement take precedence. For example, where A=3.2, J=27, KAL=302, and BOB=7.58:

```
        TYPE 23,A,J,KAL,BOB
23      FORMAT(1,E,1,E)
```

The decimal portion of A is dropped and the number 3 is printed as an integer; the value of J is printed as a normalized number; KAL is printed as an integer; and BOB is printed as a normalized number. The output would look like the following:

```
+ 3   +0.270000E+2  +  302  +0.758000E+01
```

NOTE

In READ and ACCEPT statements, although the number is read according to the FORMAT statement, it is stored according to the mode of the variable. For example:

```
5       ACCEPT 10,A
10      FORMAT (I)
```

causes the number 12.3 typed by the user to be read as 12 and stored as 0.120000E+2.

Array names included in I/O lists must be subscripted in one of the following forms:

A(V) A(V+C) A(V−C) A(C)

where A is the array name, V is a simple integer variable and C is a positive nonzero integer constant.

```
      TYPE 10,A,I,B,C(I+1),N(J+1)
10    FORMAT (E,I,E,/)

      A=A+C(J)**2-C(N)**2)

      TYPE 10,A,J,B,C(N)
```

If the list contains more names than there are elements in the FORMAT statement, the FORMAT statement is reinitialized when the elements are exhausted. The first element in the FORMAT statement then corresponds to the next name in the list. For instance, in the preceding example when the value of the variable, B, is printed in the E format, the control character, slash (/), causes a carriage return/line feed to occur. Then the FORMAT statement is reinitialized, and the array element, C(I+K), is printed in the E format and the array element N(J+L) in the I format.

The list does not have to exhaust the elements of a FORMAT statement. If there are fewer names in the list than there are elements in the FORMAT statement, the program completes the I/O operation and proceeds to the next sequential FORTRAN statement. If this next statement is another I/O statement that references a previously unexhausted FORMAT statement, that FORMAT statement is reinitialized. FORMAT statements are reinitialized when they are referenced or when all of their elements are exhausted.

FORMAT STATEMENT

The FORMAT statement controls the arrangement and mode of the information being transferred. The values of names appearing in the I/O statement list are transferred in the mode specified by the corresponding element in the FORMAT statement. These controlling elements consist of the characters E, I, A, slash (/),

and quote ("). The set of elements must be enclosed in parentheses and separated by commas. For example:

```
FORMAT (A,E,I,/,"HOLLERITH")
```

The control elements E and I are used for defining the mode of the data being transferred. When a variable is transferred in the E format, it is stored or output in floating-point form. If the variable is transferred in the I format, it is stored or output in fixed-point or integer form. Mode conversion on input or output can be accomplished because the elements in the FORMAT statement define the mode of the data. The mode of the original variable is overridden where necessary. For example:

```
        TYPE 10,A
10      FORMAT (I)
```

The variable, A, is printed as an integer, and the fractional part of A is truncated. If A has a value of 14.96, only the integer part, 14, is printed. If A has an absolute value of less than one, zero is printed.

THE A FORMAT SPECIFICATION

The control element, A, is used for defining the alphanumeric mode of data I/O. When a variable is to be assigned an alphanumeric value, data is read one character per variable. FORTRAN ignores CTRL/C, blank tape, RUBOUT, and 0200 code (leader/trailer tape). FORTRAN does not see the form-feed character when input is from the disk. The decimal equivalent of the ASCII value of the character is assigned to the variable. For example:

$$A = 301 \quad (\text{ASCII}) \quad = 192 \text{ (decimal)}$$

Any variable assigned the alphanumeric value, A, would be set equal to 192.

It is possible to do arithmetic with integer variables assigned alphanumeric values. For example:

```
        DO 10 J=1,5
        ACCEPT 12, K(J)
        IF (K(J)-141) 10,40,100
10      CONTINUE
12      FORMAT(A)
```

where the IF statement tests to see if the last character read is a carriage return (which is ASCII 215 or 141 decimal); if so, control transfers to Statement 40; if not, control stays within the DO loop.

It is not possible to do arithmetic with real variables assigned alphanumeric values. Output in alphanumeric format converts the value of the variable into an ASCII character and prints that character. For example:

```
12      FORMAT(A)
        DO 20 I=1,5
        TYPE 12, A(J)
20      CONTINUE
```

If the variables A(1) through A(5) were not originally assigned alphanumeric values, the results of the output can be meaningless.

INPUT FORMATS

Input data words can only consist of a sign, a decimal value, an exponent value if the data is floating-point, and a field terminating character such as space. Any character that is not a number, decimal point, sign, or E can be used to terminate a field except the RUBOUT character. When typing data, any number of spaces or other nonnumeric characters can be typed before the sign or decimal value in order to make the hard copy more readable.

Input data can be transferred into core memory from either the Teletype paper-tape reader, the keyboard, the high-speed reader, or DECtape. Input can be entered in either fixed- or floating-point modes (integers or decimal numbers). The mode in which data is stored in core memory is controlled by the first letter of the variable name. The characters read into core are determined by the corresponding element in the FORMAT statement.

Integer Values—the I Format

An integer data field consists of sign[11] and up to six decimal characters. Some examples of integer values are as follows:

---

[11] Plus sign can be represented by a plus or space character. Minus is represented by a minus character. If a sign character is absent from the data word, the data is stored as positive.

Typed Numbers	Values Accepted
−2001	−2001
−40	−0040
−0040	−0040
16	0016
−2041	2047

Real Values—the E Format

A floating-point input word consists of a sign, the data value up to six decimal characters, an E if an exponent is to be included, the sign of the exponent, and the exponent (i.e., the power of ten by which the data word is multiplied). For example:

ddd.dddEnn

The d's represent numerical characters in the data and the n's represent the 2-digit power of ten of the exponent (preceded by a sign). Either the sign, the decimal point, or the entire exponent part can be omitted. If the sign is omitted, the number is assumed to be positive; if the decimal point is omitted, it is assumed to appear after the rightmost decimal character. If the exponent is omitted, the power of ten is taken as zero.

Some examples of floating-point values are as follows:

Typed Numbers	Values Accepted
16.	0.16×10^2
.16E02	0.16×10^2
1600.E-02	0.16×10^2

OUTPUT FORMATS

E and I Formats

Integer values are always printed as the sign and a maximum number of four characters with spaces replacing leading zeros. On output, integers are left justified within the stated field. Sufficient trailing spaces are printed to fill the field followed by one additional space.

Floating-point values are printed in a floating-point format which consists of sign, leading zero, decimal point, six decimal characters, the character E, the sign of the exponent (minus or plus), and an exponent value of two characters. For example:

Integer Values	Output Format
−1043	−1043
−0016	− 16
+0016	+ 16

Floating-point values are printed as follows:

S0.dddddd Esnn

where: S represents the sign, minus sign, or space

dddddd represents six decimal digits of the data word
E indicates exponential representation
s represents the sign of the exponent value
nn represents the exponent value

Some examples of floating-point output are:

Decimal Value	Output Format
−8,388,608.0	−0.8388608E+07
−.000119209	0.119209E−03

FORMAT Control Specifications

In most cases when data is to be presented, it must be labeled and arranged properly on a data sheet. In order that this can be accomplished with FORTRAN, a provision has been made so that text information and spacing can be printed along with the data words. These features are provided by the special FORMAT control elements quote (") and slash (/). The slash character causes a return to the left margin.

Hollerith Output

When text information is enclosed in quotes and is contained as part of a FORMAT statement, it is output to the specified device as it appears in the statement. This output occurs when a TYPE or WRITE statement references a FORMAT statement containing text, and all other elements of the FORMAT statement previous to the text have been used. All legal Teletype characters (other than the quote character itself) can be contained within quotes and output as text.

```
      TYPE 10
10    FORMAT(/,"THIS IS HOLLERITH",/)


      TYPE 100,AMIN,AMAX
100   FORMAT(/,"MINIMUM=",E,/,"MAXIMUM=",E,/)


      TYPE 210
210   FORMAT(/,/,"    CUMULATIVE DISTRIBUTION",/,/'
      "       INCREMENTS       FREQUENCY",/)
      DO 220 K=1, 100
      TYPE 250,K,VALU(L),VALU(K+1),COUNT(K)
      CONTINUE
250   FORMAT (I,"",E," ",E,/)
```

Implementation Notes

DOUBLE SUBSCRIPTS

This version of FORTRAN does not have the facility for double-subscripted variables. To accomplish double subscripting, the programmer has to include indexing statements in the source program as illustrated below. In this example, the matrices are stored columnwise in memory; that is, sequential locations in memory are used as follows:

Element	Relative Position in Memory (INDX)
a11	1
a21	2
a31	3
a41	4
a51	5
a61	6
a12	7
a22	8
.	.
.	.
.	.
a56	35
a66	36

If referencing Element a56 in the array, M=5, N=6, (I=6 for a 6 by 6 array), and INDX=M+1*(N−1)=5+1*5=35. If referencing Element a22, INDX=2+6*1=8.

```
C       MATRIX MULTIPLY PROGRAM
        DIMENSION A(36),B(36),C(36)
C       ACCEPT DIMENSION OF ARRAY
        ACCEPT 1,I
1       FORMAT(I)
        DO 10 M=1,I
        DO 10 N=1,I
        INDX=M+I*(N-1)
C       ACCEPT FIRST MATRIX
        ACCEPT 1,A(INDX)
2       FORMAT(E)
10      CONTINUE
        TYPE 15
15      FORMAT(/,/,/)
        DO 20 M=1,I
        DO 20 N=1,I
        INDX=M+I*(N-1)
C       ACCEPT SECOND MATRIX
        ACCEPT 1,B(INDX)
        C(INDX)=0
20      CONTINUE
C       MULTIPLY MATRICES
        DO 30 M=1,I
        DO 30 N=1,I
        DO 30 K=1,I
        IC=N+I*(M-1)
        IA=K+I*(M-1)
        IB=K+I*(K-1)
        C(IC)=C(IC)+A(IA)*B(IB)
30      CONTINUE
        TYPE 15
C       PRINT RESULTS IN MATRIX FORM
        TYPE 21
        DO 40 M=1,I
        TYPE 21
        DO 40 N=1,I
        INDX=N+I*(M-1)
        TYPE 1,C(INDX)
40      CONTINUE
21      FORMAT(/)
        TYPE 15
        END
```

SUBSTATEMENT FEATURE

The most important result of treating the equal sign as a binary operator (as explained in the section on Arithmetic Statements) is that it may be used more than once in arithmetic statement. In addition to simple replacement operations (see section on Multiple Replacement), consider the following:

CPRM = (CKL – CKG) / (CPG = P*(Q+1))

The internal arithmetic statement (or substatement). CPG P* (Q+1). is set off from the rest of the statement by parentheses. The complete statement is a concise way of expressing the following common type of mathematical procedure:

$$\text{Let:} \quad C^1 = \frac{C_{kf} - C_{kg}}{C_{pg}}$$

Where: $C_{pg} = p.(q+1)$

The stating of a relation followed by the conditions for evaluating any of the variables can be expressed in a single arithmetic statement in FORTRAN.

A second use of the equal sign is shown below. For background on this short program, see the discussion of the same problem in the section on the IF statement.

```
C       SUM OF THE FIRST 50 INTEGERS
        KSUM=0
        INUM 50
2       KSUM=INUM+KSUM
        IF(INUM+INUM-1)3,3,2
3       STOP
```

In this example, the sum is formed by counting down, but the same results are achieved as in the section on the IF statement. The initialization is changed so that INUM starts with the value of 50 instead of 0, and the statement, INUM=INUM+1, is no longer required.

ERROR CHECKING

Because of the extremely compact nature of the FORTRAN-D Compiler, either FORTRAN features or error checking will suffer. In the case of FORTRAN-D, checking for certain errors is not as important as preserving the language. Therefore, the programmer is advised to follow the rules as stated in this manual and carefully check his program for mistakes. For example, the statement

```
A = B + C -
```

will compile, although at execution time it will give unpredictable results.

It should be noted that data areas must not extend below location 5600 in FORTRAN-D. No diagnostic is issued unless program and data areas actually overlap. A maximum of 896_{10} words are available for data. Care should be taken not to exceed the limits through use of large arrays, etc. Similar obvious errors are accepted by the Compiler; their effects are often unpredictable.

FORTRAN-D SOURCE PROGRAM RESTRICTIONS

The following limits are imposed upon all FORTRAN-D source programs:

1. Not more than 896 data cells. This includes all dimensional variables, user-defined variables, constants, and all constants generated by the usage of a DO loop.
2. Not more than 20 undefined forward references to unique statement numbers per program. An undefined forward reference is a reference to any statement label that has not previously occurred in the program. Multiple references to the same undefined statement numbers are considered as one reference.
3. Not more than 64 different variable names per program.
4. Not more than 128 characters per input statement.
5. Not more than 40 numbered statements per program.

FORTRAN-D COMPILER AND OPERATING SYSTEM CORE MAP

The Compiler occupies the following core locations:

0003-7600	Compiler plus tables
7200-7600	Compiler tables (undefined forward reference tables, etc.)

The Operating System occupies locations:

0000-5200	Operating System for paper I/O
0000-6000	Operating System for disk I/O

Locations 5201 through 7576 are available for the user's program when using paper tape input/output.

Table 9-10. FORTRAN-D Statement Summary

Statement and Form	Explanation
Arithmetic Statements	
$v = e$	v is a variable (possibly subscripted); e is an expression.
Control Statements	
GO TO n	n is a statement number.
GO TO $(n_1,n_2,...,n_n),i$	$n_1,...,n_n$ are statement numbers; i is a nonsubscripted integer variable.
IF (e) n_1,n_2,n_3	e is an expression; n_1,n_2,n_3 are statement numbers.
DO n $i=k_1,k_2,k_3$	n is the statement number of a CONTINUE; i is an integer variable; k_1,k_2,k_3 are integers or nonsubscripted integer variables.
CONTINUE	Proceed
PAUSE	Temporarily suspend execution.
STOP	Terminate execution.
END	Terminate compilation; last statement in program.
Specification Statements	
DIMENSION $v_1(n_1)$, $v_2(n_2),...,v_n(n_n)$	$v_1,...,v_n$ are variable names; $n_1,...,n_n$ are integers.
DEFINE device	Device is DISK or TAPE.
FORMAT $(s_1,s_2,...,s_n)$	s is a data field specification.
COMMENT	Designated by C as first character on line.
Input/Output Statements	
ACCEPT f, list	f is a FORMAT statement number; list is a list of variables; input is from the Teletype.
TYPE f, list	f is a FORMAT statement number; list is a list of variables; output is to the Teletype.

Table 9-10 (Cont.). FORTRAN-D Statement Summary

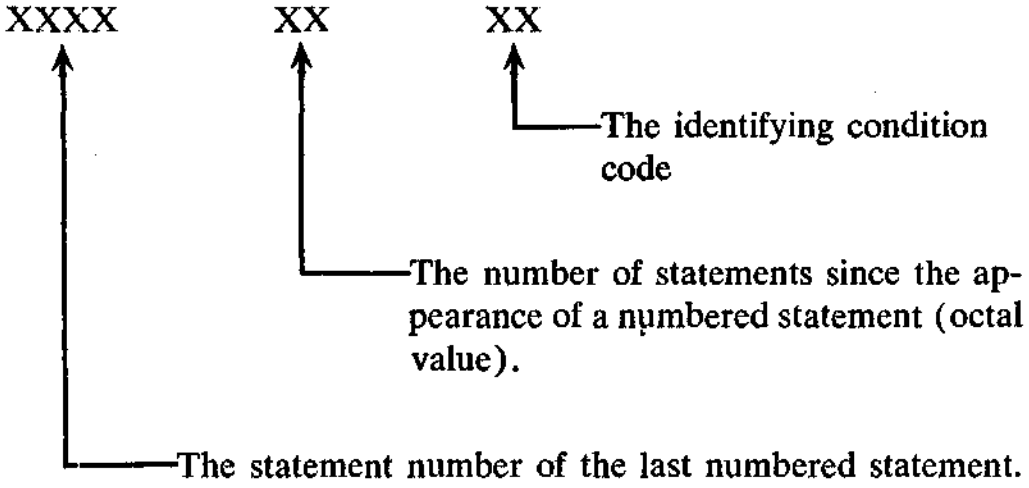
Statement and Form	Explanation
READ u,f, list	u is an integer representing the device which data is to be read: 1=Teletype, 2= high-speed reader, 3=disk; f is a FORMAT statement number; list is a list of variables.
WRITE u,f, list	u is an integer representing device onto which data will be written: 1=Teletype, 2=high-speed punch, 3=disk, f is a FORMAT statement number; list is a list of variables.

FORTRAN-D Error Diagnostics

Diagnostic procedures are provided in the Compiler to assist the programmer in program compilation. When the compiler detects errors in a FORTRAN source program, it prints the error messages on the user terminal. These messages indicate the source of the errors and direct the programmer's efforts to correct them. To speed up the Compiler process, the Compiler prints only an error code. The programmer then looks up the error message corresponding to the code in Tables 9-11 through 9-13 and takes the appropriate corrective measures.

COMPILER COMPILATION DIAGNOSTICS

Format of Diagnostics



For example:

```
10        A=I(J+1)
          B=A*(B+SINF(THTA)
```

During compilation of the previous statements, the following error code would be printed:

```
10 01 11
```

indicating that a statement which occurs one statement octal (one decimal) after the appearance of Statement 10 is in error. The message corresponding to Code 11 shows that the number of left and right parentheses in the statement is not equal.

If a statement number is referenced but does not appear in the source program, the diagnostic code will be printed as follows:

xxxx 77 20

where the number usually reserved for the last numbered statement (xxxx) is replaced by the missing statement, e.g.

```
GO TO 100
```

The diagnostic would appear as follows where Statement 100 is never defined.

```
100 77 20
```

Table 9-11. FORTRAN-D Compiler Compilation Diagnostics

Error Code	Explanation
00	Mixed mode arithmetic expression
01	Missing variable or constant in arithmetic expression
03	Comma was found in arithmetic expression
04	Too many operators in this expression
05	Function argument is in fixed-point mode
06	Floating-point variable used as a subscript
07	Too many variable names in this program
10	Program too large, core storage exceeded
11	Unbalanced right and left parentheses

Table 9-11 (Cont.). FORTRAN-D Compiler Compilation Diagnostics

Error Code	Explanation
12	Illegal character found in this statement
13	Compiler could not identify this statement
14	More than one statement with same statement number
15	Subscripted variable did not appear in a DIMENSION statement
16	Statement too long to process
17	Floating-point operand should have been fixed-point
20	Undefined statement number
21	Too many numbered statements in this program
22	Too many parentheses in this statement
23	Too many statements have been referenced before they appear in the program
25	DEFINE statement was preceded by some executable statement
26	Statement does not begin with a space, tab, C, or number

COMPILER SYSTEMS DIAGNOSTICS

Certain errors can make it impossible for the Compiler to proceed in the normal manner. These errors occur before the Compiler has been loaded into core. They may be caused by improperly loading the Compiler, by not having an END statement on a source file, by a machine malfunction, or by other errors. These errors, referred to as system errors, are explained in Table 9-12.

Table 9-12. FORTRAN-D Compiler Systems Diagnostics

Error Code	Explanation
0240	System file error. One of the FORTRAN components cannot be found or the disk is full, preventing FORTRAN from proceeding. Try recalling FORT.
3100	Illegal operator on compiler stack.
3417	Preprecedence error.
6145	Could not find FOSL on system device; if the error occurs, it may be necessary to reload FORT and FOSL.
6223	Error while loading the Compiler.
6226	Same as above.
6257	Same as above.
6724	No END statement on source device.
6746	Same as above.
7114	Same as above.

OPERATING SYSTEM DIAGNOSTICS

Not all errors are detected by the Compiler. Some errors can only be detected by the operating system (FOSL). Also there are some conditions which indicate errors on the part of the Compiler and/or operating system. When such an error occurs during running of a program, the computer prints out an error message containing the word TILT or ERROR and an error number. The computer then halts. If the CONTINUE switch is pressed, the computer takes the action listed in Table 9-13.

Table 9-13. FORTRAN-D Operating System Diagnostics

Error Code	Explanation
01	Checksum error on FORTRAN binary input
02	Illegal origin or data address on FORTRAN binary input
04	Disk input-output error[12]
05	High-speed reader error
06	Illegal FORTRAN binary input device
11	Attempt to divide by zero
12	Floating-point input data conversion error
13	Illegal op code
14	Disk input-output error[12]
15	Non-FORMAT statement used as a FORMAT
16	Illegal FORMAT specification
17	Floating-point number larger than 2047
20	Square root of a negative number
21	Exponential negative number
22	Logarithm of a number less than or equal to zero
40	Illegal device code used in READ or WRITE statement
41	System device full, could not complete a WRITE statement
76	Stack underflow error[13]
77	Stack overflow error[13]

[12] May be caused by machine malfunction or operating system error.

[13] May be caused by source program or loading error; to correct, do the following in descending order.

a. Use Diagnose to determine where the error occurred.
b. Recompile the source program.
c. Examine source program (in particular the arithmetic statements and subscripted variables).

PAL-D ASSEMBLER

Introduction

The EduSystem 50 Assembly System is composed of the PAL-D Symbolic Assembler, LOADER, and ODT. The PAL-D Assembler is used to translate the user's source program into an object program (binary or machine code). LOADER is used to transfer the user's object program from the disk into core for debugging or execution. ODT (Octal Debugging Technique) is used to dynamically debug the object program which has been loaded into core using LOADER.

The PAL-D Assembler is fully documented in *Programming Languages 1972*, Chapter 14. PAL-D source programs are usually written on-line using the EDIT program, which stores them in disk files. The Assembler checks for assembly language syntax errors and for undefined user symbols. To call the PAL-D Assembler, the user types:

```
.R PALD
```

PAL-D responds by requesting INPUT: Type the name of the source program or programs to be assembled. A maximum of three files can be assembled together. PAL-D then requests OUTPUT: Type the name of the new file in which PAL-D will store the assembled program in executable binary form. PAL-D then requests OPTION: For a normal assembly, press the RETURN key. If an assembly listing is not desired, respond to OPTION with N.

PAL-D then proceeds to assemble the program: any errors in the program are indicated; the program symbol table is printed; and finally, an assembly listing of the source program is printed. When the listing is completed and the assembly finished, control is returned to the Monitor.

EduSystem 50 PAL-D

Because of the necessary hardware changes made for timesharing on EduSystem 50, PAL-D has been revised in the following ways (as differing from PAL-D on a non-timeshared PDP-8):

A. PAL-D, under EduSystem 50, allows 245 (decimal) user symbols in addition to the permanent symbols listed in

Table 9-14. All symbols reside in locations 5200 through 7777. The permanent symbol table has been revised to include all instructions peculiar to the time-sharing system.

B. A CTRL/C (↑C) from the Teletype terminates the assembly, and halts PAL-D, sending the user back to the Monitor.

Example of a PAL-D Program

```
.R PALD

 INPUT:TYPE2
OUTPUT:BIN2
OPTION:

COUNT   0415
CRLF    0417
LOOP    0406
OUT     0425
REG     0416
START   0400

                /PROGRAM TO TYPE OUT "0123456789"
                *0400
0400    7200    START,  CLA
0401    4217            JMS CRLF
0402    1377            TAD (-12)
0403    3215            DCA COUNT
0404    1376            TAD (260)       /ASCII CODE FOR ZERO (0)
0405    3216            DCA REG
0406    1216    LOOP,   TAD REG
0407    4225            JMS OUT
0410    2216            ISZ REG
0411    2215            ISZ COUNT
0412    5206            JMP LOOP
0413    4217            JMS CRLF
0414    7402            HLT
0415    0000    COUNT,  0
0416    0000    REG,    0

0417    0000    CRLF,   0
0420    1375            TAD (215)       /ASCII FOR CARRIAGE RETURN
0421    4225            JMS OUT
0422    1374            TAD (212)       /LINE FEED
0423    4225            JMS OUT
0424    5617            JMP I CRLF

0425    0000    OUT,    0
0426    6046            TLS
0427    6041            TSF
0430    5227            JMP .-1
0431    7200            CLA
0432    5625            JMP I OUT
0574    0212
0575    0215
0576    0260
0577    7766
^BS
.
```

Table 9-14. EduSystem 50 Symbol List

Mnemonic	Code	Operation	Event Time
Memory Reference Instructions			
AND	0000	Logical AND	
TAD	1000	Twos complement add	
ISZ	2000	Increment & skip if zero	
DCA	3000	Deposit & Clear AC	
JMS	4000	Jump to subroutine	
JMP	5000	Jump	
Group 1 Operate Microinstructions			
NOP	7000	No operation	1
IAC	7001	Increment AC	3
RAL	7004	Rotate AC & link left one	3
RTL	7006	Rotate AC & link left two	3
RAR	7010	Rotate AC & link right one	3
RTR	7012	Rotate AC & link right two	3
CML	7020	Complement link	2
CMA	7040	Complement AC	2
CLL	7100	Clear link	1
CLA	7200	Clear AC	1
Group 2 Operate Microinstructions			
HLT	7402	Halts the computer	4
OST	7404	Inclusive OR switch register with AC	3
SKP	7410	Skip unconditionally	1
SNL	7420	Skip on nonzero link	1
SZL	7430	Skip on zero link	1
SZA	7440	Skip on zero AC	1
SNA	7450	Skip on nonzero AC	1
SMA	7500	Skip on minus AC	1
SPA	7510	Skip on plus AC (zero is positive)	1
Combined Operate Microinstructions			
CIA	7041	Complement & increment AC	1
STL	7120	Set link to 1	1
GLK	7204	Get link (put link in AC, bit 11)	1
STA	7240	Set AC = −1	1
LAS	7604	Load AC with switch register	1

PSEUDO-OPERATORS

DECIMAL	OCTAL
EXPUNGE	PAGE
FIELD	PAUSE
FIXTAB	TEXT
I	XLIST
	Z

Table 9-14 (Cont.). EduSystem 50 Symbol List

Mnemonic	Code	Operation	Event Time
IOT Microinstructions			
PROGRAM INTERRUPT			
IOT	6000	(See *Introduction to Programming 1972*, Chapter 6.)	
KEYBOARD READER			
KSF	6031	Skip if keyboard/reader flag = 1	1
KCC	6032	Clear AC & keyboard/reader flag	2
KRS	6034	Read keyboard/reader buffer	3
KRB	6036	Clear AC & read keyboard buffer, & clear keyboard flag	2,3
KSB	6400	Set keyboard break	
SBC	6401	Set buffer control flags	
KSR	6030	Read keyboard string	
TELEPRINTER/PUNCH			
TSF	6041	Skip if teleprinter/punch flag = 1	1
TCF	6042	Clear teleprinter/punch flag	2
TPC	6044	Load teleprinter/punch buffer, Select & print	3
TLS	6046	Load teleprinter/punch buffer, Select & print, and clear Teleprinter/punch flag	2,3
SAS	6040	Send a string	
HIGH-SPEED READER (TYPE PC02)			
RSF	6011	Skip if reader flag = 1	1
RRB	6012	Read reader buffer & clear flag	2
RCF	6014	Clear flag & buffer & fetch character	3
RRS	6010	Read reader string	
HIGH-SPEED PUNCH (TYPE PC03)			
PSF	6021	Skip if punch flag = 1	1
PCF	6022	Clear flag & buffer	2
PPC	6024	Load buffer & punch character	3
PLS	6026	Clear flag & buffer, load & punch	2,3
PST	6020	Punch string	
DECTAPE TRANSPORT (TYPE TU55) AND CONTROL (TC01)			
DTXA	6764	Load status register A	3
DTSF	6771	Skip on flags	1
DTRB	6772	Read status register B	2

Table 9-14 (Cont.). EduSystem 50 Symbol List

Mnemonic	Code	Operation	Event Time
PROGRAM CONTROL			
URT	6411	User run time	
TOD	6412	Time of day	
RCR	6413	Return clock rate	
DATE	6414	Date	
STM	6415	Quantum synchronization	
TSS	6420	Skip on TSS/8	
USE	6421	User	
SSW	6430	Set switch register	
CKS	6200	Check status	
ASD	6440	Assign device	
REL	6442	Release device	
DUP	6402	Duplex	
CON	6422	Console	
FILE CONTROL			
WHO	6616	Who	
SIZE	6614	Segment size	
RFILE	6603	Read file	
WFILE	6605	Write file	
ACT	6617	Account number	
REN	6600	Rename file	
OPEN	6601	Open file	
CLOS	6602	Close file	
PROT	6604	Protect file	
CRF	6610	Create file	
EXT	6611	Extend file	
RED	6612	Reduce file	
FINF	6613	File information	

Table 9-15. PAL-D Error Diagnostics

Error Code	Explanation
BE	Two PAL-D internal tables have overlapped—This situation can usually be corrected by decreasing the level of literal nesting or number of current page literals used prior to this point on the page.
DE	System device error—An error was detected when trying to read or write onto the system device; after three failures, control is returned to the Monitor.
DF	Systems device full—The capacity of the systems device has been exceeded; assembly is terminated and control is returned to the Monitor.
IC	Illegal character—An illegal character was encountered other than in a comment or TEXT field; the character is ignored and the assembly continued.
ID	Illegal redefinition of a symbol—An attempt was made to give a previously defined symbol a new value by means other than the equal sign; the symbol was not redefined.
IE	Illegal equals—An equal sign was used in the wrong context. Examples: TAD A+=B A+B=C the expression to the left of the equal sign is not a single symbol or, the expression to the right of the equal sign was not previously defined
II	Illegal indirect—An off-page reference was made; a link could not be generated because the indirect bit was already set.

Table 9-15 (Cont.). PAL-D Error Diagnostics

Error Code	Explanation
ND	The program terminator, $, is missing.
PE	Current nonzero page exceeded—An attempt was made to: a. override a literal with an instruction, or b. override an instruction with a literal; this can be corrected by (1) decreasing the number of literals on the page or (2) decreasing the number of instructions on the page.
PH	Phase error—PAL-D has received input files in an incorrect order; assembly is terminated and control is returned to the Monitor.
SE	Symbol table exceeded—Assembly is terminated and control is returned to the Monitor; the symbol table may be expanded to contain up to 1184 user symbols by saving a file named .SYM on the system device.
US	Undefined symbol—A symbol has been processed during pass 2 that was not defined before the end of pass 1.
ZE	Page 0 exceeeded—Same as PE except with reference to page 0.

UTILITY PROGRAMS

Symbolic Editor

The EduSystem 50 Symbolic Editor (EDIT) provides the user with a powerful tool for creating and modifying source files on-line. Its precise capabilities and commands are detailed in *Introduction to Programming 1972,* Chapter 5. EDIT allows the user to delete, insert, change, and append lines of text, and then obtain a clean listing of the updated file. EDIT also contains commands for searching the file for a given character.

EDIT considers a file to be divided into logical units, called pages. A page of text is generally 50-60 lines long, and hence corresponds to a physical page of program listing. A FORTRAN-D program is generally 1-3 pages in length; a program prepared for PAL-D may be several pages in length. EDIT operates on one page of text at a time, allowing the user to relate his editing to the physical pages of his listing. EDIT reads a page of text from the input file into its internal buffer where the page becomes available for editing. When a page has been completely updated, it is written onto the output file and the next page of the input file is made available. EDIT provides several powerful commands for paging through the source file quickly and conveniently.

NOTE

The end of a page of text is marked by a form feed (CTRL/L) character. Form feed is ignored by all EduSystem 50 language processors.

To call the Editor, type:

```
.R EDIT
```

EDIT responds by requesting INPUT: Type and enter the name of the source file to be edited. If a new file is to be created using EDIT, there is no input file. In this case, strike the RETURN key. EDIT then requests OUTPUT: Type the name of the new, edited file to be created. The name of the output file must be different from the name of the input file. If EDIT is being called to list the input file, there is no need to create an output file; strike

the RETURN key. When EDIT sets up its internal files and is ready for a command, it rings the bell on the Teletype.

For example:

```
.R EDIT
INPUT:WXZOLD.                    (Bell rings at this point.)
OUTPUT:XYZNEW
```

Table 9-16. Symbolic Editor Operations Summary

Special Characters	Function
Carriage Return (RETURN Key)	Text Mode—Enter the line in the text buffer.
	Command Mode—Execute the command.
Back Arrow (←)	Text Mode—Delete from right to of text, continue typing on same line.
	Command Mode—Cancel command. Editor issues a ? and carriage return/line feed.
Rubout (\)	Text Mode—Delete from right to left one character for each rubout typed. Does not delete past the beginning of the line. Is not in effect during a READ command.
	Command—Mode—Same as back arrow.
Form Feed (CTRL/FORM Combination)	Text Mode—End of inputs return to command mode.
Period (.)	Command Mode—Current line counter used as argument alone or in combination with + or − and a number (.,.+5L).
Slash (/)	Command Code—Value equal to number of last line in buffer. Used as argument (/−5,L/).
Line Feed (↓)	Text Mode—Used in SEARCH command to insert a CR/LF combination into the line being searched.

Table 9-16 (Cont.). Symbolic Editor Operations Summary

Special Characters	Function
Right Angle Bracket (>)	Command Mode—List the next line (equivalent to .+1L).
Left Angle Bracket (<)	Command Mode—List the previous line (equivalent to .−1L).
Equal Sign (=)	Command Mode—Used in conjunction with . and / to obtain their value (.=27).
Tabulation (CTRL/TAB Key Combination)	Text Mode—Produces a tabulation which, on output, is interpreted as spaces if bit 1 of the switch register is set to 0, or as a tab character/rubout combination if bit 1 is set to 1.

Table 9-17. EDIT Command Summary

Command	Format	Meaning
READ	R	Read text from the input file and append to buffer until a form feed is encountered.
APPEND	A	Append incoming text from keyboard to any already in the buffer until a form feed is encountered.
LIST	L	List the entire buffer.
	nL	List line n.
	m,nL	List lines through n inclusive.
PROCEED	P	Output the contents of the buffer to the output file, followed by a form feed.
	nP	Output line n, followed by a form feed.
	m,nP	Output lines m through n inclusive followed by a form feed.
TERMINATE	T	Close out the output file and return to the Monitor.

Table 9-17 (Cont.). EDIT Command Summary

Command	Format	Meaning
NEXT	N	Output the entire buffer and a form feed, kill the buffer and read the next page.
	nN	Repeat the above sequence n times.
KILL	K	Kill the buffer (i.e., delete all text lines).
DELETE	nD	Delete line n of the text.
	m,nD	Delete lines m through n inclusive.
INSERT	I	Insert before line 1 all the text from the keyboard until a form feed is entered.
	nI	Insert before line n until a form feed is entered.
CHANGE	nC	Delete line n, replace it with any number of lines from the keyboard until a form feed is entered.
	m,nC	Delete lines m through n, replace from keyboard as above until form feed is entered.
MOVE	m,n$kM	Move lines m through n inclusive to before line k.
GET	G	Get and list the next line beginning with a tag.
SEARCH	S	Search the entire buffer for the character specified (but not echoed) after the carriage return. Allow modification when found. Editor outputs a slash (/) before beginning a SEARCH.
	nS	Search line n, as above, allow modification.
	m,nS	Search lines m through n inclusive, allow modification.

Table 9-17 (Cont.). EDIT Command Summary

Command	Format	Meaning
END	E	Output the contents of the buffer. Read any pages remaining in the input file, outputting them to the output file. When everything in the input file has been moved to the output file, close it out and return to the Monitor. E is equivalent to a sufficient number of N's followed by a T command.
↑C	CTRL/C	Stop listing and return to Command Mode.

Loader

LOADER is used to load programs in BIN format from a disk file into the user's core area for execution. These files in BIN format can be created by PAL-D in the course of an assembly or they can be loaded from paper tape using PIP (see the PIP section for special instructions on loading BIN format tapes).

To call LOADER, type:

```
.R LOADER
```

LOADER responds by asking for INPUT: Respond by entering the name of the file or files to be loaded. Although many System Library Programs allow multiple input files, the LOADER uses this feature to special advantage. Because it loads the files in the order they are typed, LOADER can be used to load patches and overlays. After it has requested INPUT, LOADER requests OPTION: For normal operation strike the RETURN key; LOADER is able to load into any part of core below 7750. If the program to be loaded is to be debugged, respond to OPTION: with D. This will cause ODT to be loaded along with the input files and started. ODT indicates that it is ready by printing a second line feed. ODT uses locations 7000 through 7577; and if loaded along with a program which uses any of these locations, the result of the load is unpredictable.

EXAMPLE 1: NORMAL OPERATION

```
.R LOADER
INPUT:MAIN, PATCH1, PATCH2
OPTION:
↑BS
```

EXAMPLE 2: LOAD ODT WITH INPUT FILE

```
.R LOADER
INPUT:PROG1
OPTION: D
```

As seen in the first example, LOADER returns control to Monitor when it is finished. The user can then start the program by using the Monitor command START. For example, LOADER can be used to load and run the short program given as an example in the section on PAL-D.

```
.R LOADER

INPUT: BIN2
OPTION:
↑BS
.START 400


0123456789
↑BS
```

NOTE

All BIN format files loaded by LOADER include a checksum. If LOADER detects a checksum error while loading, it prints LOAD ERROR and terminates the load.

Octal Debugging Technique (ODT)

ODT is a powerful octal debugging tool for testing and modifying PDP-8 programs in actual machine language. It allows the user to control the execution of his program and, where necessary, make immediate corrections to the program without the need to reassemble.

The complete command repertoire of ODT is documented in *Introduction to Programming 1972,* Chapter 5. ODT (on EduSystem 50) is the high-core version which resides in locations 7000 through 7577. The paper-tape output commands of regular ODT are not available in EduSystem 50 ODT. To call ODT, the user types:

```
.LOAD 2 ODTHI Ø 7000
.START 7000
```

If ODT is to be used to debug a program being loaded with LOADER, ODT can be loaded and started directly by specifying the Debug (D) option to LOADER.

ODT executes an SRA (Set Restart Address) as part of its initialization process. As a result, typing CTRL/C always returns control to ODT. If the program being debugged sets up its own restart address, typing CTRL/C transfers control to the new restart address. It is necessary to type ↑BS followed by START 7000 to force control back to ODT. Every time ODT regains control, it puts the Teletype in duplex mode. Users debugging programs which do not operate in duplex mode, should be aware of this fact.

ODT saves the state of the delimiter mask, when it regains control via a breakpoint. The state of this mask is restored on a Continue (C) command, but not on a GO (G) command.

Table 9-18. ODT Command Summary

Command	Meaning
nnnn/	Open register designated by the octal number nnnn.
/	Reopen latest opened register.
RETURN	Close previously opened register.
LINE FEED	Close register and open the next sequential one for modification.
Up Arrow (↑) (SHIFT/N)	Close register, take contents of that register as a memory reference and open it.

Table 9-18 (Cont.). ODT Command Summary

Command	Meaning
Back Arrow (←) (SHIFT/O)	Close register open indirectly.
Illegal Character	Current line typed by user is ignored, ODT types ?CR/LF.
nnnnG	Transfer program control to location nnnn.
nnnnB	Establish a breakpoint at location nnnn.
B	Remove the breakpoint.
A	Open for modification, the register in which the contents of AC were stored when the breakpoint was encountered.
C	Proceed from a breakpoint.
nnnnC	Continue from a breakpoint and iterate past the breakpoint nnnn times before interrupting the user's program at the breakpoint location.
M	Open the search mask register, initially set to 7777. It may be changed by opening the search mask register and typing the desired value after the value typed by ODT, then closing the register.
LINE FEED	Close search mask register and open next register immediately following, containing the location at which the search begins. It may be changed by typing the lower limit after the one typed by ODT, then closing the register.
LINE FEED	Close lower search register, open next register containing the upper search limit initially set to 7000 or 1000 (location of ODT). It may be changed by typing the desired upper limit after the one typed by ODT and closing the register with a carriage return.
nnnnW	Search the portion of core as defined by the upper and lower limits for the octal value nnnn.

Catalog (CAT)

The Monitor maintains a library of disk files for each user. The System Library Program CAT is used to obtain a catalog of the contents of this library. For each file, CAT prints the size of the file in units of disk segments. The size of a disk segment may vary among installations. Generally, it is 256 (decimal) words of disk storage. The protection code for the file is also given. (See the section on Advanced Monitor Commands for a precise explanation of protection codes.) If the program was created by any of the System Library Programs, it has a protection code of 12, meaning that other users can read the file, but only the owner can change it. To call CAT, type:

```
.R CAT
```

The CAT program then prints a listing similar to the one shown below and concludes by printing ↑BS and exiting to the Monitor.

```
.R CAT

DISK FILES FOR USER  3,13 ON  9-JUN-70

NAME       SIZE  PROT    DATE
FIE   .BIN    1    17   3-JUN-70
PROG  .FCL    2    12   9-JUN-70
INTER .BAS    1    17   9-JUN-70
BAS000.TMP    1    17   9-JUN-70
BAS100.TMP    1    17   9-JUN-70
INT2  .BAC    1    37   9-JUN-70
FCLPRG.FCL    2    12   9-JUN-70

TOTAL DISK SEGMENTS:  9

↑BS
```

System Status (SYSTAT)

It is frequently useful to know the status of the system as a whole; how many users are on-line, where they are, what they are doing, etc. The SYSTAT program provides this capability. To call SYSTAT, type:

```
.SYSTAT
```

or

```
.R SYSTAT
```

SYSTAT responds by printing on the first line: the version of the Monitor being run, the time, and the date. SYSTAT then reports the uptime which is the length of time in hours, minutes, and seconds since the system was last put on-line.

SYSTAT then lists all on-line users. Each user is identified by his account number. The job number assigned to him and the number of the console he is using are indicated, as is the particular System Program he is running. The exact running state of each user, whether he is actually running (RUN), typing (KEY) or printing (TTY), doing input/output on another system device (IO or FIP), or not running (↑B), is indicated. The amount of computer time used by each user since he logged in is given.

If more users are on-line than the system has core fields to hold them, the fact that the system is swapping is reported. The number of free core blocks used internally by the Monitor for Teletype buffering and various other purposes is printed. Then SYSTAT reports any unavailable devices, i.e., devices which are assigned to individual users. The job to which they are attached and their status (AS if they are assigned but not active, AS+INIT if they are assigned and active) is also indicated. Finally, the number of available segments of disk storage is reported.

A sample SYSTAT listing is shown below. SYSTAT terminates by printing ↑BS and exiting to the Monitor.

```
.R SYSTAT

STATUS OF TSS/8.21 DEC PDP-8 #1 AT 16:32:52 ON  2 JUN 70

UPTIME 08:55:52

JOB       WHO      WHERE      WHAT       STATE      RUNTIME

 1       10,10      K00       COPY        ↑B        00:00:06
 2        3,13      K15       SYSTAT      RUN       00:00:05
 3        0,10      K11       BASIC       KEY       00:00:02
 4        0,10      K22       BASIC       KEY       00:00:01

AVAILABLE CORE  0K   FREE CORE=222

BUSY DEVICES

DEVICE    JOB       WHY

D0        1         AS+INT
D1        1         AS+INT
D2        1         AS
D3        1         AS
D6        3         AS+INT

119 FREE DISK SEGMENTS


↑BS
.
```

PROGRAMS FOR PAPER TAPE AND DECTAPE CONTROL

PIP (Peripheral Interchange Program)

All System Library programs discussed in previous sections operate only on files which are on the disk. Disk is a convenient storage medium for many files; however, it may be more useful to keep some programs on paper tape. PIP provides a convenient means of transferring files between disk and paper tape, for those users who wish to preserve copies of their files off-line.

PIP CONVENTIONS

PIP may be considered a link between disk file storage and paper-tape devices. To punch a desired file, PIP obtains that file from the disk and punches it on paper tape. Similarly, to load a paper tape, PIP inputs the tape from the reader, then outputs it to a disk file.

The way files are named is important to PIP. Files on disk are always named. Paper tapes, on the other hand, have no names as far as the system is concerned (although the user can label the physical tape in any manner he chooses). Paper tapes never have file names; therefore, PIP uses the absence of a file name to indicate a paper tape (absence of a file name is indicated by striking the RETURN key).

The way in which INPUT: and OUTPUT: is indicated provides the means for determining the direction of file transfer. If PIP is to get its input from the disk, the input is a file name; if the input is from a paper tape, no file name is given. Similarly, if PIP is to output to the disk, the file name is indicated; if output is to paper tape, no name is given. To call PIP, type:

```
.R .PIP
```

PAPER TAPE TO DISK TRANSFERS

To move a paper tape to disk, strike the RETURN key when PIP requests INPUT: Since PIP must output to the disk, respond to OUTPUT: by typing a file name. When PIP requests OPTION:

type T to indicate that the paper tape is being loaded from the Teletype reader. For example:

```
.R PIP

INPUT:
OUTPUT: FILE1
OPTION:T
```

The paper tape in the low-speed reader is read and stored in the system as FILE1.

DISK TO PAPER TAPE TRANSFERS

To move a disk file onto paper tape, the use of file names is reversed since PIP must input a disk file and output it to paper tape. The option remains the same. For example:

```
.R PIP

INPUT:FILE1
OUTPUT:
OPTION:T
```

The contents of FILE1 are then punched at the Teletype.

HIGH-SPEED READER/PUNCH ASSIGNMENTS

PIP can also be used with high-speed paper-tape devices. The format of the INPUT: and OUTPUT: responses is the same. However, for the high-speed reader, the option is R and for the punch it is P.

Since the reader and punch are assignable devices, they are not always available (other users may have one or both assigned). Therefore, whenever PIP is given a command which utilizes one of these devices, it checks to make sure that the device is available. If it is, PIP automatically assigns it (thus, it is not necessary to assign the device before running PIP). If the device is unavailable, PIP informs the user. For example:

```
INPUT:
OUTPUT: ABCD
OPTION:R
```

PIP reads the paper tape in the high-speed reader and stores it in the system as ABCD.

```
INPUT:ABCD
OUTPUT:
OPTION:P
```

PIP punches out file ABCD on the high-speed punch.

```
INPUT:ABCD
OUTPUT:
OPTION:P
DEVICE NOT AVAILABLE
```

The punch is assigned to another user, or there is no punch on the EduSystem 50, or there is one but it is turned off.

BIN FORMAT FILE TRANSFERS

The examples above work for all ASCII file transfers (except BASIC programs, explained below.) They are also valid for punching BIN files with either high- or low-speed devices. Loading BIN format tapes, however, is a special case.

BIN format tapes must end with trailer codes. The easiest way to ensure that they do is to cut off the tape near the end of the trailer code. Failure to do this (or cutting it off very unevenly) does not prevent PIP from loading tape into the disk file. However, later attempts to load the file with LOADER will result in load errors.

NOTE

Some installations do not allow any BIN format tapes to be loaded from the low-speed reader.

MOVING DISK FILES

PIP can be used to move the contents of one file into another. This is often useful in copying a file from another user's library (providing the file is not protected) into your own library. To copy from disk file to disk file, specify a file name for both input and output. Reply to OPTION: by striking the RETURN key. For example:

```
INPUT:FOCAL 2
OUTPUT:FOCALX
OPTION:
```

PIP gets FOCAL from account number 2's library and moves it into the file FOCALX.

DELETING DISK FILES

One of the principal reasons for punching files on paper tape is to free disk space. Once punched, the disk file is no longer needed. PIP offers a convenient means of deleting files, the Delete option:

```
INPUT:ABCD
OUTPUT:
OPTION:D
```

PIP deletes file ABCD, provided that the file is not protected against being changed.

BASIC FILE TRANSFERS

BASIC stores its programs in a unique file format. Therefore, it is not possible to load or punch BASIC files in the usual way. To provide a convenient means of handling BASIC programs, the B option is available in PIP. The B option is used for both reading and punching BASIC programs. The responses to INPUT: and OUTPUT: indicate the direction of the transfer; the high-speed reader or punch is always assumed for the B option. (To read or punch tapes at low-speed, use BASIC itself.)

PIP assumes that any BASIC tapes it loads are clean and error-free. Only tapes actually created by BASIC should be loaded with PIP. Tapes created off-line, and thus liable to contain errors, should be loaded low-speed by BASIC itself with the TAPE command.

SAVE FORMAT FILE TRANSFERS

Another special file format is that of the SAVE files, those programs directly executed by EduSystem 50. (The System Library Programs are examples of SAVE format files.) PIP provides the S option, to allow these files to be punched on paper tape. SAVE format tapes make sense only to PIP. They cannot be input to any other System Program.

The responses to INPUT: and OUTPUT: indicate the direction of the transfer; the high-speed reader or punch is always assumed for the S option.

NOTE

SAVE format tapes include a checksum. If PIP detects an incorrect read, it prints LOAD ERROR, and terminates the load, repeating the request for input.

Table 9-19. PIP Option Summary

Option	Explanation
T	Transfer a file between the disk and the Teletype reader or punch. The response to INPUT: and OUTPUT: indicates the direction of the transfer.
R	Read a tape from the high-speed reader and store it as a disk file.
P	Punch the contents of a disk file on the high-speed punch.
D	Delete the file specified for input.
B	Transfer a BASIC program file between the disk and the high-speed reader or punch. The response to input and output indicates the direction of the transfer.
S	Transfer a SAVE format file between the disk and the high-speed reader or punch. The response to INPUT: and OUTPUT: indicates the direction of the transfer.

COPY Program

Many EduSystem 50 installations include one or more DECtapes. For these installations, DECtape provides a convenient and inexpensive means of file storage. The COPY program is used to transfer files between disk and DECtape.

USING AND CALLING COPY

COPY is the intermediary between disk and DECtape. To write a disk file out to DECtape, COPY inputs the file from the disk, then outputs it to the DECtape. To bring a DECtape file onto the disk, COPY inputs from the DECtape, then outputs to the disk.

Files kept on DECtape have file names just as they do on the disk. To avoid confusion, the user must tell COPY where the file is to be found. If it is on DECtape, the DECtape designation and the number of the DECtape unit must preface the file name. The DECtape number is always separated from the file name by a colon. Thus D1:FILE1 means the file name FILE1 on the DECtape which is currently mounted on DECtape unit number one. The number of available tape units varies among installations. The maximum is eight (numbered 0-7). If a file name is not prefaced by a DECtape number, the file is assumed to be on the system disk.

Files stored on DECtape do not have protection codes in the sense that disk files do. They are, however, protected against unauthorized access. When a DECtape is not mounted, it is not available to any user. When it is mounted, it is available only to the user who has assigned the DECtape unit on which it is mounted. Even then it can not be altered unless the DECtape unit is set to WRITE ENABLE. Users should be sure to assign a DECtape unit before mounting their tape, and dismount the tape before releasing the device. Normally, the DECtape unit to be used should be assigned before calling COPY.

To call COPY, type:

```
.R COPY
```

COPY responds by asking which option the user wishes to employ. The COPY options are discussed below and summarized in Table 9-20.

LOADING FILES FROM DECTAPE

To load a file onto the disk from DECtape, use the COPY option. When COPY requests OPTION, respond with COPY, or C, or strike the RETURN key (the COPY option is assumed). When COPY requests INPUT, type the number of the DECtape unit on which the file can be found (D0, D1, D2, D3, D4, D5, D6, or D7) followed by a colon and the name of the file on the DECtape. When COPY requests OUTPUT, type and enter the name to be given to the output file on the disk. COPY then moves the DECtape file onto the disk. (When using COPY, it is

not mandatory to insert a space between the device designator and the device number.) For example:

```
OPTION- COPY
INPUT - D4:PQR
OUTPUT - PQR
```

If for any reason, COPY cannot find the DECtape file specified for input (the specified DECtape is unavailable or nonexistent, or the file name does not exist on that DECtape), COPY prints a ? and repeats the request for input. If the disk file specified for output already exists, COPY prints a ? and repeats the request for output. COPY does not overwrite an existing file. For example:

```
OPTION - C
INPUT - D9:PQR
?INPUT - D4:PQRS
?INPUT - D4:PQR
OUTPUT - FILE1
?OUTPUT - PQR
```

SAVING DISK FILES ON DECTAPE

Saving a disk file on DECtape is very similar to loading one. The option is still COPY. For input, respond with the name of the file on the disk. For output, type the DECtape unit number, colon, and the name to be given to this file. For example:

```
OPTION - C
INPUT - ABCD
OUTPUT - D4:ABCD
```

If COPY cannot find the file on the disk, or if it is protected, COPY prints a ? and repeats the request for input. If COPY cannot create the desired DECtape file (the specified DECtape does not exist or is unavailable, or it is not WRITE ENABLED, or a file by that name already exists on the tape) COPY prints a ? and repeats the request for output.

LISTING DIRECTORIES

COPY can be used to list the directory of a device. To list a directory, respond to OPTION by typing LIST, or just L. COPY then asks which device directory it is to list. To list a DECtape

directory, respond with the device name (D0,...,D7). Do not follow it by a colon. For example:

```
.R COPY

OPTION- LIST
INPUT- D 1

1372. FREE BLOCKS

NAME        SIZE     DATE
BASIC .SAV   66    9-MAR-70
FACTAL.BAS   10    2-MAR-70
CONVER.BAC    6    1-MAR-70
PALD  .SAV   32   31-MAR-70
```

The unit of DECtape storage is the block, which is 128 (decimal) words. Because the unit of disk storage, the segment, is generally 256 words, a file occupies twice as many blocks of DECtape storage as it did segments on the disk.

COPY can also be used to list the user's disk directory. Use the LIST option, but respond to DEVICE by simply striking the RETURN key. The directory listing is similar to the listing obtained by running the CAT program.

DELETING FILES

COPY can be used to delete files, either on the disk or on a selected DECtape. To delete a file, respond to OPTION by typing DELETE, or just D. Respond to INPUT by typing the name of the file to be deleted. If the file is on a DECtape, preface the file name with the DECtape unit number and a colon. For example:

```
OPTION - DELETE
INPUT - D4:ABCD
```

If COPY cannot find the file to be deleted, or having found it, cannot delete it (it is a protected disk file or a DECtape file on a unit which is not WRITE ENABLED), COPY prints a ? and repeats the request for INPUT.

DELETING ALL EXISTING FILES ON A DEVICE

COPY can be used to delete all existing files on a device. To do so, respond to OPTION by typing ZERO, or just Z. When COPY requests INPUT respond with the name of the device. To delete all files on the disk, strike the RETURN key. The ZERO option

should also be used to format a blank DECtape before attempting to copy any files onto it. For example:

```
OPTION - ZERO
INPUT - D4

OPTION - Z
INPUT -
```

COPY cannot delete files from a DECtape unless it is WRITE ENABLED. It cannot delete disk files which are write protected.

Table 9-20. COPY Option Summary

Option	Abbre-viation	Explanation
LIST	C	Transfer a file between disk and DECtape.
COPY	L	List a directory.
DELETE	D	Delete a file.
ZERO	Z	Delete all files.

EXAMPLE OF COPY USAGE

```
.ASSIGN D 5
D 5 ASSIGNED
.R COPY

OPTION - ZERO
DEVICE - D5

OPTION - LIST
DEVICE - D5

1462. FREE BLOCKS

NAME      SIZE      DATE

OPTION - LIST
DEVICE -

DISK FILES FOR USER 54,40 ON 27-MAY-70.

NAME       SIZE  PROT     DATE
SOLVE .BAS   1     12    27-MAY-70

TOTAL DISK SEGMENTS: 1
```

```
OPTION - COPY
INPUT - SOLVE
OUTPUT - D5: SOLVE

OPTION - DELETE
INPUT - SOLVE

OPTION - LIST
DEVICE - D5

1460. FREE BLOCKS

NAME     SIZE      DATE
SOLVE       2    27-MAY-70

OPTION- LIST
DEVICE-

DISK FILES FOR USER 54,40 ON 27-MAY-70.

NAME      SIZE   PROT      DATE

TOTAL DISK SEGMENTS: 0

OPTION- COPY
INPUT- D5:SOLVE
OUTPUT- ABCD

OPTION- LIST
DEVICE-

DISK FILES FOR USER 54,40 ON 27-MAY-70.

NAME     SIZE     PROT         DATE
ABCD        1      12        27-MAY-70

TOTAL DISK SEGMENTS: 1

OPTION- RENAME
INPUT- ABCD
OUTPUT- FIE.BIN <17>

OPTION- LIST
DEVICE-

DISK FILES FOR USER 54,40 ON 27-MAY-70.

NAME     SIZE     PROT       DATE
FIE .BIN  1        17      27-MAY-70

OPTION- ↑BS
.RELEASE D5
```

ADVANCED MONITOR COMMANDS

Introduction

The fundamental Monitor commands described previously are those needed to utilize existing System Library Programs. The EduSystem 50 Monitor also provides powerful commands for users who wish to create their own library programs.

To use the System Library Programs described previously, it was not necessary to be familiar with the actual machine that runs them, the PDP-8/E. To create new library programs for EduSystem 50, this is necessary because they are written in the PDP-8 assembly language. The user codes his programs for a 4K PDP-8, subject to the time-sharing conventions discussed in this section. The programs are created with EDIT, then assembled by PAL-D and loaded by LOADER. Only at this point are the programs able to be run by EduSystem 50. In the course of this program development, the same program exists in many formats.

The source program is a disk file containing ASCII characters in an Editor format. PAL-D reads the file and translates it into a second file, the assembled program in BIN format. Neither of these files is capable of being executed directly by EduSystem 50. The BIN format tape must be loaded into core by LOADER before it can actually be executed.

At this point it is possible to save the program in a file format that is directly executable by EduSystem 50. Such a file, called a SAVE format file, contains an image of the user's core area after the program has been loaded by LOADER. These SAVE format files differ from all the files which are created by System Library Programs and cannot be executed directly by EduSystem 50. Thus, it is not possible to save a BASIC program (e.g., FILE1 while running BASIC), then return to Monitor, type R FILE1, and get meaningful results. The program in FILE1 must be executed under control of the BASIC language processor. Only SAVE format files can be called into execution directly by the R command. (All System Library Programs are stored in SAVE format and can be run with the R command.)

NOTE

In the following examples, Sn, Cn, and Dn are used to stand for alphanumeric strings (such as file names), octal numbers, and decimal numbers, respectively.

A number of Monitor command conventions are available to make the commands easier to use. First, more than one command may be typed on a line. Individual commands are separated by a semi-colon (;). Second, only enough characters of a command to uniquely specify it need be typed. Thus, DEPOSIT can be abbreviated DE or DEP.

```
.LOAD FILE1; DEP  20  7000; ST 200
```

is exactly equivalent to:

```
.LOAD FILE1
.DEPOSIT 20 7000
.START 200
```

These conventions are available for the elementary Monitor commands as well. They are, however, especially convenient for the advanced commands.

Control of User Programs

Once a PAL-D program has been loaded by LOADER, several Monitor commands are available for controlling its execution. These commands are shown in Table 9-21.

It is possible to give these utility commands while a user program is running. The CTRL/B character (↑B) gets the attention of the Monitor without stopping program execution. (↑B followed by the S command stops the program.) ↑B can be used together with the WHERE command to follow program execution. After executing these commands, Monitor does not put the Teletype back into Monitor mode.

Table 9-21. Monitor Program Control Commands

Command	Explanation
START C1	Start execution of a user program at location C1. When a program is started, keyboard input is no longer interpreted as commands to Monitor. Input characters are passed to the running program. START C1 clears the user's AC and link.
START	Restart execution of a user program where it was interrupted (either by execution of an HLT or by ↑BS typed at the keyboard). When the START command is given, the program's state is restored.
DEPOSIT C1 C2...Cn	Deposit the octal values C2 to Cn in the locations starting at C1. DEPOSIT is used to make small octal modifications to a user program. No more than 10 decimal locations can be modified by a single DEPOSIT instruction.
EXAMINE C1	Print the octal contents of location C1.
EXAMINE C1 D1	Print the contents of D1 locations starting at C1.
WHERE	Print the present status of the user program. The user's AC, PC, and LINK are printed. If the processor includes the extended arithmetic element, two additional registers, the SC and MQ are printed.

Defining Disk Files

The Monitor allows the user to save core images of his program on the disk for future use. However, before saving such a core image, the user must define a disk file in which to save it.

Disk files, like the user's core, are made up of 12-bit words. Unlike the user's core, which is always 4K in size, a file can be any size. The unit of disk file storage is the segment; in most installations a segment is 256 (decimal) words but can be from

128 to 1024 words long. Files are at least one segment long when created and grow by appending additional segments to the end of the file. In defining a file, the user first creates it, then extends it to whatever length he needs. To have a whole 4K image on a system with a segment size of 256 (decimal) words, a 16 segment file is required. If only part of the contents of the user's core is to be saved, a correspondingly smaller file can be used.

A file can be created at any time. However, to modify or redefine it in any way, the file must be open. Up to four files can be open for a user simultaneously. Opening a file connects it to an internal open file number (0, 1, 2, or 3). Once a file is open, it is referenced by this internal file number rather than by its file name.

CREATING A DISK FILE

The CREATE command defines an area of disk space and associates it with the name given in the command line.

The file name can be one to six alphanumeric characters of which the first must be a letter. Creating a file deletes any existing file of the same name, unless that file is write protected. When created, files are always one segment in size. A new file is arbitrarily assigned a protection code of 12, meaning that other users may access it but only the owner may change it. Until it has been written, the contents of a newly defined file are undefined.

OPENING AND CLOSING A FILE

To use a file, it must first be opened with the OPEN command. A file can be opened on any of four internal file numbers: 0, 1, 2, or 3. A user can have up to four files open at a time. If a file is open on an internal file number for which a file is already open, that file is first closed. For example:

```
.CREATE AB
.OPEN 1 AB
```

AB is now an open file and can be referenced as file 1.

An open file can be closed at any time by means of the CLOSE command. Once closed, a file cannot be accessed in any way until it is reopened. It is possible to close more than one file with a single command. For example:

```
•CLOSE 0 1 2 3
```

EXTENDING, REDUCING, AND RENAMING A DISK FILE

When created, a file is one segment long. If a larger file is needed, the original file can be extended. For example, the command:

```
•EXTEND C1 D1
```

extends the file presently open on internal file C1 by D1 segments. Extending a file adds one or more segments to the end of that file. The contents of the old part of the file are not changed. Until written, the contents of the newly added segments are unspecified. An existing file may be reduced in size by means of the REDUCE command. For example, the command:

```
•REDUCE C1 D1
```

reduces the file presently open on internal file C1 by D1 segments. Reducing a file deletes the number of segments indicated from the end of the file. The contents of remaining segments of the file are unchanged. If a file is reduced to zero segments, or if D1 is greater than the number of segments in the file, it is deleted entirely. An example of the creation and deletion of a 4K file:

```
•CREATE FOURK
•OPEN 3 FOURK
•EXTEND 3 15
•REDUCE 3 16
```

Existing opened files can be renamed. Renaming a file does not change its contents in any way. For example, the command:

```
•RENAME C1 S1
```

renames as S1 the file open on internal file number C1.

PROTECTION CODES

The user can protect his files against unauthorized access. He can also specify the extent of access certain other users can have

to his files. For example, a user's associates can be permitted to look at the data of certain files but not permitted to alter that data.

When it is created, a file is assigned a protection code of 12. This protection code is defined below and can be changed (see Storage Allocation), but only by the owner of that file. For example, the command:

```
.PROTECT C1 C2
```

gives the protection code C2 to the file open on internal file number C1.

The protection code is actually a 5-bit mask. Each bit specifies a unique level of protection. (See the PROTECT IOT command for the meaning of each bit.)

File protection masks (C2) are assigned as follows:

1 Read protect against users whose project number differs from owner's.

2 Write protect against users whose project number differs from owner's.

4 Read protect against users whose project number is same as owner's.

10 Write protect against users whose project number is same as owner's.

20 Write protect against owner. To change the program the owner must change the protect code.

Protection codes are determined as the unique sum of any of the above codes. Some of the more common protection codes are as follows:

Command	Explanation
PROTECT 1 12	Allow other users to access the file but not change it.
PROTECT 1 17	Allow only the file owner to read the file. He can also change it.
PROTECT 1 37	Allow only the file owner to read the file. He cannot, however, change it. (To change it, he must first change the protection.)

Code	Explanation
.PROTECT 1 0	Allow other users to access the file and change it.

Finally, the user can ask what file is open on a given internal file number by means of the F (File information) command. For example, the command:

```
.F C1
```

prints the following information about the file presently open on an internal file C1:

a. Account number of file owner.
b. Name of file.
c. Protection code.
d. Size of file in segments (decimal).

For example:

```
.F 1
0010 TYPE 0012    2
.
```

ERROR CONDITIONS

There are a number of error conditions which prevent the execution of the file definition commands (as previously described). One of the following error messages is printed by Monitor if an error condition is detected:

Message	Explanation
FILE NOT OPEN	An EXTEND, REDUCE, PROTECT, or RENAME command has been issued for an internal file number for which no file is open.
PROTECTION VIOLATION	An attempt has been made to change a file which is write protected against the user.

Message	Explanation
FILE IN USE	An EXTEND, REDUCE, PROTECT, or RENAME command has been issued for a file which is in use elsewhere by another user. Because changing a file which is being used (i.e., has been opened) could disrupt another user's work, under these conditions such a change is prohibited.
DIRECTORY FULL	A CREATE command has been issued, but the user's directory is full. He can delete any of his files to make room for the new file.
FILE NOT FOUND	The user has attemped to OPEN a nonexistent file.
FAILED BY n SEGMENTS	The user has attempted to extend a file, but the system has run out of disk segments. The number of segments requested, but not available, is printed.

Saving and Restoring User Programs

Once a file has been defined, the user can save all or any part of his user core in the file. Files and user core are addressed in the same way, by 12-bit words. The user can transfer his file into any part of core.

The SAVE command requires one to five parameters. The name of the file to be written into must always be given. If the file is not in the user's own library, the appropriate account number is entered before the file name. (Writing into a file owned by another user is subject to file protection.) In either case, the

parameters are separated by spaces. The SAVE command writes the indicated section of core out into the indicated file.

If no parameters follow the file name, Monitor starts at location zero of the user's core and saves it in location zero of the disk file. It continues to write core locations into the disk file until: (a) it has written the whole 4K or (b) it has filled the file. Either condition completes the SAVE.

The user can further define his SAVE command by indicating parts of core to be saved in specific parts of the disk file. He does this by typing one to three parameters following the file name. The first parameter following the file name indicates a specific disk file address at which to begin writing. The second parameter following the file name indicates a specific core address at which to terminate the transfer. If only the first two parameters are typed, the transfer terminates when either the end of core or the end of file is reached.

Command	Explanation
SAVE S1 SAVE C1 S1	Assuming that a disk file S1 exists, and that it is not write protected, the contents of core are saved in S1. In the first case, S1 is assumed to be in the library of the user giving the command. In the second case, it is assumed to be in the library of the user whose account number is C1.
SAVE C1 C2 C3 C4	Locations C3 to C4 (inclusive) are saved in file S1 starting at disk file location C2. S1 is assumed to be in the user's own library. If S1 is preceded by the parameter C1, it is assumed to be in the library of the user whose account number is C1.

Once a core image has been saved in a disk file, it can be restored to core by means of the LOAD command. It should be noted that the Monitor command LOAD is very different from

the System Library Program LOADER. LOADER loads a BIN format file (created by PAL-D) into the user's core. LOAD loads a SAVE format file (created by a previous SAVE command) into core.

The LOAD command requires from one to five parameters. The name of the file to be loaded must always be given. If the file is in the user's own library, this file name is typed after the SAVE command itself. If it is in another user's library, his account number is entered before the file name. (Reading another user's file is subject to file protection.) In either case, the parameters are separated by spaces.

Command	Explanation
LOAD	Read the indicated section of a disk file into the indicated section of core.
LOAD S1 LOAD C1 S1	Assuming that a disk file S1 exists, and that it is not read protected, the contents of the file S1 are loaded into core. In the first case S1 is assumed to be in the library of the user giving the command. In the second case, it is assumed to be in the library of the user whose account number is D1.

The user can further define his LOAD command by using the same optional parameters discussed in the section on the SAVE command.

Command	Explanation
LOAD S1 C2 C3 C4	Locations C3 to C4 (inclusive) are loaded from file S1 starting at file location C2.
LOAD NEWF 5 10 17	Words 5 to 14 (inclusive) of the file named NEWF are loaded into locations 10 to 17 of the user's core.

It is not necessary to open a file before using it in a LOAD or SAVE command. Both commands automatically open the specified file on internal file number 3 before performing the transfer. After completion of the command, the file remains open on file number 3.

A special macro-command, RUN, exists to allow a program to be loaded and started all in one command.

Command	Explanation
RUN S1 RUN C1 S1	Load file S1 into core from the disk and start execution at location 0. In the first example, file S1 is assumed to be in the user's own library. In the second, it is assumed to be in the library of the user whose account number is C1. RUN S1 is exactly equivalent to LOAD S1; START 0. RUN C1 S1 is exactly equivalent to LOAD C1 S1; START 0.

The R command (see the section EduSystem 50 Monitor) is a special case of the RUN command. For example, the command:

```
.R S1
```

loads file S1 from the System Library (account number 2) and starts at location 0. R S1 is exactly equivalent to RUN 2 S1.

Utility Commands

The Monitor provides a number of special purpose commands to aid in program development and use. The Monitor utility commands are summarized in Table 9-22.

Table 9-22. Monitor Utility Commands

Command	Explanation
USER	Print the number of the job connected with this user and the console number of the job.
USER C1	Print the console numbers of job C1.
SWITCH C1	Set the user's switch register to C1. Monitor maintains a switch register for each user. When his program executes on OSR (OR the switch register into the AC) this value is the one which is loaded.
BREAK	Print the current value of the user's delimiter mask.
BREAK C1	Set the user's delimiter mask to C1. (The use of the delimiter mask is discussed in the chapter on assembly language programming.
DUPLEX	Place the user's Teletype in duplex mode. All characters typed at the keyboard are automatically printed as they are entered.
UNDUPLEX	Take the user's Teletype out of duplex mode. Input characters are received by the Monitor and by the user program without their being printed at the console.
RESTART C1	Set the user program restart address to C1. If CTRL/C is typed at the keyboard, Monitor forces a jump to location C1 in the user's program.
VERSION	Print the version of the Monitor being used.

WRITING ASSEMBLY LANGUAGE PROGRAMS

Introduction

In addition to the higher-level programming languages available in the EduSystem 50 library, the user can also code and run programs written in the PDP-8 assembly language, PAL-D (Program Assembly Language). These programs are prepared with EDIT, assembled with PAL-D, then loaded with LOADER. For those users unfamiliar with assembly language programs, *Introduction to Programming 1972* is a useful guide.

A user can program EduSystem 50 just as he would any other 4K PDP-8. (Assembly language programs must fit in 4K of core.) All memory reference instructions (AND, TAD, ISZ, DCA, JMS, and JMP) function as on a stand-alone PDP-8. All operate instructions (instruction code 7) also function as on a regular PDP-8 (except that microcoding HLT or OSR with any other operate instruction but CLA gives unpredictable results).

The major difference between EduSystem 50 programming and regular PDP-8 programming is in the IOT (input/output transfer) instructions. Some instructions which are valid on stand-alone PDP-8s, such as CDF, CIF, ION, IOF are considered illegal instructions under timesharing. There are a great many new IOTs within EduSystem 50 that are not valid on a regular PDP-8. Finally, there are IOTs which operate on EduSystem 50 in the same manner as on stand-alone PDP-8s. (Table 9-24 is a summary of EduSystem 50 IOT Instructions.)

The way EduSystem 50 actually executes an IOT instruction is also different. Non-IOT instructions (except HLT and OSR) are executed by the hardware, while IOTs (and HLT and OSR) are executed by the EduSystem 50 Monitor.

In general, EduSystem 50 provides the programming capabilities of a 4K PDP-8 and allows programs of considerably greater complexity to be run within the constraints of each user's 4K of core. System Library Programs, all of which were written in assembly language and make use of the EduSystem 50 IOTs dealt with below, are examples of programs which can be run on EduSystem 50.

Console I/O

User programs handle console (Teletype) I/O in almost the same way as stand-alone PDP-8 programs. The KRB instruction is used to input a character, the TLS instruction to output a character. The KSF and TSF (followed by JMP.−1) can be used but are not needed. Monitor handles all timing problems whether these skip IOTs are present or not.

EduSystem 50 differs from the stand-alone PDP-8 in that under EduSystem 50 the user program interacts with multi-character input and output buffers (maintained by Monitor) rather than with single character registers. Depending on the state of the system, these buffers may have one, many, or no characters in them. During normal program execution, this fact is of no consequence. User programs still send and receive characters one at a time. There are times, however, when it is useful to clear out any and all characters in the buffers; a special IOT exists for this purpose (SBC).

On a stand-alone system, characters are input as soon as they are typed, whether they are of immediate interest or not. Usually, these characters are stored by the program until a terminating (or delimiting) character is found. At this time, the whole line of characters is processed. On a swapping, time-sharing system such as EduSystem 50, this mode of operation is wasteful. It is far more efficient to allow input characters to accumulate in the Monitor input buffer until a delimiter is found. There is an IOT to specify which characters are to be considered delimiters (KSB).

EduSystem 50 also allows programs to input and output strings of characters. The read string (KSR) and send string (SAS) instructions provide a convenient and efficient means of doing lengthy transfers.

All keyboard input uses full-duplexed hardware; there is no wired connection between the keyboard and printer (i.e., characters are not printed on the console as typed). Input characters are echoed to the console under program control rather than by hardware. Because input characters are allowed to accumulate in buffers before being passed to the user program, it is important to have Monitor perform the echoing rather than user programs. There is an IOT (DUP) to set up this automatic echoing as well as an IOT (UND) to inhibit echoing for such operations as reading tapes.

Read Keyboard Buffer (KRB) Octal Code: 6036

Operation: Read the next input character into bits 4-11 of the AC.

Load Teleprinter Sequence (TLS) Octal Code: 6046

Operation: The ASCII character in AC bits 4-11 is printed on the user's console.

Skip on Keyboard Flag (KSF) Octal Code: 6031

Operation: The next instruction is skipped if there is a delimiter character in the user's input buffer.

Read Keyboard String (KSR) Octal Code: 6030

Operation: Execution of this instruction initiates a transfer of one or more characters from the user's keyboard to a designated core area. Before executing KSR, load the AC with the address of a two-word block, where:

Word 1: negative of the number of characters to be transferred.

Word 2: address of the core area into which characters are to be placed minus one.

The transfer is terminated when either:

a. the indicated number of characters have been input or
b. a delimiter is seen. At the end of the transfer, the word count and core address are updated and the AC is cleared.

Send A String (SAS) Octal Code: 6040

Operation: Before executing an SAS, load the AC with the address of a two-word block, where:

Word 1: contains the negative of the number of characters to be sent.

Word 2: contains the address −1 of the first word of the string.

The characters are stored one per word right justified starting at the address specified by word 2. Upon execution of SAS, the system takes only as many characters as will fit in the output buffer. It then makes the appropriate adjustment to word 2 to indicate a new

starting address and to word 1 to indicate the reduced character count; it returns to the instruction following the SAS. If the character count is reduced to zero, the instruction following SAS is skipped. The instruction following the SAS should contain a JMP .−2 to continue the block transfer of Teletype characters. The AC is clearly by SAS.

Set Keyboard Break (KSB) Octal Code: 6400

Operation: Rather than activate a user's program to receive each character as it is typed, EduSystem 50 accumulates input characters until a certain character, or characters, is seen. To tell the Monitor which characters to look for (these characters are referred to as delimiters), load the AC with a 12-bit mask before executing a KSB. For each bit in the mask which is set, Monitor considers the corresponding character or characters to be delimiters.

Bit	Specifies
0	0 = check rest of mask 1 = any character is break.
1	301-332 (all letters)
2	260-271 (all numbers)
3	211 (Horizontal tab)
4	212-215 (line feed, vertical tab, form feed, RETURN)
5	241-273 (! " # $ % & ' ( ) * + , − . / : ;)
6	240 (space)
7	274-300 (< = > ? @)
8	333-337 ([/] ↑ ←)
9	377 (RUBOUT)
10	375 (ALT MODE)
11	anything not in bits 1-10

Duplex (DUP) Octal Code: 6402

Operation: DUP informs Monitor that the user wishes each character typed at the console to be echoed on that console's printer as it is received by Monitor. The DUP instruction does not affect the user's registers.

Unduplex (UND) Octal Code: 6403

Operation: UND informs Monitor that the user wishes to suppress character echoing. This can be done for reasons of privacy or because a program does its own character echoing. The user's registers are unaffected by UND.

Set Buffer Control (SBC) Octal Code: 6401

Operation: SBC permits the user program to clear its Teletype input and/or output buffer. Before executing SBC set bits 0 and 1 of the AC as indicated below:

Bit 0	Clear output buffer.
Bit 1	Clear input buffer.

Files and Disk I/O

All user programs can gain access to disk storage. The time-sharing Monitor maintains a pool of available disk space which is allocated in units referred to as segments. (The size of a disk segment varies among installations. Segments may be 128, 256, 521, or 1024 words each.) These segments are used to make up user files on the disk. Monitor also maintains, for each user, a directory of all files which he has defined.

The IOTs which allow the user to access the disk are of two types: those which define files on the disk and those which transfer data between a defined file and the user's core.

NOTE

CREATE and OPEN require that a user set up a file name in core. FINF and WHO return file names to core. Each must be specified in internal code (excess 40 code) as shown in Table 9-23. Characters are packed two to a word.

The first step in defining a file is to create it. Creating a file reserves a single segment of disk storage and associates it with a name. This file can then be extended to any length desired. Extending a file appends more segments to it. Similarly, a file can be reduced by any number of segments. Reducing a file removes the

last segment or segments from the file. Reducing a file to zero segments deletes it entirely. Once created, a file can be protected, thereby restricting access to it. When created, a file can be read by any user, but only the creator can write in it. This protection can be reset if desired. Finally, it is possible to rename an existing file.

None of these actions affect the contents of the file—they only reserve space on the disk. Until it has been written in, the actual content of a file is unspecified. Extending a file does not alter the content of the file as it previously existed. Once defined, files can be used to read and write data. Any number of words (1 to 4096) can be moved from any part of the user's core to any part of a file (subject to file protection). The user program specifies a location in core and a word count. This indicates how many words are to be transferred and from (or to) where in core they are to be moved. Also specified is a disk file address indicating what part of the file is involved. This address is the address of a word in the file. Files are addressed in the same manner as core: in 12-bit words. Unlike core, however, files can be longer than 4K. To address these files provision is made for a 24-bit disk file address, containing the high-order and low-order file addresses.

File addresses are independent of any consideration of segments. The file address is meaningful only in defining files. Files can be read and written across segment boundaries without restriction. (The user cannot read or write beyond the last segment boundary.)

When it executes a file read or write IOT, the system updates the core address and word count and places an error code in the error word (see RFILE) if any error is detected. (The error word must be cleared before executing the IOT.) At the end of a successful transfer, the word count is set to zero and the core address set to the last word transferred. If the transfer cannot be completed for some reason, the word count and core address indicate how much of the transfer was successful; the error word indicates the cause of the failure. All file operations except CREATE (and OPEN) require that the file be open. Up to four files can be open at a time. The process of opening a file associates it with one of four internal file numbers (0, 1, 2, or 3). All file IOTs except CREATE and OPEN, are specified in terms of one of these internal file numbers, rather than a file name. IOTs operate on the file which is indicated by that internal file number at the time. It is therefore possible to

write file handling programs which are independent of the actual file(s) they operate on.

File IOTs, that are successfully completed, return with the AC cleared. If an error was found which prohibited execution of the IOT, one of the following error codes is returned:

Code	Explanation
4000	There was no file opened on the specified internal file number.
4400	Attempting to redefine a file which is open to another user.
5000	Attempting to create a file for a user whose directory is full.
6000	File protection violation.
6400	Invalid file name.
7000	Attempting to open a nonexistent file.
7400	Disk is full.

Create a File (CRF) Octal Code: 6610

Operation: The user can request the system to create a new file of one segment. The user program provides the new name for the file. Load the AC with the beginning address of a 3-word block, where:

Words 1 through 3: contain the 6-character name.

If there is some reason why the request cannot be granted, the system will return a non-zero error code in the AC. The protection code of a newly created file is 12.

Extend A File (EXT) Octal Code: 6611

Operation: To extend the length of an existing file, that file must be currently open. Load the AC with the beginning address of a 2-word block, where:

Word 1: contains the internal file number of the file to be extended.

Word 2: contains the number of segments the system should append to the file.

If for some reason the request to extend a file cannot be granted, the AC will contain 4000, 4400, 6000, or the number of segments it failed to append.

Reduce A File (RED) Octal Code: 6612

Operation: To reduce the length of an existing file, that file must be currently open. Load the AC with the beginning address of a 2-word block, where:

Word 1: contains the internal file number of the file to be reduced.

Word 2: contains the number of segments to be removed.

This request is granted unless the file to be reduced is currently opened to another user or if the file is write protected against the user.

Rename A File (REN) Octal Code: 6600

Operation: REN is used to change the name of a file. Load the AC with the address of a 4-word block where:

Word 1: contains the internal file number associated with the file whose name is to be changed.

Words 2-4: contains the new name. This name is in 6-bit characters packed two in a word.

Protect A File (PROT) Octal Code: 6604

Operation: The owner of a file can protect his file from unauthorized attempts to access it by using this instruction. Before executing PROT, load the AC with:

Bits 5 and 6 Internal file number of the reserved file to be protected.

Bit 7 Write protect against owner.

Bit 8 Write protect against users whose project number is same as owner's.

Bit 9	Read protect against users whose project number is same as owner's.
Bit 10	Write protect against users whose project number differs from owner's.
Bit 11	Read protect against users whose project number differs from owner's.

A file must be opened before it can be protected. PROT is legal only when performed by the file owner, i.e., the user who created the file. All attempts to access the file which violate any of the protection flags are considered illegal. (For further information on project numbers, see the section on Storage Allocation.)

Open A File (OPEN) — Octal Code: 6601

Operation: OPEN is used to associate a file with an internal file number, which is necessary because all file operations are in terms of the internal file numbers. Before executing the OPEN IOT, load the AC with the beginning address of a 5-word block, where:

Word 1:	contains the internal file number.
Word 2:	contains the account number of the owner of the file. If 0, the account number of the current user is specified.
Word 3-5:	contain the name of the file to be opened. This name is in 6-bit characters packed two to a word.

If there was another file associated with the internal file number before the execution of the OPEN IOT, it is closed automatically before the new file is associated with the internal file number.

Close A File (CLOS) — Octal Code: 6602

Operation: CLOS terminates the association between files and their internal file numbers. Before executing CLOS, load the AC with a selection pattern for the internal file numbers whose associated files are to be closed. The file is closed if bit I is 1, where I = bit 0, 1, 2, or 3.

READ File (RFILE) and Write File (WFILE)

Octal Code: 6603 & 6605

Operation: Once the association of a file with an internal file number has been made, these IOTs allow the actual file reference to be made. They are illegal on a file that has not been opened (associated with an internal file number).

To read or write a file, load the AC with the address of a 6-word block, then execute the IOT. The format for the 6-word block is:

Word 1: contains the high-order file word address.

Word 2: contains the internal file number.

Word 3: contains the negative of the number of words for the operation. This number is either the number of words to be read or the number of words to be written.

Word 4: contains a pointer to the beginning address −1 of a buffer located in the user program. On a read operation this buffer receives the information from the file: on a write operation this buffer holds the information that is to be sent to the file.

Word 5: contains the least significant 12 bits of the initial file word address to begin the operation.

Word 6: contains an error code:

0 if no error
1 if parity error
2 if file shorter than word count
3 if file not open
4 if protection violated

The read or write begins at the word specified by words 1 and 5. For example:

```
        TAD X
        WFILE

X,      .+1
        0
        1
        -200
        6477
        200
```

means: write 200 (octal) words starting at word 200 of the file that is associated with internal file number one from a core area starting at location 6500.
After completion of the transfer, the word count (word 3) and core address (word 4) are updated. If an error was detected the appropriate error code is placed in word 6.

File Information (FINF) Octal Code: 6613

Operation: FINF enables a user program to determine what file, if any, is associated with an internal file number. Load the AC with the beginning address of a 7-word block, where:

Word 1: contains the internal file number for which the user program wishes information.

Words 2 through 7: contain the information that the system returns after executing FINF.

Word 2: contains the account number of the owner or zero if no file is associated with the internal file number, that is, the file is not open.

Words 3-5: contains the name of the file in 6-bit code.

Word 6: contains:

Bit 1	Means
7	write protected against owner
8	write protected against users whose project number is same as owner's
9	read protected against users whose project number is same as owner's
10	write protected against users whose project number differs from owner's
11	read protected against users whose project number differs from owner's

Word 7: contains the number of segments which compose the file.

Assignable Devices

Users can access both their own Teletype and the disk; with the remaining system devices (referred to as the assignable devices) this is not true. One function of the Monitor is to ensure that device usage never conflicts. Only one user at a time can access the high-speed paper-tape reader or punch, or any one of the DECtapes. To ensure that only one user can access a device, EduSystem 50 requires that the device be assigned before it is used. After a device is assigned, it is not available until it is released by its owner.

Once assigned, the device is programmed exactly as on a standalone PDP-8. The RRB instruction is used to read a character from the high-speed reader; the PLS instruction is used to punch one on the high-speed punch. The skip IOTs (RFS and PSF) can be used (followed by JMP .−1) but are not necessary. For block transfers, there are two string transfer commands: RRS and PST.

The DECtape instructions have been simplified. A single instruction, DTXA, initiates the transfer of a block of data. The DTRB instruction is then used to determine if the transfer was successful. The skip instruction, DTSF, can be used (followed by JMP .−1) but is not necessary.

Executing any of the assignable device IOTs without first assigning the device gives the following results: (a) If the device is assigned to another user, the instruction is considered illegal; program execution is now terminated and an error message printed; (b) If the device is available it is automatically assigned before execution of the IOT. The device then belongs to this user until he releases it.

Because these devices are shared by all users, the Monitor must ensure that they are operable at all times. In particular, the Monitor must ensure that a user is not waiting for a device which is not available. This situation can arise when trying to use the punch when it is turned off, or when the reader has read off the end of a tape. All these conditions, known as "hung devices" are considered to be system errors. If the program doing the transfer has been enabled for system errors (by executing an SEA), control transfers to the error routine indicated which must clear the error flag in the status word before continuing (See Program and System Status). If the user program has not been enabled for system errors, a hung device causes the program to be terminated and an error message is printed.

Assign Device (ASD) Octal Code: 6440

Operation: If the device specified by the content of the AC is available, it is assigned to the user program and the AC is cleared. Otherwise, the number of the job owning the device is placed in the AC. If the device does not exist, 7777 is returned in the AC.

4000	Paper-tape reader
4001	Paper-tape punch
4005 + N	DECtape unit N

The assignment is in effect until a corresponding REL instruction or LOGOUT.

Release Device (REL) Octal Code: 6442

Operation: The device specified by the contents of the AC is released (providing it was owned by the user executing the REL). The AC is cleared. Releasing a device makes it available to other users.

Skip on Reader Flag (RSF) Octal Code: 6011

Event Time: 1

Operation: The reader flag is sensed, and if it contains a binary 1, the contents of the PC are incremented by one so that the next sequential instruction is skipped. The reader flag is bit 8 of status register 1, and has a value of 1 if the reader buffer is not empty.

Read Reader Buffer (RRB) Octal Codes: 6012 & 6016

Event Time: 2

Operation: The contents of the reader buffer are transferred into bits 4 through 11 of the AC and the reader flag is cleared if the reader buffer is empty. This instruction does not clear the AC. If the reader buffer is empty, the user program is dismissed until the reader flag is 1 or an end-of-tape condition is detected.

Table 9-23. EduSystem 50 Internal Character Set

Character	6-Bit[14] Octal	8-Bit Octal	Character	6-Bit[14] Octal	8-Bit Octal
Space	00	240	@	40	300
!	01	241	A	41	301
"	02	242	B	42	302
#	03	243	C	43	303
$	04	244	D	44	304
%	05	245	E	45	305
&	06	246	F	46	306
'	07	247	G	47	307
(	10	250	H	50	310
)	11	251	I	51	311
*	12	252	J	52	312
+	13	253	K	53	313
,	14	254	L	54	314
-	15	255	M	55	315
.	16	256	N	56	316
/	17	257	O	57	317
0	20	260	P	60	320
1	21	261	Q	61	321
2	22	262	R	62	322
3	23	263	S	63	323
4	24	264	T	64	324
5	25	265	U	65	325
6	26	266	V	66	326
7	27	267	W	67	327
8	30	270	X	70	330
9	31	271	Y	71	331
:	32	272	Z	72	332
;	33	273	[	73	333
<	34	274	\	74	334
=	35	275	]	75	335
>	36	276	↑	76	336
?	37	277	←	77	337

[14] The 6-bit octal code is used to store passwords and file names only.

Reader Fetch Character (RFC) Octal Code: 6014

Event Time: 3

Operation: The reader flag and the Monitor reader buffer are both cleared, the reader is started to fill the Monitor reader buffer and the reader flag is set after the buffer is full or the end of tape is detected.

Read Reader String (RRS) Octal Code: 6010

Operation: This instruction initiates a transfer from the high-speed reader to a selected area in the user's core. Before executing RRS, load the AC with the address of a 2-word block, where:

Word 1: minus the number of characters to be transferred.

Word 2: the address of the user core area minus one.

The transfer is terminated by either of two conditions: (a) the word count is zero indicating that the required number of characters have been read or (b) the reader has read off the end of the tape (a system error condition). In either case, the word count and core address are updated. RRS clears the AC.

Load Punch Buffer Sequence (PLS) Octal Code: 6026

Operation: The ASCII character is in AC bits 4 through 11 and is transmitted to the high-speed punch. PLS does not clear the accumulator.

Skip on Punch Flag (PSF) Octal Code: 6021

Event Time: 1

Operation: The punch flag is sensed, and if it contains a binary 1, the contents of the PC is incremented by one so that the next sequential instruction is skipped. The punch flag is bit 9 of status register 1, and has a value of 1 if the punch buffer is not full. If the punch flag is 0, the program is dismissed until the punch flag is 1.

Punch String (PST) Octal Code: 6020

Operation: PST allows a user program to punch a string of characters. Before executing PST, load the AC with the beginning address of a 2-word block, where:

Word 1: contains the negative of the number of characters to be punched.

Word 2: contains the beginning address −1 of the string to be punched; the characters should be right justified one per word.

After execution of PST, the system takes only as many characters as fit in the punch buffer; it then makes the appropriate adjustment to word 2 to indicate a new starting address and to word 1 to indicate the reduced character count. It returns to the instruction following the PST which should be a JMP .−2 to continue the transfer. If the character count is reduced to zero, the instruction following PST is skipped. The AC is cleared by PST.

Load Status Register A (DTXA) Octal Code: 6764

Operation: DTXA allows a user program to read and write records (129-word blocks) on DECtape. Load the AC with the beginning address of a 3-word block, where:

Word 1: contains:

Bit 1	Means
0-2	contains the transport unit select number,
3-5	0,
6-8 = 2	for read data function,
4	for write data function,
9-11	0.

Word 2: contains the DECtape block number.

Word 3: contains the beginning core address −1 of record buffer.

After DTXA is given, the DECtape request is placed in the DECtape request queue. After the completion of any DECtape request, the DECtape flag in status register 2 is turned on. DTXA does not update word 3. The AC is cleared by DTXA.

Skip on Flags (DTSF) Octal Code: 6771

Operation: The content of both the error flag and the DECtape flag is sampled, and if either flag contains a binary 1, the content of the PC is incremented by one to skip the next sequential instruction. If both flags are zero, the user program is dismissed until the skip is satisfied.

Read Status Register B (DTRB) Octal Code: 6772

Operation: The content of DECtape status register B is loaded into the AC by an OR transfer. The AC bit assignments are:

Bit	Assignment
0	error flag
1	mark track error
2	end of tape
3	select error
4	parity error
5	timing error
6-10	unused
11	DECtape flag

Program Control

There are a number of ways that the status of a running program can be changed. The program can be terminated in one of three ways: by execution of a HLT, by the user typing ↑BS to force a program halt, or by a program error which forces Monitor to terminate the program after printing an error message.

It is also possible for the status of a running program to change without it being terminated. First, the user program can request that it handle its own program error conditions. In this case, Monitor does not terminate a job on an error; instead, it transfers control to a user error handler. This error handler then determines what the error was, by a CKS instruction and takes appropriate action. Monitor also provides the program with an interrupt key, ↑C. If the user types a ↑C, the Monitor unconditionally transfers

control to a restart address. Thus, the user program can handle its own restarts.

Halt (HLT) Octal Code: 7402

Operation: This instruction is used to stop the user program and return control to Monitor. Executing HLT is equivalent to typing ↑BS followed by RETURN.

Set Restart Address (SRA) Octal Code: 6417

Operation: This instruction allows the user to specify an address to which control is transferred when an ↑C is typed on the user's console. Load the AC with the restart address and execute SRA. If ↑C is detected, the program's input and output buffers are cleared, the AC and Link are cleared and control goes to the restart address.

Set Error Address (SEA) Octal Code: 6431

Operation: This instruction allows the user to specify an address to which control is transferred in the event of a system error. Load the AC with an address before executing SEA. If a system error is detected, Monitor simulates a JMS to the error address. The program counter is stored in the error address and control transferred to the error address +1. AC, Link, and input/output buffers are not affected. The error code of the system error is in STR0 bits 9-11. Bit 10 of STR1 is set. The error routine must read these bits (by a CKS) to determine the cause of the error, then clear them by means of a CLS.

The only error code that occurs in the course of normal system usage is due to a hung device. This error occurs when the user attempts to use a punch not already turned on, access a DECtape not yet selected, or allows the paper-tape reader to run off the end of a tape. The error routine must release the device to clear the error condition. The illegal IOT error probably means that an assignable device IOT was executed without the device first being assigned. Swap and file errors occur if a hardware error is detected while Monitor is swapping user programs or while reading or writing file directories. These are system malfunctions from which there is no recovery.

Program and System Status

Because EduSystem 50 programs run under control of a time-sharing Monitor, it is important for them to determine their status within the system and the status of the system as a whole. Several IOTs, listed below, have been defined for this purpose.

Check Status (CKS) Octal Code: 6200

Monitor maintains for each user a complete set of status information, his program's running status and the state of his input/output devices. This status information, stored in three words, can be accessed by a running program with the CKS instruction. Before executing a CKS, load the accumulator (AC) with the address of a three-word block. Executing CKS stores the three status words (STR0, STR1, and STR2) in the three-word block and clears the AC. Information about the status of individual devices can also be checked by the skip IOTs.
The formats of these registers are:

STR0 Bits

0	Run Bit	User program is in the run state
1	Error Enable	Program handles its own errors
2-4		Unused
5	JSIOT	System use only
6	JSIOTC	System use only
7	JSEXON	System use only
8		Unused
9-11	Error Code	System detected error condition 1 Illegal IOT 2 Swap read error 3 Swap write error 5 Disk file error 6 Hung device

STR1 Bits

0	Timer	Time is up
1	File 0	Internal file 0 is not busy
2	File 1	Internal file 1 is not busy
3	File 2	Internal file 2 is not busy
4	File 3	Internal file 3 is not busy

STR1 Bits

5	Delimiter	There is a delimiter in the input buffer
6		Unused
7	Teleprinter	Output buffer is not full
8	Reader	Character in reader buffer
9	Punch	Punch buffer is not full
10	Error	System error detected, code in bits 7 through 11 of STR0
11	Wait	Job is not waiting

STR2 Bits

0-8		Unused
9	DECtape	DECtape transfer requested
10	Error	DECtape error
11	DECtape	DECtape flag

Each user has available to him a 12-bit switch register just as he does on a stand-alone PDP-8. This switch register can be manipulated by means of the Monitor command SWITCH or under program control.

OR With Switch Register (OSR) Octal Code: 7404

Operation: The content of the user's switch register is inclusively ORed into the AC.

Set Switch Register (SSW) Octal Code: 6430

Operation: The content of the AC is stored in the user's switch register. The AC is cleared. Assembly language programs run under control of the Monitor. The following IOTs are defined to allow a program to determine the status of the system as a whole.

Segment Size (SIZE) Octal Code: 6614

Operation: The segment is the basic unit of on-line file storage. Different EduSystem 50's have differing segment sizes. SIZE allows a program to determine the segment size. The segment size is returned in the AC.

Segment Count (SEGS) Octal Code: 6406

Operation: The number of available disk segments is returned in the AC.

Account (ACT) Octal Code: 6617

Operation: The account number (of the job number given) is returned in the AC. If AC is 0, the account number for the current job is returned. If the requested job does not exist, zero is returned.

Who (WHO) Octal Code: 66-16

Operation: The account number and password of the current job are returned to the 3-word block whose address is in the AC and the AC is cleared.

User (USE) Octal Code: 6421

Operation: Return in the AC the number of the current job.

Console (CON) Octal Code: 6422

Operation: Return in the AC the console unit number assigned to the job whose number is in the AC, if that console number does not exist, −1 is returned.

User Run Time (URT) Octal Code: 6411

Operation: Load the AC with the address of a 3-word block, where word 1 contains the number of the job for which the run time is sought. The run time is returned in the last two locations of the block. If job 0 is specified, the run time of the current job is returned. The AC is cleared.

Time-of-Day (TOD) Octal Code: 6412

Operation: Returns the value of the System Clock in military time (using a 24-hour cycle) in the two locations starting at the location of the address in the AC. The AC is cleared.

Return Clock Rate (RCR) Octal Code: 6413

Operation: The number of clock ticks per second is returned in the AC.

Date (DATE) Octal Code: 6414

Operation: Returns the date in the AC. The format of this 12-bit number is:

DATE=((YEAR-1964)*12+(MONTH-1))*31+DAY−1

Skip on EduSystem 50 (TSS) Octal Code: 6420

Operation: This instruction is used by programs which run under both EduSystem 50 and on a standard PDP-8. Under EduSystem 50, the instruction following TSS is skipped and the Monitor version number is returned in the AC. On a standard PDP-8, the IOT has the effect of an NOP instruction.

Quantum Synchronization (SYN) Octal Code: 6415

Operation: Upon execution of this instruction, the system dismisses the user program and sets it in the run state so that it will be run again in turn. Ordinarily, this instruction is used to ensure a full time quantum to perform some critical operation.

Set Time (STM) Octal Code: 6416

Operation: The system provides a clock time for each user program. By means of this IOT, the time can be set to "fire" after a specified number of clock ticks have elapsed. Load the AC with the time (in seconds) to prime the timer. Upon execution of the STM instruction, the system sets the time to "fire" in the specified number of seconds and turns the time bit (bit 0) in status register 1 to 0, clears the AC, and dismisses the job. After the specified time has elapsed, the system turns bit 0 back to 1, and the job is restarted.

PDP-8 Compatibility

Programming EduSystem 50 in assembly language is very similar to programming a stand-alone PDP-8. All instructions except the IOTs operate identically in either case. As discussed previously, programming such devices as the Teletype and high-speed reader/punch for EduSystem 50 is somewhat simpler. EduSystem 50 runs programs which include timing loops. It also properly executes the IOTs not mentioned in this manual, e.g., TCF, PCF, etc. Thus, programs written for stand-alone PDP-8s with Teletype

and high-speed reader or punch will run on EduSystem 50, although generally not as efficiently as programs which are written specifically for EduSystem 50.

The same is not true for disk and DECtape operations because EduSystem 50 uses a simplified programming structure for these devices. The actual differences in coding are very small. It is a simple task to adapt previously written code for EduSystem 50 disk and DECtape.

There are a few standard changes which users generally make in adapting PDP-8 code to EduSystem 50. Monitor does the echoing rather than the user program. The TLS which does the echo can be deleted and a DUP instruction added somewhere near the start of the program. Also, for efficiency, the EduSystem 50 delimiter capability can be used. A KSB in the program determines what the delimiters are.

Many PDP-8 programs execute a reader and punch IOT early in the program, to initialize the device, whether they are actually to be used or not. If the devices are free, they can be assigned and thus made unavailable to other users. If they are unavailable, the program terminates on an illegal IOT. Thus, it is important not to execute these IOTs randomly. If disk or DECtape is involved, the actual transfer code must be altered to conform to EduSystem 50. (The fact that core page 37, locations 7600 through 7777, is available to EduSystem 50 programs is useful in making these changes. New code can be placed in the area normally reserved for the Binary Loader.)

The most difficult code to convert is that code which operates under interrupt. To be run under EduSystem 50, these programs must be recoded so as not to use the interrupt.

IOTs for nonexistent devices are ignored as are CDFs and CIFs to field zero. (Other CDFs and CIFs are illegal.) It must be remembered that many IOTs have been redefined for use as special EduSystem 50 instructions. In all other situations, EduSystem 50 remains compatible with stand-alone systems whenever possible.

Table 9-24. EduSystem 50 IOT Instruction Summary

Number	Instruction	Function
Program Control		
6200	CKS	Check Status
6402	DUP	Duplex Console
6403	UND	Unduplex Console
6405	CLS	Clear Status
6411	URT	User Run Time
6412	TOD	Time of Day
6413	RCR	Return Clock Rate
6414	DATE	Date
6415	SYN	Quantum Synchronization
6416	STM	Set Timer
6417	SRA	Set Restart Address
6420	TSS	Skip on TSS/8
6421	USE	User
6422	CON	Console
6430	SSW	Set Switch Register
6431	SEA	Set Error Address
6440	ASD	Assign Device
6442	REL	Release Device
7402	HLT	Halt
7404	OSR	OR With Switch Register
File Control		
6406	SEGS	Segment Count
6600	REN	Rename File
6601	OPEN	Open File
6602	CLOS	Close File
6603	RFILE	Read File
6604	PROT	Protect File
6605	WFILE	Write File
6610	CRF	Create File
6611	EXT	Extend File
6612	RED	Reduce File
6613	FINF	File Information
6614	SIZE	Segment Size
6616	WHO	Who
6617	ACT	Account Number

Table 9-24 (Cont.). EduSystem 50 IOT Instruction Summary

Number	Instruction	Function
Input Buffer Control		
6030	KSR	Read Keyboard String
6031	KSF	Skip On Keyboard Flag
6032	KCC	Clear Keyboard Flag
6034	KRS	Read Keyboard Buffer Static
6036	KRB	Read Keyboard Buffer Dynamic
6400	KSB	Set Keyboard Break
6401	SBC	Set Buffer Control Flags
Output Buffer Control		
6040	SAS	Send A String
6041	TSF	Skip On Teleprinter Flag
6042	TCF	Clear Teleprinter Flag
6044	TPC	Load Teleprinter and Print
6046	TLS	Load Teleprinter Sequence
High-Speed Paper Tape Reader and Control (Type PC02)		
6010	RRS	Read Reader String
6011	RSF	Skip On Reader Flag
6012	RRB	Read Reader Buffer
6014	RFC	Reader Fetch Character
High-Speed Paper Tape Punch and Control (Type PC03)		
6020	PST	Punch String
6021	PSF	Skip On Punch Flag
6022	PCF	Clear Punch Flag
6024	PPC	Load Punch Buffer and Punch Character
6026	PLS	Load Punch Buffer Sequence
DECtape Control (Type TC01)		
6764	DTXA	Load Status Register A
6771	DTSF	Skip On Flags
6772	DTRB	Read Status Register B

STORAGE ALLOCATION

Storage Map

The system's storage allocation is illustrated below.

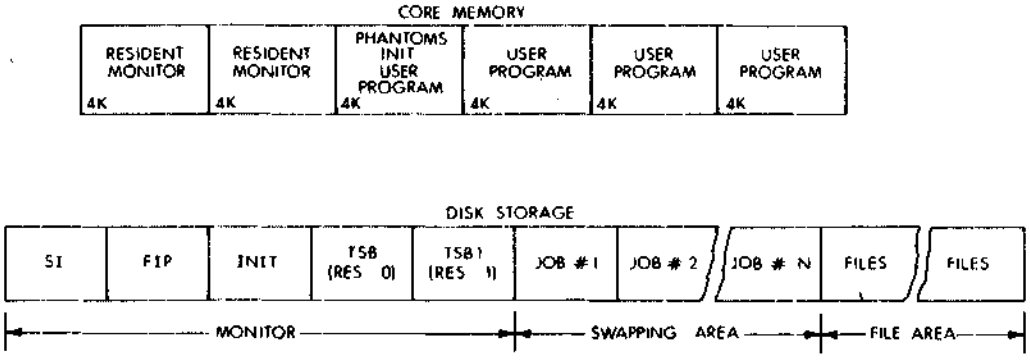


Figure 9-5. EduSystem 50 Storage Map

File Directories

There are two directories on the disk: the Master File Directory (MFD) referenced mainly by the system, and the User File Directory (UFD), referenced by the user. One of the functions of the MFD is to service the UFD. A UFD is a particular user's file directory containing the names of programs he has created on the disk.

The UFD is a file like any other file except that its filename is the project-programmer number and password. When a user is logged in under a specific number and references the disk, he is actually referencing his own file area on the disk through the UFD which has his project-programmer number as its name. He can specifically code his routine to reference UFDs of other users or the MFD; whether he is successful or not depends on the type of protection that has been specified for the area he is trying to reference.

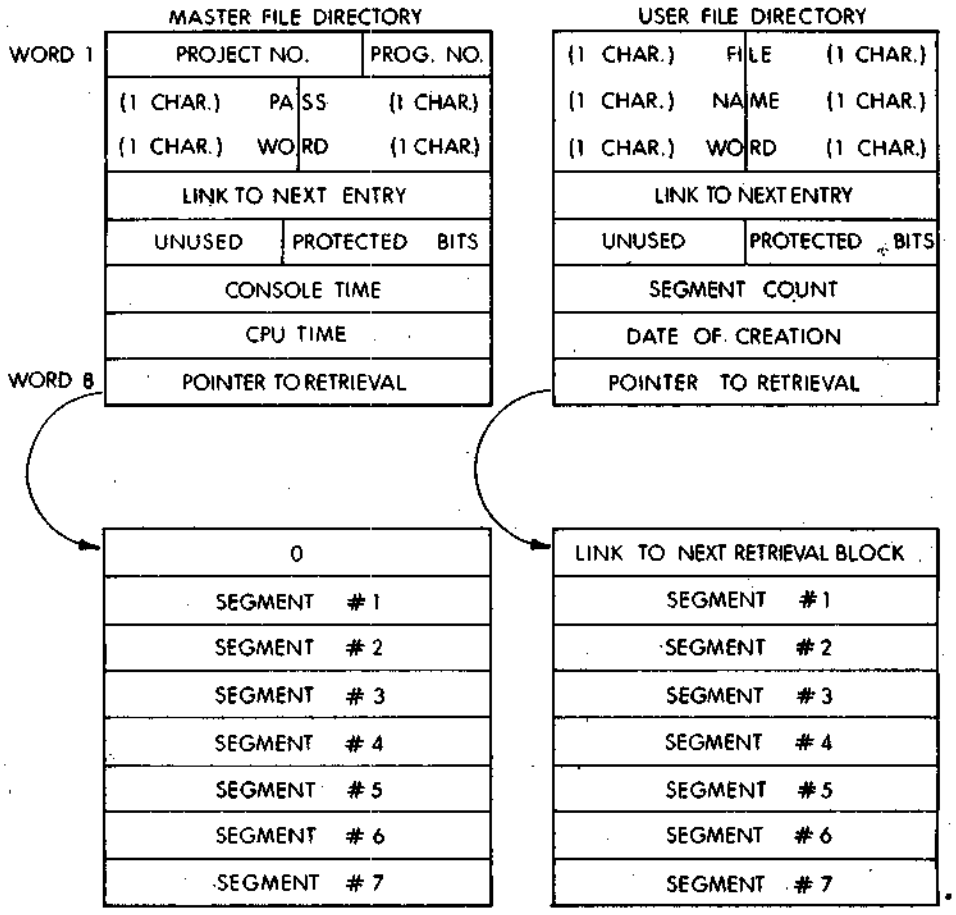


Figure 9-6. File Directories

Project-Programmer Numbers

System account numbers are a combination of project number and programmer number. The account number (always written in octal) has a binary equivalent. When expressed as a 12-bit binary number, the project number is formed by the leftmost 7 bits and the programmer number is the rightmost 5 bits. For example, the account number 623 (octal) equals 000 110 010 011 (binary). The left 7 bits (leading zeros included if any) are 0001100 (binary) and equal 14 (octal) which is the project number. The programmer number is 10011 (binary) or 23 (octal). Therefore a user with account number 623 has: project number 14 and programmer number 23.

appendix a
character codes

ASCII [1] Character Set

Character	8-Bit Octal	6-Bit Octal	Decimal Equivalent (A1 Format)	Character	8-Bit Octal	6-Bit Octal	Decimal Equivalent (A1 Format)
A	301	01	96	!	241	41	—1952
B	302	02	160	"	242	42	—1888
C	303	03	224	#	243	43	—1824
D	304	04	288	$	244	44	—1760
E	305	05	352	%	245	45	—1696
F	306	06	416	&	246	46	—1632
G	307	07	480	'	247	47	—1568
H	310	10	544	(	250	50	—1504
I	311	11	608	)	251	51	—1440
J	312	12	672	*	252	52	—1376
K	313	13	736	+	253	53	—1312
L	314	14	800	,	254	54	—1248
M	315	15	864	-	255	55	—1184
N	316	16	928	.	256	56	—1120
O	317	17	992	/	257	57	—1056
P	320	20	1056	:	272	72	—352
Q	321	21	1120	;	273	73	—288
R	322	22	1184	<	274	74	—224
S	323	23	1248	=	275	75	—160
T	324	24	1312	>	276	76	—96
U	325	25	1376	?	277	77	—32
V	326	26	1440	@	300		32
W	327	27	1504	[	333	33	1760
X	330	30	1568	\	334	34	1824
Y	331	31	1632	]	335	35	1888
Z	332	32	1696	↑(^)[2]	336	36	1952
0	260	60	—992	←(_)[2]	337	37	2016
1	261	61	—928	Leader/Trailer	200		
2	262	62	—864	LINE FEED	212		
3	263	63	—800	Carriage RETURN	215		
4	264	64	—736	SPACE	240	40	—2016
5	265	65	—672	RUBOUT	377		
6	266	66	—608	Blank	000		
7	267	67	—544	BELL	207		
8	270	70	—480	TAB	211		
9	271	71	—416	FORM	214		

[1] An abbreviation for American Standard Code for Information Interchange.

[2] The character in parentheses is printed on some Teletypes.

appendix a
read-in mode loader

The Read-In Mode (RIM) Loader is the first program loaded into an EduSystem computer.[1] This program is loaded by toggling 17 instructions into core memory using the console SWITCH REGISTER (SR). The RIM Loader instructs the computer to receive and store, in core, data punched on paper tape in RIM coded format—primarily the EduSystem system tapes.

There are two RIM Loader programs: one is used when the input is to be from the low-speed (Teletype) paper tape reader; the other is used when input is to be from the high-speed paper tape reader. The locations and corresponding instructions for both programs are listed in Table A-1. The procedure for loading (toggling) the RIM program into core is illustrated in Figure A-1. The RIM Loader is loaded into field zero of core.

Table A-1. RIM Loader Programs

	INSTRUCTION	
Location	Low-Speed	High-Speed
7756	6032	6014
7757	6031	6011
7760	5357	5357
7761	6036	6016
7762	7106	7106
7763	7006	7006
7764	7510	7510
7765	5357	5374
7766	7006	7006
7767	6031	6011
7770	5367	5367
7771	6034	6016
7772	7420	7420
7773	3776	3776
7774	3376	3376
7775	5356	5357
7776	0000	0000

[1] The RIM Loader is not needed if the EduSystem has a hardware bootstrap.

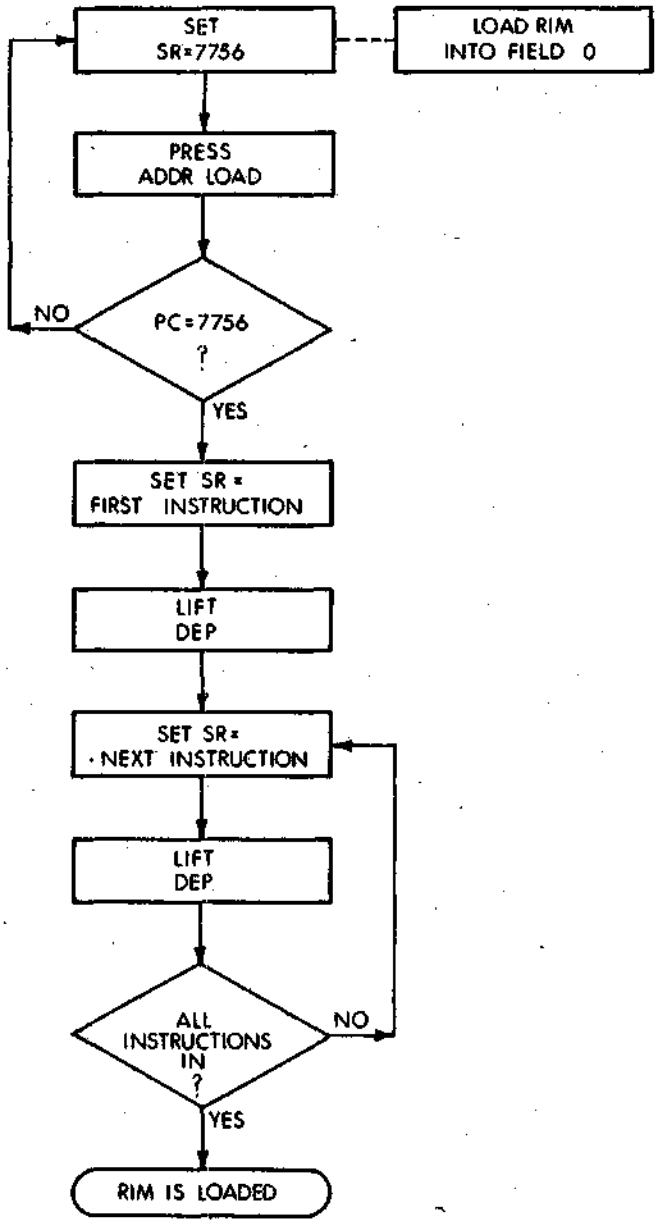


Figure A-1. Loading the RIM Loader

After RIM has been loaded, it is good programming practice to verify that all instructions were stored properly. This can be done by performing the steps illustrated in Figure A-2, which also shows how to correct an incorrectly stored instruction.

When loaded, the RIM Loader occupies absolute locations 7756 through 7776. EduSystems do not use the RIM locations; therefore, RIM need not be reloaded unless the contents of the RIM locations have been altered by the user.

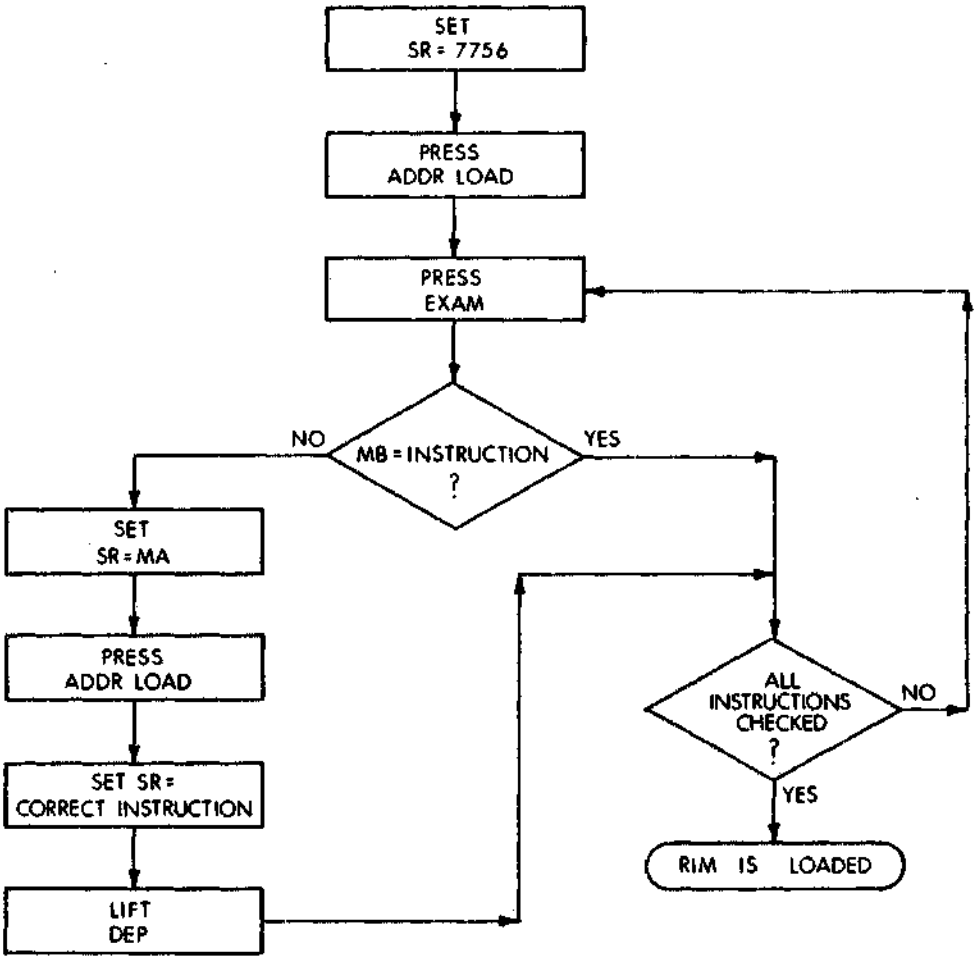


Figure A-2. Checking the RIM Loader

appendix b
character codes

The ASCII[1] character codes shown in the following table are used by EduSystems as the argument in the CHR$ function. For each ASCII code a second acceptable form is permitted in CHR$. The second code is obtained by adding 128 to the code given in the following table. For example, CHR$ would print A in response to either 65 or 193 as an argument. These codes are also used with the CHANGE statement in EduSystem 50.

Character	ASCII Code No. (Decimal)	Character	ASCII Code No. (Decimal)
linefeed	10		
formfeed	12		
RETURN	13		
space	32	@	64
!	33	A	65
"	34	B	66
#	35	C	67
$	36	D	68
%	37	E	69
&	38	F	70
'	39	G	71
(	40	H	72
)	41	I	73
*	42	J	74
+	43	K	75
,	44	L	76
-	45	M	77
.	46	N	78
/	47	O	79
0	48	P	80
1	49	Q	81
2	50	R	82
3	51	S	83
4	52	T	84
5	53	U	85
6	54	V	86
7	55	W	87
8	56	X	88
9	57	Y	89
:	58	Z	90
;	59	[	91
<	60	\	92
=	61	]	93
>	62	↑	94
?	63	←	95

[1] An abbreviation for American Standard Code for Information Interchange.

appendix C

edusystem 50 monitor command summary

MONITOR COMMANDS

A Monitor command is a string of characters terminated by a semicolon (;), a colon (:), or a carriage return (RETURN key). Parameters of commands can be octal numbers, decimal numbers, character strings, or single letters. In the following summary, parameters are coded as follows:

C1, C2, . . .	represent octal numbers
D1, D2, . . .	represent decimal numbers
S1, S2, . . .	represent character strings
L1, L2, . . .	represent single letters

Logging In and Out

LOGIN C1 S1	Request to login: C1 = user's account number S1 = user's password
LOGOUT	Request to logout: processing and console time are printed.
TIME C1	Request printout of processing time: C1 = job number Without C1, current job is assumed: before logging in and without C1, time-of-day is typed out; If C1 = job 0, time-of-day is printed.

Device Allocation

ASSIGN L1	Access device: L1 = R for paper tape reader P for paper tape punch D for any DECtape unit L for line printer

ASSIGN D C1	Access DECtape unit; D = DECtape C1 = device number
RELEASE L1	Release device: L1 = R, P (see ASSIGN L1)
RELEASE D C1	D = DECtape unit C1 = DECtape number

File Handling

OPEN C1 S1 C2	Establish association between internal file number and file: C1 = internal file number S1 = file name C2 = account number
CLOSE S1	Close files: S1 = list of internal file numbers
CREATE S1	Create new file: S1 = name of new file
RENAME C1 S1	Rename a file: C1 = internal file number S1 = new name of file
REDUCE C1 D1	Reduce length of file: C1 = internal file number D1 = number of segments to be removed from end of file
EXTEND C1 D1	Extend length of file: C1 = internal file number D1 = number of segments to be added to end of file
PROTECT C1 C2	Protect a file: C1 = internal file number C2 = new file protection mask 1 read protect against users with different project number

	2 write protect against users with different project number 4 read protect against users with same project number 10 write protect against users with same project number 20 write protect against owner or the sum of any combination
F C1	Print association between internal file numbers and files: C1 = internal file number

Control of User Programs

START C1	Execute user program: C1 = starting location
START	Restart user program.
RESTART C1	Set program restart address.
DEPOSIT C1 C2 . . . Cn	Store in core memory: C1 = location C2 = contents to be stored Cn = location C1+n−1 n ⩽ 10 decimal
EXAMINE C1 D1	List specified contents: C1 = first location D1 = number of location to be listed D1 ⩽ 10 decimal

Utility Commands

SAVE S1 SAVE C1 S1 C2 SAVE C1 S1 C2 C3 SAVE C1 S1 C2 C3 C4	Save core image: C1 = owner's account number S1 = name of file C2 = file address of first word to be saved; if not specified, entire 4K is saved

	C3 = core address of first word to be saved; if not specified, entire core is saved C4 = core address of last word to be saved; if not specified, entire core is saved
LOAD C1 S1 LOAD C1 S1 C2 LOAD C1 S1 C2 C3 LOAD C1 S1 C2 C3 C4	Load core image: C1 = owner's account number S1 = name of file C2 = file address of first word to be loaded; if not specified, entire 4K is loaded C3 = core address of first word to be loaded, if not specified, entire core is loaded C4 = core address of last word to be loaded; if not specified, entire core is loaded
R S1	Run system file: S1 = name of file
RUN S1	Run user file: S1 = name of file
RUN C1 S1	Run user file: C1 = owner's account number S1 = name of file
S	Stop execution.
WHERE	Print contents of location counter, accumulator, link, and switch register.
USER	Print number of the job and devices owned.
USER C1	Print device numbers: C1 = user's account number
SWITCH C1	Set switch register: C1 = word to be set

BREAK C1	Set keyboard break mask: C1 = new mask
DUPLEX	Echo typed characters on printer.
UNDUPLEX	Ignore previous DUPLEX command.
TALK C1 S1	Send a message to console C1: C1 = destination console S1 = message

appendix d
edutest user's guide

INTRODUCTION

EduTest is a system for grading and analyzing test responses. The EduTest system is designed to run on a minimum configuration of EduSystem 30 or EduSystem 40. EduTest offers two benefits to teachers:

1. Less work in grading tests and analyzing results.
2. Increased information on student performance, question difficulty, and class response patterns.

TEACHER'S GUIDE

Test Characteristics

Any teacher-made multiple choice test can be scored, analyzed, and graded with EduTest. The program handles a maximum of 100 questions with a choice of five answers labeled A through E. Tests in excess of 25 questions are subdivided into parts of 25 questions each.

A, B, C, D, and E are the available choices for each answer. The types of questions may vary; True-False questions (answers A and B) and multiple choice questions (using A through E answers) may be intermixed.

EduTest Output

The EduTest System produces a variety of output reports helpful in analyzing test results and evaluating a test and its answers. The standard statistical measures of mean, median, and standard deviation are automatically calculated and a score distribution graph printed. A student response matrix indicating each student's response to each question is produced as well as item and question analyses which can be used to evaluate the validity of the questions and the available answers.

EduTest allows a teacher to weight questions so that a correct

response to some questions is worth more than correct responses for others. One simple method of weighting is to give each "average" question a weight of 1, then a weight of 2 means a question is worth twice the points of an average question. EduTest permits a user to weight questions with any value and automatically calculates the ratio to a scale of 100.

At user option EduTest also allows grade specification according to numeric value and displays the resulting grade distribution. Grade assignment may be redone until the desired distribution curve is achieved. At the conclusion of the run, EduTest prints a list of students by grade achieved.

Teacher Responsibilities

The teacher writes a test of no more than 100 questions. Because a maximum of 25 responses can be recorded on a single answer card, a test in excess of 25 questions, must be divided into parts, i.e. a maximum of four parts. Each part except the last *must* contain 25 questions, e.g., an 80 question test is numbered:

Part 1—Questions 1-25
Part 2—Questions 26-50
Part 3—Questions 51-75
Part 4—Questions 76-80

To use EduTest, the teacher must also assign a unique two digit student number to each member of the class. Student number 00 is reserved for the teacher and is used to identify the key cards containing the correct answers.

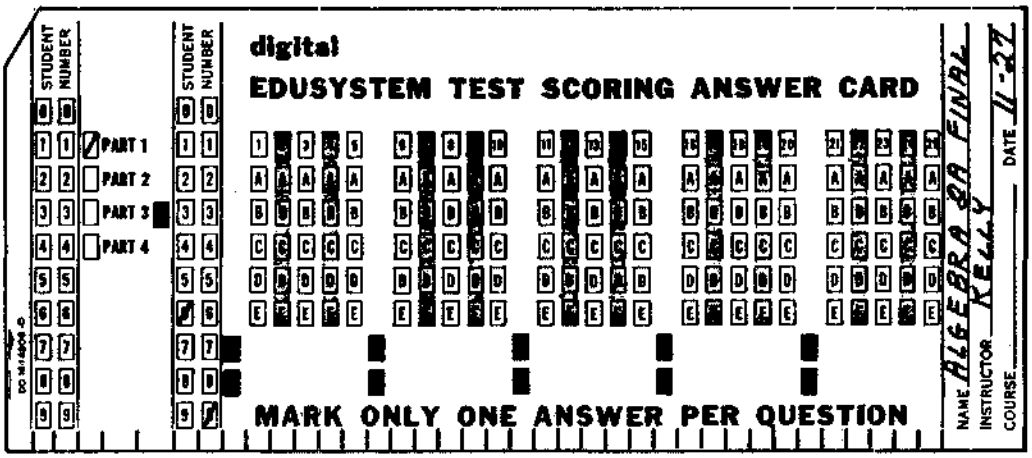
Key Card Preparation

Key cards are prepared to input the correct answers to EduTest. To prepare key cards for EduTest:

1. Leave the first Student Number columns blank.
2. Mark the part number corresponding to that part of the test for which the key is prepared.
3. In the second Student Number columns:
   - Place a mark through the EN in the word STUDENT.
   - Mark the 6 directly below this EN.
   - Mark the 9 in the next column.
4. Mark the correct answers with a firm vertical mark within the appropriate box.

5. Repeat steps 1 through 4 for the remaining parts of the test.

The following is a correctly marked key card for Part 1.

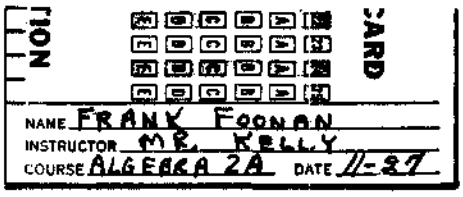


Student Answer Cards

Each student answer card may contain a maximum of 25 test question answers. Cards must be marked in accordance with the following rules:

1. Use a *Number 2 pencil*
2. Write name and class information on the right end of the card—STAY BELOW THE QUESTION 25 LINE

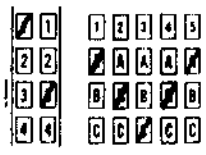
RIGHT



WRONG

3. To mark a box—make a firm vertical mark *within* the box.

RIGHT



WRONG

4. Mark Student Number in *both* Student Number columns.

5. Mark the appropriate part number box according to which part of the test is being answered:

Questions 1-25	Part 1
Questions 26-50	Part 2
Questions 51-75	Part 3
Questions 76-100	Part 4

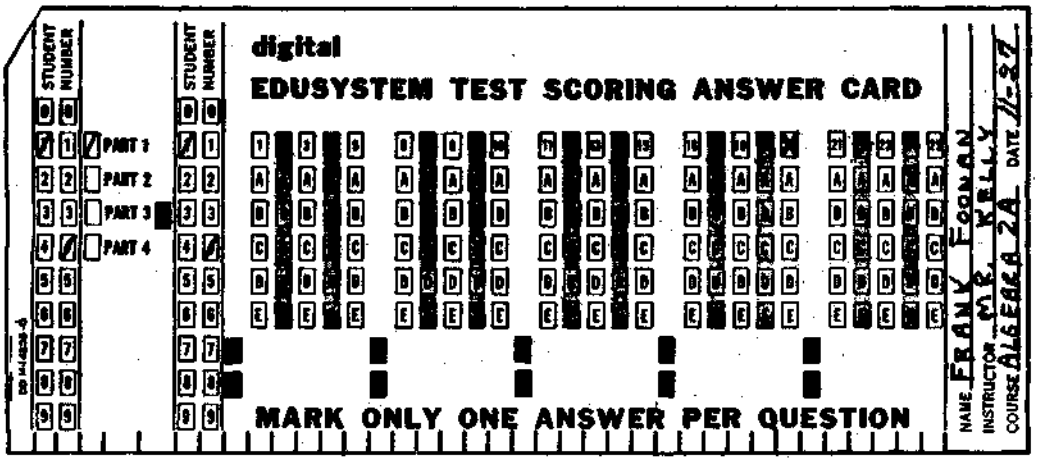
6. Answering the test:
   - Cross out all unanswered questions by marking the corresponding boxes.

NOTE

EduTest reads the answer cards in blocks of five questions. This technique necessitates crossing out any "left over" questions in a block. Giving an 18 question test means that the fourth block of answers (questions 16, 17, 18, 19, and 20) is marked only through question 18. Students must be instructed to cross out questions 19 and 20.

   - Mark only ONE answer for each question.
   - Erase *completely* to change the answer.
   - Recheck the answer card to make certain that no column contains two marks.
7. Repeat the above procedure for each part of the test.
8. Make no stray marks or doodling on the card.

A correctly marked card for an 18 question test is illustrated below:



OPERATING INSTRUCTIONS

Storing EduTest

The EduTest programs must be permanently stored on your EduSystem disc or DECtape before the EduTest System can be run.

The following procedure stores the programs for use and must be performed only once for each EduSystem.

1. Load EduSystem 30 according to the instructions in Chapter 7 for EduSystem 30 or Chapter 8 for EduSystem 40.
2. Type: NEW
   NEW FILE NAME—EDTEST
3. EduSystem 30 responds with READY.
   Read EDTEST paper tape from the reader by typing TAPE.
4. When reading is complete TTY responds with READY.
   Type: PRI
   Passford (password does not echo)
5. Type: SAVE
   (EDTEST will be permanently stored on disc or DECtape).
6. Repeat steps 2 through 5 for each of the following programs:

   EDTST2
   EDTST3
   EDTST4
   EDTST5
   EDTST6

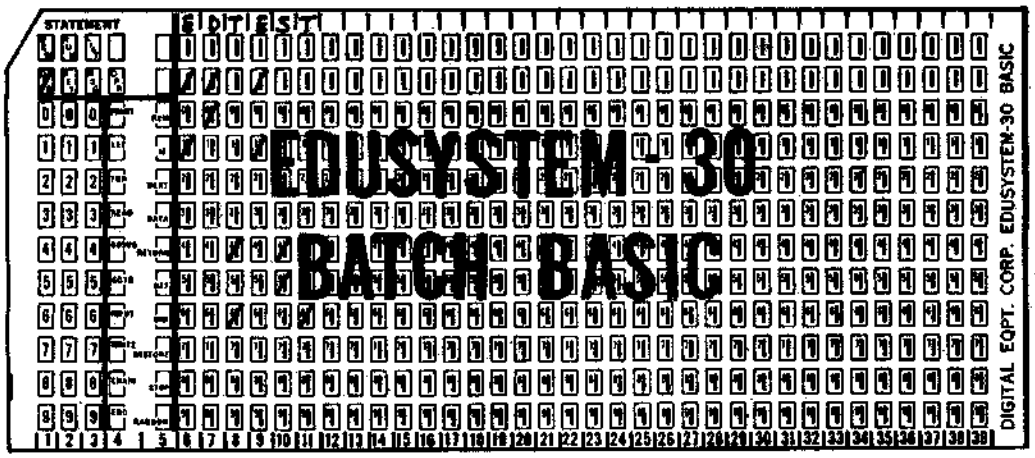
7. Continue with the instructions in the section on Operating EduTest.

Preparing EduTest Input

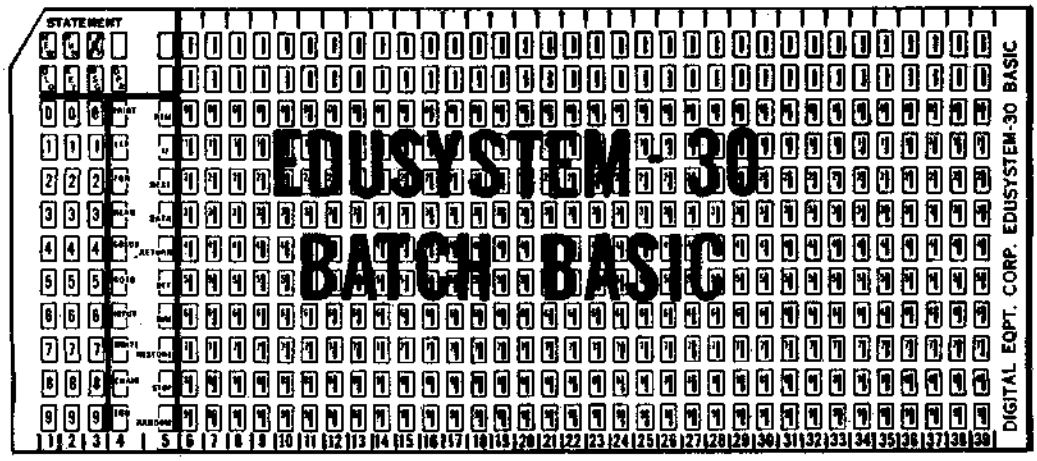
Three control cards are required to run EduTest. These cards contain the marks for OLD EDTEST, LIST, and RUN.

The OLD EDTEST control card is placed before the student answer cards; the answer cards are followed by the key cards, and the LIST and RUN control cards. The student answer cards may be in any sequence, i.e., they need *not* be in order by student number.

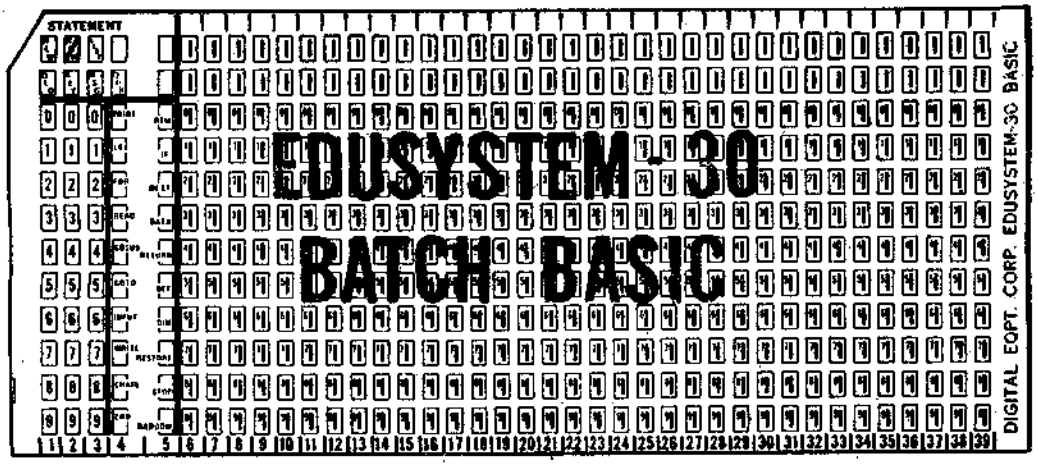
1. A card marked OLD EDTEST.



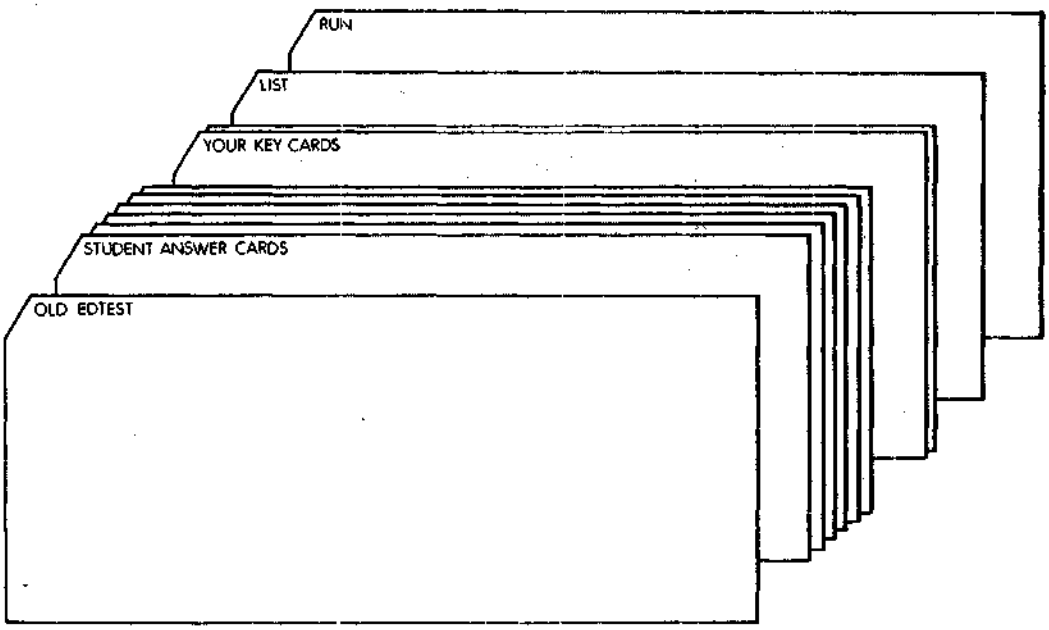
2. A card marked LIST.



3. A card marked RUN.



Arrange the control cards in the following order.



Operating EduTest

If the computer is not currently reading cards, do the following:

1. Place a blank card at the front of the desk of control cards, and place the answer cards and key cards deck in the card reader:
   - Face down
   - Top edges toward the front
2. Turn the reader on and press the READY (or START) button.
3. Turn the switch on the console Teletype to LINE.
4. Hold down the CTRL key and strike C.
5. Strike the RETURN key.
6. Type SCRATCH and strike the RETURN key.
7. The Teletype should respond with READY. (If not, repeat steps 3 through 6.)
8. Type BATCH and strike the RETURN key.

After the computer reads the deck of cards, it prints:

OLD EDTEST, and then
LIST

The computer then prints listing of the key cards and the student answer cards. (The program considers such input cards as data.) After all the DATA statements have been printed (Statement #292 in the sample on page 16 is the last one), type CTRL/C.

The computer types:

```
RUN
```

after reading the last control cards.

Console Messages and Responses

If EduTest detected errors or inconsistencies in the input data, error messages are printed as shown in Table D-1.

Modifying EduTest

As delivered, EduTest is dimensioned for 50 questions and 35 students. Using EduTest for more students and fewer questions requires that the user redimension EDTST2, EDTST3, EDTST4 and EDTST5. The DIM statement in each of these programs is line 140.

To redimension EduTest type your entry in Entry column, EduTest in Response column:

Entry	Response
OLD	OLD FILE NAME
EDTST2	READY
LISTNH140	140 DIM K(50),N(36),R(5),V(50)
SAVE	CTRL/C

Retype line 140 with the new subscripts, noting that N must be dimensioned 1 greater than the number of students, and save the redimensioned version. Repeat with EDTST3, etc.

Letting Q=# questions, and S=# students, the subscripted variables are:

```
EDTST2:  K(Q),N(S+1),R(5),V(Q)
EDTST3:  C(S),K(Q),R(5),T(S),V(Q)
EDTST4:  K(Q),A(25,6)
EDTST5:  E(Q),F(20),K(Q),T(S),V(Q)
```

Errors: SS Redimensioning not accomplished
GS,SP Not enough room—use smaller groups of students or delete the V routines.

EduSystem-30 utilizes only 4K words of core memory. This restricted program area limits EduTest to the following mixes of questions and students:

25 questions x 50+students
50 questions x 35 students
75 questions x less than 10 students

More space may be obtained by deleting the non-equal weighting routine, V(I) in all programs.

Table D-1. EduTest Error Messages

Message	Explanation	Reply/Action
HOW MANY QUESTIONS IN THE TEST?		Type total number of questions in test (e.g. 25, 80); strike RETURN key.
HOW MANY STUDENTS TOOK THE TEST?		Type number of students; strike RETURN key.
KEY CARD X HAS BAD ID (where X is test part number)	Key card for Part X is incorrectly coded in second student number zone or key cards not in sequence at end of deck.	Correct card or sequence and restart from Step 1, Operating EduTest.
BAD MARKS ON KEY CARD X (where X is test part number)	Key card for part X has at least one blank column where question answer is required.	Correct card and restart from Step 1, Operating EduTest.
STUDENT NUMBER X HAS MISMATCHED I.D. ON CARD Y (where X is student number and Y is test part number).	In a test requiring Y or more answer cards, student number X has not submitted enough answer cards or has incorrectly marked his student number on an answer card.	Answer N to the SHALL WE GO ON question. Locate student X's Part Y card and correct his marking of the second number zone or make up a Part Y card with student number and add it to the deck. (EduTest scores zero for that part). Restart at Step 1, Operating EduTest.

Message	Explanation	Reply/Action
STUDENT NUMBER X HAS INSUFFICIENT MARKS ON CARD Y.	Student number X has not marked all columns on card Y. Examine data output of student number X for blanks.	Answer N to SHALL WE GO ON question. Find answer card. If student did not answer questions, cross out corresponding question number box. If the column is marked, darken the mark with a pencil. Restart at Step 1, Operating EduTest.
NOT ENOUGH STUDENTS HERE—THERE ARE X STUDENTS. (Where X is total number of input students).	A warning message. Number of students actually processed is smaller than number entered from console at start of run.	If no students have been omitted, Answer Y to SHALL WE GO ON question. If students were omitted; find cards and restart at Step 1, Operating EduTest.
TOO MANY CARDS FOR X STUDENTS. (Where X is total number of input students).	Too many student answer cards for number of students specified.	Restart at Step 1, Operating EduTest and input correct number of students being processed.
END OF CARD CHECK—SHALL WE GO ON (Y OR N)?		If you have not received any error messages, answer Y. If you have received MISMATCHED I.D., INSUFFICIENT MARKS or TOO MANY CARDS messages, strike N and restart at Step 1, Operating EduTest after correcting cards in error.

Message	Explanation	Reply/Action
ARE THE QUESTIONS EQUALLY WEIGHTED (Y OR N)?		Y—If questions are of equal value e.g., 5 points for each of 20 questions. N—if you want to weight questions. Specify weight factor after each question number.
DO YOU WISH TO SPECIFY GRADES (Y OR N)?		Y—to specify cut-off levels for grades A to D. Type in numeric value for each grade level. N—terminate EduTest run.
DO YOU WISH TO RESPECIFY GRADES (Y OR N)?		Y—to enter new values for each grade level. N—terminate run. List of students organized by grade is typed.

EDUTEST PROGRAMS

The EduTest system is comprised of six programs:

NAME	FUNCTION
EDTEST	Reads answer cards and writes data onto disk or DECtape.
EDTST2	Checks answer cards for inconsistent I.D. numbers, missing marks, etc.
EDTST3	Produces the Student Response Matrix.
EDTST4	Produces the Item Analysis.
EDTST5	Produces the Question Analysis and Distribution of Scores.
EDTST6	Calculates the Statistical Measures, and Produces Grading, if desired.

All programs make extensive use of multiple statements per line. The separator between statements on the same line is SHIFT/L which prints as \ on the teletype, and is shown as ◊ on the sample listing. The following pages contain a listing of the BASIC programs that make up EduTest. Comments are limited to describing the function of sets of code.

Sample Output

The following copy is produced by the LIST card. Lines 1 and 2 are the key cards, lines 101 and 102 are student number 10's cards for parts 1 and 2, etc. Note that the second student number in each line appears as DATA, and that Edu-30 BASIC, interprets the marks as:

Question unanswered (crossed out) — 1
Answers: A — 2
B — 3
C — 4
D — 5
E — 6
Blank — no mark

Note also that each block of five questions is read as one five digit number.

```
EDTEST  EDU BASIC



1 DATA -9,44326,22425,53524,52262,43342
2 DATA -9,23456,54323,45654,32345,65432
101 DATA 10,44322,22324,53523,42262,43452
102 DATA 10,23356,54333,45654,32245,65432
111 DATA 11,44326,22424,53524,52242,43242
112 DATA 11,23456,54422,45654,32345,66432
121 DATA 12,44325,22425,53523,52264,43342
122 DATA 12,23466,53322,45634,32345,66433
2000 REM "EDTEST" FIRST PROGRAM IN EDUTEST
2040 DIM N(550)
2041 FOR I=1 TO 550\READ N(I)\IF N(I)=-2 THEN 2042\NEXT I
2042 FOR J=1 TO I-1\WRITE N(J)\NEXT J
2043 RESTORE\CHAIN "EDTST2"
2045 DATA -1,-1,-1,-1,-1,-1,-2
2046 END
```

```
EDTEST  EDU BASIC


HOW MANY QUESTIONS IN THE TEST?50
HOW MANY STUDENTS TOOK THE TEST?3
**********CHECKING ANSWER CARDS FOR MISSENTRIES.
END OF CARD CHECK - SHALL WE GO ON (Y OR N)?Y
```

your responses

```
ARE THE QUESTIONS EQUALLY WEIGHTED (Y OR N)?N
ENTER THE WEIGHT FACTOR FOR EACH QUESTION.

PART 1
QUESTION # 1 ?2
QUESTION # 2 ?2
QUESTION # 3 ?4
QUESTION # 4 ?4
QUESTION # 5 ?1
QUESTION # 6 ?2
QUESTION # 7 ?1
QUESTION # 8 ?2
QUESTION # 9 ?2
QUESTION # 10 ?1
QUESTION # 11 ?3
QUESTION # 12 ?3
QUESTION # 13 ?1
QUESTION # 14 ?3
QUESTION # 15 ?1
```

```
QUESTION # 16 ?1
QUESTION # 17 ?2
QUESTION # 18 ?4
QUESTION # 19 ?1
QUESTION # 20 ?3
QUESTION # 21 ?2
QUESTION # 22 ?2
QUESTION # 23 ?1
QUESTION # 24 ?1
QUESTION # 25 ?1

PART 2
QUESTION # 1 ?4
QUESTION # 2 ?3
QUESTION # 3 ?4
QUESTION # 4 ?2
QUESTION # 5 ?2
QUESTION # 6 ?2
QUESTION # 7 ?2
QUESTION # 8 ?3
QUESTION # 9 ?4
QUESTION # 10 ?2
QUESTION # 11 ?3
QUESTION # 12 ?3
QUESTION # 13 ?1
QUESTION # 14 ?1
QUESTION # 15 ?3
QUESTION # 16 ?3
QUESTION # 17 ?4
QUESTION # 18 ?2
QUESTION # 19 ?2
QUESTION # 20 ?2
QUESTION # 21 ?2
QUESTION # 22 ?3
QUESTION # 23 ?4
QUESTION # 24 ?2
QUESTION # 25 ?1
```

STUDENT RESPONSE MATRIX

.....CORRECT RESPONSES ARE INDICATED BY -
.....INCORRECT RESPONSES ARE LISTED
.....A NO-RESPONSE IS INDICATED BY *

STUDENT NUMBER	PART I QUESTION NUMBER																									NUMBER RIGHT	CUM % GRADE
	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5		
10	-	-	-	-	A	-	-	B	-	C	-	-	-	-	B	C	-	-	-	-	-	-	C	D	-	18	36.8
11	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-	-	-	-	C	-	-	-	A	-	-	22	41.2
12	-	-	-	-	D	-	-	-	-	-	-	-	-	-	-	B	-	-	-	-	C	-	-	-	-	22	39.5
KEY	C	C	B	A	E	A	A	C	A	D	D	B	D	A	C	D	A	A	E	A	C	B	B	C	A		

STUDENT NUMBER	PART 2 QUESTION NUMBER																									NUMBER RIGHT	CUM % GRADE
	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5		
10	-	-	B	-	-	-	-	-	B	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	22	84.2
11	-	-	-	-	-	-	-	C	-	A	-	-	-	-	-	-	-	-	-	-	-	E	-	-	-	22	90.4
12	-	-	-	E	-	-	B	-	-	A	-	-	-	B	-	-	-	-	-	-	-	E	-	-	B	19	86
KEY	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A		

.....

ITEM ANALYSIS

.....NUMBERS SHOWN ARE TO THE NEAREST PERCENT
.....CORRECT CHOICES ARE INDICATED BY *

PART 1

QUESTION NUMBER	ANSWER A(%)	ANSWER B(%)	ANSWER C(%)	ANSWER D(%)	ANSWER E(%)	NO RESPONSE
1	0	0	100 *	0	0	0
2	0	0	100 *	0	0	0
3	0	100 *	0	0	0	0
4	100 *	0	0	0	0	0
5	33	0	0	33	33 *	0
6	100 *	0	0	0	0	0
7	100 *	0	0	0	0	0
8	0	33	67 *	0	0	0
9	100 *	0	0	0	0	0
10	0	0	67	33 *	0	0
11	0	0	0	100 *	0	0
12	0	100 *	0	0	0	0
13	0	0	0	100 *	0	0
14	100 *	0	0	0	0	0
15	0	67	33 *	0	0	0
16	0	0	33	67 *	0	0
17	100 *	0	0	0	0	0
18	100 *	0	0	0	0	0
19	0	0	33	0	67 *	0
20	67 *	0	33	0	0	0
21	0	0	100 *	0	0	0
22	0	100 *	0	0	0	0
23	33	33 *	33	0	0	0
24	0	0	67 *	33	0	0
25	100 *	0	0	0	0	0

PART 2

QUESTION NUMBER	ANSWER A(%)	ANSWER B(%)	ANSWER C(%)	ANSWER D(%)	ANSWER E(%)	NO RESPONSE
1	100 *	0	0	0	0	0
2	0	100 *	0	0	0	0
3	0	33	67 *	0	0	0
4	0	0	0	67 *	33	0
5	0	0	0	0	100 *	0
6	0	0	0	100 *	0	0
7	0	33	67 *	0	0	0
8	0	67 *	33	0	0	0
9	67 *	33	0	0	0	0
10	67	33 *	0	0	0	0
11	0	0	100 *	0	0	0
12	0	0	0	100 *	0	0
13	0	0	0	0	100 *	0
14	0	33	0	67 *	0	0
15	0	0	100 *	0	0	0
16	0	100 *	0	0	0	0
17	100 *	0	0	0	0	0
18	33	67 *	0	0	0	0
19	0	0	100 *	0	0	0
20	0	0	0	100 *	0	0
21	0	0	0	0	100 *	0
22	0	0	0	33 *	67	0
23	0	0	100 *	0	0	0
24	0	100 *	0	0	0	0
25	67 *	33	0	0	0	0

QUESTION ANALYSIS

QUESTION	VALUE	PERCENT CORRECT	TIMES MISSED	
PART 1				
1	1.75	100	0	I
2	1.75	100	0	I
3	3.51	100	0	I
4	3.51	100	0	I
5	.88	33.3	2	IXX
6	1.75	100	0	I
7	.88	100	0	I
8	1.75	66.7	1	IX
9	1.75	100	0	I
10	.88	33.3	2	IXX
11	2.63	100	0	I
12	2.63	100	0	I
13	.88	100	0	I
14	2.63	100	0	I
15	.88	33.3	2	IXX
16	.88	66.7	1	IX
17	1.75	100	0	I
18	3.51	100	0	I
19	.88	66.7	1	IX
20	2.63	66.7	1	IX
21	1.75	100	0	I
22	1.75	100	0	I
23	.88	33.3	2	IXX
24	.88	66.7	1	IX
25	.88	100	0	I
PART 2		CORRECT		
1	3.51	100	0	I
2	2.63	100	0	I
3	3.51	66.7	1	IX
4	1.75	66.7	1	IX
5	1.75	100	0	I
6	1.75	100	0	I
7	1.75	66.7	1	IX
8	2.63	66.7	1	IX
9	3.51	66.7	1	IX
10	1.75	33.3	2	IXX
11	2.63	100	0	I
12	2.63	100	0	I
13	.88	100	0	I
14	.88	66.7	1	IX
15	2.63	100	0	I
16	2.63	100	0	I
17	3.51	100	0	I
18	1.75	66.7	1	IX
19	1.75	100	0	I
20	1.75	100	0	I
21	1.75	100	0	I
22	2.63	33.3	2	IXX
23	3.51	100	0	I
24	1.75	100	0	I
25	.88	66.7	1	IX

Note the actual decimal value assigned to each response as a result of the individualized weighting factors.

This chart gives you the quickest look at what the most difficult questions were.

DISTRIBUTION OF SCORES

RANGE	NUMBER	
100	0	I
95 TO 99.99	0	I
90 TO 94.99	1	IX
85 TO 89.99	1	IX
80 TO 84.99	1	IX
75 TO 79.99	0	I
70 TO 74.99	0	I
65 TO 69.99	0	I
60 TO 64.99	0	I
55 TO 59.99	0	I
50 TO 54.99	0	I
45 TO 49.99	0	I
40 TO 44.99	0	I
35 TO 39.99	0	I
30 TO 34.99	0	I
25 TO 29.99	0	I
20 TO 24.99	0	I
15 TO 19.99	0	I
10 TO 14.99	0	I
5 TO 9.99	0	I
0 TO 4.99	0	I

STATISTICAL MEASURES

.....THE MEAN (AVERAGE) SCORE IS 86.8419
.....THE MEDIAN (MIDDLE) SCORE IS 85.965
.....THE STANDARD DEVIATION IS 2.58237

DO YOU WISH TO SPECIFY GRADES (Y OR N)?Y

A FOR THOSE SCORES EQUAL TO OR GREATER THAN?95
B FOR REMAINING SCORES EQUAL TO OR GREATER THAN?85
C FOR REMAINING SCORES EQUAL TO OR GREATER THAN?75
D FOR REMAINING SCORES EQUAL TO OR GREATER THAN?0

GRADE	NUMBER	
A	0	I
B	2	IXX
C	1	IX
D	0	I
F	0	I

```
DO YOU WISH TO RESPECIFY THE GRADES (Y OR N)?Y

A FOR THOSE SCORES EQUAL TO OR GREATER THAN?85.965
B FOR REMAINING SCORES EQUAL TO OR GREATER THAN?80
C FOR REMAINING SCORES EQUAL TO OR GREATER THAN?60
D FOR REMAINING SCORES EQUAL TO OR GREATER THAN?30


GRADE          NUMBER

  A              1          IX

  B              2          IXX

  C              0          I

  D              0          I

  F              0          I


DO YOU WISH TO RESPECIFY THE GRADES (Y OR N)?N


A GRADES ARE:
STUDENT # 11

B GRADES ARE:
STUDENT # 10
STUDENT # 12

C GRADES ARE:

D GRADES ARE:

F GRADES ARE:



THIS IS THE END OF THE EDUTEST RUN.
THANK YOU.
```

The key cards and the student answer cards are imbedded in EDTEST as a mass of data statements. EDTEST is just the barest of programs necessary to read and write this data onto the disk (DECtape) and CHAIN to the next program, EDTST2. This technique allows EDTST2 to have much more 'working' program and to pull in data as it is required.

The 4K version of EduSystem-30 allows either 6,000 characters of program and no variables, 600 variables and no program, or some trade-off in between.

This version of EduTest allows you approximately:

```
2000 REM "EDTEST" FIRST PROGRAM IN EDUTEST
2010 REM COPYRIGHT 1972 DIGITAL EQUIPMENT CORPORATION
2020 REM MAYNARD, MASSACHUSETTS
2030 REM EDUSYSTEMS 30 AND 40
2040 DIM N(550)
2041 FOR I=1 TO 550\READ N(I)\IF N(I)=-2 THEN 2042\NEXT I
2042 FOR J=1 TO I-1\WRITE N(J)\NEXT J
2043 RESTORE\CHAIN "EDTST2"
2045 DATA -1,-1,-1,-1,-1,-1,-2
2046 END
```

25 questions x 50+ students
50 questions x 35 students
75 questions x less than 10 students.

The quickest way to significantly expand these limits is to delete the Unequal Weighting Option of EDTST2 and all handling of V(1) thereafter. If done, 100/Q is the value of each question.

```
100 REM "EDTST2" SECOND PROGRAM IN EDUTEST
110 REM COPYRIGHT 1972 DIGITAL EQUIPMENT CORPORATION
120 REM MAYNARD, MASSACHUSETTS
130 REM EDUSYSTEMS 30 AND 40
140 DIM K(50),N(36),R(5),V(50)
150 PRINT "HOW MANY QUESTIONS IN THE TEST";
160 INPUT Q
170 C=INT((Q-1)/25)+1
180 FOR I=1 TO C
190 READ N\IF N=-9 THEN 200\PRINT "KEY CARD";I;"HAS ,BAD I.D."\STOP
200 READ B1,B2,B3,B4,B5\NEXT I\RESTORE
210 FOR I=1 TO C\READ N
220 FOR J=1 TO 5\READ B\GOSUB 1000
230 FOR K=1 TO 5\K((I-1)*25+(J-1)*5+K)=R(K)
240 IF (I-1)*25+(J-1)*5+K=Q THEN 270
250 IF R(K)>=1 THEN 260\PRINT "BAD MARKS ON KEY CARD";I\STOP
260 NEXT K\NEXT J\NEXT I
270 FOR I=INT((Q-1)/5)+2 TO 5*C\READ B\NEXT I
280 PRINT "HOW MANY STUDENTS TOOK THE TEST";
290 INPUT S
300 PRINT "**********CHECKING ANSWER CARDS FOR MISSENTRIES."
310 C=INT((Q-1)/25)+1
320 FOR I=1 TO S
330 Z=0\M=0
340 READ N(I)
350 IF N(I)>=-1 THEN 380
360 PRINT "NOT ENOUGH STUDENTS HERE - THERE ARE";I-1;"STUDENTS"
370 S=I-1\GO TO 600
```

Q=# students
C=# cards/student

This set of code checks the key cards for blanks and sets K(1) to the correct key.

S=# students

Z=# marks counted,
M=# missing marks
end of file?

```
380 J=1
390 FOR K=1 TO 5
400 READ B
410 GOSUB 1000
420 NEXT K
430 IF Z>=Q THEN 560
440 GOSUB 960
450 FOR J=2 TO C
460 READ N2
470 IF N2=N(I) THEN 500
480 PRINT "**********STUDENT NUMBER";N(I);" HAS MISMATCHED I.D.";
490 PRINT " ON CARD";J
500 FOR K=1 TO 5
510 READ B\GOSUB 1000
520 NEXT K
530 IF Z>=Q THEN 560
540 GOSUB 960
550 NEXT J
560 NEXT I
570 READ N
580 IF N=-1 THEN 600
590 PRINT "TOO MANY CARDS FOR";S;"STUDENTS"
600 PRINT "END OF CARD CHECK - SHALL WE GO ON (Y OR N)?";\INPUT $A\PRINT
610 IF A=#Y THEN 620\IF A<>#N THEN 600\STOP
620 FOR I=1 TO 65\PRINT ".";\NEXT I\PRINT
630 PRINT
640 PRINT
650 PRINT\PRINT "ARE THE QUESTIONS EQUALLY WEIGHTED (Y OR N)?";
660 INPUT $A\PRINT
670 IF A=#Y THEN 840\IF A<>#N THEN 650
680 PRINT "ENTER THE WEIGHT FACTOR FOR EACH QUESTION."
690 FOR I=1 TO C#
```

Read a 5 digit answer block.

This loop is entered only when each student enters more than 1 card (>25 questions).

This set of code continues to check for missing marks, and also matches the I.D. numbers of cards 2, 3, etc. against the student's first card.

This is the code which allows unequal weighting and generates V_i, the decimal value of each question.

```
700 PRINT
710 PRINT "PART";I
720 FOR J=1 TO 25
730 PRINT "QUESTION #";J;
740 D=(I-1)*25+J
750 INPUT V(D)
760 T=T+V(D)
770 IF D=Q THEN 800
780 NEXT J
790 NEXT I
800 FOR I=1 TO Q
810 V(I)=V(I)*100/T
820 NEXT I
830 GO TO 870
840 FOR I=1 TO Q
850 V(I)=100/Q
860 NEXT I
870 PRINT
880 PRINT
890 RESTORE\FOR I=6*C*(S+1)+6 TO 1 STEP -3\FOR J=1 TO I/3
900 READ B1,B2,B3\NEXT J\WRITE B1,B2,B3\RESTORE\NEXT I\WRITE C,Q,S
910 FOR I=1 TO 6*C*(S+1)+6\READ B\NEXT I\FOR I=1 TO Q\WRITE K(I)\NEXT I
920 FOR I=1 TO Q\WRITE V(I)\NEXT I\FOR I=1 TO S\WRITE N(I)\NEXT I
930 PRINT
940 RESTORE
950 CHAIN "EDTST3"
960 IF Z+M=25*J THEN 990
970 PRINT "**********STUDENT NUMBER";N(I);" HAS INSUFFICIENT ";
980 PRINT "MARKS ON CARD";J\M=M+25*J-Z
990 RETURN
```

Subroutine to match total marks against questions.

```
1000 R(0)=0
1010 FOR X=1 TO 5
1020 B=B-R(X-1)*10^(6-X)
1030 R(X)=INT(B/10^(5-X))
1040 IF R(X)=0 THEN 1070
1050 Z=Z+1
1060 NEXT X
1070 RETURN
1080 END
```

This subroutine takes the 5 digit answer block and breaks it into five separate digits: R_1, R_2, R_3, R_4 and R_5. It also checks for a zero digit (no mark) and increments Z for each good mark.

Lines 890 and 900 shift the entire data file up 3 spaces, and insert C, Q and S at the front. Lines 910 and 920 write the new data at the end of the data file.

If V_i routines are deleted to gain space, each routine which reads the data file must be changed to reflect the absence in EduTest 2, 3, 4, and 5.

```
100 REM "EDTST3" THIRD PROGRAM IN EDUTEST
110 REM COPYRIGHT 1972 DIGITAL EQUIPMENT CORPORATION
120 REM MAYNARD, MASSACHUSETTS
130 REM EDUSYSTEMS 30 AND 40
140 DIM C(35),K(50),R(5),T(35),V(50)
150 FOR I=1 TO 65\PRINT ",";\NEXT I\PRINT
160 READ C,Q,S
170 FOR I=1 TO 6*C*(S+1)+6\READ B\NEXT I
180 FOR I=1 TO Q\READ K(I)\NEXT I\FOR I=1 TO Q\READ V(I)\NEXT I
190 RESTORE\FOR I=1 TO 6*C+3\READ B\NEXT I
200 PRINT\PRINT
210 PRINT TAB(20);"STUDENT RESPONSE MATRIX"
220 PRINT
230 PRINT ".........CORRECT RESPONSES ARE INDICATED BY ="
240 PRINT ".........INCORRECT RESPONSES ARE LISTED"
250 PRINT ".........A NO-RESPONSE IS INDICATED BY *"
260 FOR I=1 TO C
270 PRINT
280 PRINT TAB(26);"PART";I
290 PRINT "STUDENT";TAB(25);"QUESTION";TAB(58);"NUMBER";TAB(66);
300 PRINT "CUM %"\PRINT "NUMBER";TAB(26);"NUMBER";TAB(58);"RIGHT";
310 PRINT TAB(66);"GRADE"\PRINT TAB(8);
320 FOR J=1 TO 25\PRINT CHR$(INT(J/10)+48);" ";\NEXT J
330 PRINT\PRINT TAB(8);
340 FOR J=1 TO 25\PRINT CHR$(J-INT(J/10)*10+48);" ";\NEXT J
350 PRINT
360 PRINT
370 FOR J=1 TO S
```

Dummy read to the teacher's key.

Prints
```
000    2
123....5
```

```
380 C(J)=0
390 READ N
400 PRINT N;TAB(7);
410 FOR K=1 TO 5
420 READ B
430 GOSUB 790
440 FOR L=1 TO 5
450 D=(I-1)*25+(K-1)*5+L
460 IF R(L)=K(D) THEN 490
470 IF R(L)=1 THEN 480\PRINT " ";CHR$(R(L)+63);\GO TO 510
480 PRINT " *";\GO TO 510
490 PRINT " -";
500 C(J)=C(J)+1\T(J)=T(J)+V(D)
510 IF D=W THEN 530
520 NEXT L\NEXT K
530 FOR M=K+1 TO 5\READ B\NEXT M
540 T=INT(T(J)*10+.5)/10
550 N=INT(LOG(C(J)+.5)/LOG(10))+1\P=INT(LOG(T+.5)/LOG(10))+1
555 IF C(J)<>0 THEN 560\N=N+1\P=P+1
560 PRINT TAB(61-N);C(J);TAB(68-P);T
570 FOR M=2 TO C
580 READ N,B1,B2,B3,B4,B5
590 IF N<>-1 THEN 610
600 IF I=C THEN 720
610 NEXT M
620 NEXT J
630 RESTORE\FOR P=1 TO 6*C+3\READ B\NEXT P
640 FOR M=1 TO I
650 READ N,B1,B2,B3,B4,B5
660 NEXT M
670 PRINT "KEY";TAB(8);
680 FOR J=1 TO 25\L=(I-1)*25+J\PRINT CHR$(K(L)+63);" ";
```

This set of code prints the student response matrix.

How many digits?

Dummy read to the next student's card.

Rewind the data file and get ready for the next part.

```
690 IF L=Q THEN 700\NEXT J\PRINT
700 NEXT I
710 GO TO 740
720 PRINT "KEY";TAB(8);
730 FOR J=(I-1)*25+1 TO Q\PRINT CHR$(K(J)+63);" ";\NEXT J\PRINT
740 PRINT\PRINT
750 RESTORE\FOR I=1 TO 6*C*(S+1)+2*Q+S+9\READ B\NEXT I
760 FOR I=1 TO S\WRITE T(I)\NEXT I
770 RESTORE
780 CHAIN "EDTST4"
790 R(0)=0
800 FOR X=1 TO 5
810 B=B-R(X-1)*10^(6-X)
820 R(X)=INT(B/10^(5-X))
830 NEXT X
840 RETURN
850 END
```

Dummy read to end of file.
Write new data.

Note that line 500 calculates each student's score. If deleting the V_i routines to gain room, replace the V(D) of line 500 with 100/Q.

```
100 REM "EDTST4" FOURTH PROGRAM IN EDUTEST
110 REM COPYRIGHT 1972 DIGITAL EQUIPMENT CORPORATION
120 REM MAYNARD, MASSACHUSETTS
130 REM EDUSYSTEMS 30 AND 40
140 DIM K(50),A(25,6)
150 FOR I=1 TO 65\PRINT ",";\NEXT I\PRINT\PRINT\PRINT
160 READ C,Q,S
170 FOR I=1 TO 6*C*(S+1)+6\READ B\NEXT I
180 FOR I=1 TO Q\READ K(I)\NEXT I\RESTORE
190 PRINT TAB(20);"ITEM ANALYSIS"\PRINT
200 PRINT ".........,NUMBERS SHOWN ARE TO THE NEAREST PERCENT"
210 PRINT ".........,CORRECT CHOICES ARE INDICATED BY *"
220 FOR I=1 TO C
230 FOR J=1 TO 25
240 FOR K=1 TO 6\A(J,K)=0\NEXT K
250 NEXT J
260 PRINT\PRINT\PRINT TAB(26);"PART";I
270 PRINT "QUESTION";
280 FOR J=1 TO 5\PRINT TAB(J*7+5);"ANSWER";\NEXT J
290 PRINT TAB(49);"NO"
300 PRINT "NUMBER";TAB(13);"A(%)";TAB(20);"B(%)";TAB(27);"C(%)";
310 PRINT TAB(34);"D(%)";TAB(41);"E(%)";TAB(46);"RESPONSE"
320 RESTORE\READ C,Q,S\FOR J=1 TO C\READ N,B1,B2,B3,B4,B5\NEXT J
330 FOR J=1 TO I-1\READ N,B1,B2,B3,B4,B5\NEXT J
340 FOR J=1 TO S\READ N
350 FOR K=1 TO 5
360 READ B\GOSUB 640
370 FOR L=1 TO 5\X=L+5*(K-1)
```

Zero the counters.

This set of code counts the number of times each part of each question is chosen.

```
380 FOR M=1 TO 5
390 IF R(L)<>M+1 THEN 400\A(X,M)=A(X,M)+1\GO TO 410
400 NEXT M\A(X,6)=A(X,6)+1
410 IF 25*(C-1)+X=Q THEN 450
420 NEXT L
430 NEXT K
440 GO TO 460
450 FOR L=K+1 TO 5\READ B\NEXT L
460 FOR L=1 TO C-1\READ N,B1,B2,B3,B4,B5\NEXT L
470 NEXT J
480 FOR J=1 TO 25
490 X=2\PRINT J;
500 FOR K=1 TO 6\Z=INT(A(J,K)/S*100+.5)\GOSUB 590
510 PRINT TAB((K+1)*7-D);Z;\IF K=6 THEN 520\GOSUB 620\NEXT K
520 IF K(25*(I-1)+J)<2 THEN 530\PRINT\GO TO 540
530 PRINT "*"
540 IF 25*(I-1)+J=Q THEN 580
550 NEXT J
560 RESTORE\FOR J=1 TO 6*C+3\READ B\NEXT J
570 NEXT I
580 PRINT\RESTORE\CHAIN "EDT$T5"
590 IF Z<>0 THEN 600\D=1\GO TO 610
600 D=INT(LOG(Z)/LOG(10))+1
610 RETURN
620 IF X<>K(25*(I-1)+J) THEN 630\PRINT "*";
630 X=X+1\RETURN
640 R(0)=0
650 FOR Y=1 TO 5
```

Here these counts are converted to percentage points and printed.

How many digits? Subroutine.

Is this the correct answer? Subroutine.

```
660 B=B-R(Y-1)*10^(6-Y)
670 R(Y)=INT(B/10^(5-Y))
680 NEXT Y
690 RETURN
700 END
```

The A(25,6) dimensioned in line 140 is used to count the number of times each of the six possible marks (crossed out, A, B, C, D, E) is chosen for each of the twenty five questions in each part of the test. Line 500 is where the complete count is converted to a percentage point prior to printing.

```
100 REM "EDTST5" FIFTH PROGRAM IN EDUTEST
110 REM COPYRIGHT 1972 DIGITAL EQUIPMENT CORPORATION
120 REM MAYNARD, MASSACHUSETTS
130 REM EDUSYSTEMS 30 AND 40
140 DIM E(50),F(20),K(50),T(35),V(50)
150 FOR I=1 TO 65\PRINT ",";\NEXT I\PRINT
160 READ C,Q,S
170 FOR I=1 TO 6*C*(S+1)+6\READ B\NEXT I
180 FOR I=1 TO Q\READ K(I)\NEXT I
190 FOR I=1 TO Q\READ V(I)\NEXT I
200 FOR I=1 TO S\READ B\NEXT I
210 FOR I=1 TO S\READ T(I)\NEXT I\RESTORE
220 PRINT TAB(23);"QUESTION ANALYSIS"\PRINT
230 PRINT "QUESTION";TAB(13);"VALUE";TAB(20);"PERCENT";
240 PRINT TAB(30);"T I M E S  M I S S E D"
250 FOR I=1 TO 6*C+3\READ B\NEXT I
260 FOR I=1 TO S\FOR J=1 TO C\READ N
270 FOR K=1 TO 5\READ B\GOSUB 700
280 FOR L=1 TO 5\D=(J-1)*25+(K-1)*5+L
290 IF R(L)=K(D) THEN 310\E(D)=E(D)+1
300 IF D=Q THEN 320
310 NEXT L\NEXT K
320 FOR M=K+1 TO 5\READ B\NEXT M
330 NEXT J\NEXT I
340 FOR I=1 TO C
350 PRINT "PART";I;TAB(20);"CORRECT"
```

Count the number of times each question was missed. E_i=# misses on Ith question.

```
360 FOR J=1 TO 25
370 D=(I-1)*25+J
380 PRINT J;TAB(13);INT((V(D)+.005)*100)/100;TAB(20);
390 PRINT INT(((S-E(D))/S+.0005)*1000)/10;TAB(30);E(D);
400 PRINT TAB(35);"I";
410 IF E(D)=0 THEN 450
420 FOR K=1 TO E(D)
430 PRINT "X";
440 NEXT K
450 PRINT
460 IF D=Q THEN 490
470 NEXT J
480 NEXT I
490 PRINT\PRINT
500 FOR I=1 TO 65\PRINT ".";\NEXT I\PRINT\PRINT
510 PRINT TAB(21);"DISTRIBUTION OF SCORES"
520 FOR I=1 TO S
530 J=INT((T(I)+.005)/5)
540 F(J)=F(J)+1
550 T=T+T(I)
560 NEXT I
570 PRINT
580 PRINT "RANGE","NUMBER"
590 PRINT 100,F(20),"I";
600 FOR I=1 TO F(20)\PRINT "X";\NEXT I\PRINT
610 FOR I=95 TO 0 STEP -5
620 PRINT I;"TO";I+4.99,F(I/5),"I";
630 FOR J=1 TO F(I/5)\PRINT "X";\NEXT J\PRINT\NEXT I
640 RESTORE\WRITE S,T\FOR I=1 TO S\WRITE T(I)\NEXT I
650 FOR I=1 TO 6*C*(S+1)+2*Q+7-S\READ B\NEXT I
660 FOR I=1 TO S\READ T(I)\NEXT I\RESTORE
670 FOR I=1 TO S+2\READ B\NEXT I
```

Print question number, value, percent correct, times missed, and bar chart of times missed.

Set F(0)=# scores between 0 and 4.99
F(1)=# scores between 5 and 9.99, etc.
T=total of all scores

Print the bar chart for the distribution of scores.

```
680 FOR I=1 TO S\WRITE T(I)\NEXT I\RESTORE
690 CHAIN "EDTST6"
700 R(0)=0
710 FOR Y=1 TO 5
720 B=B-R(Y-1)*10^(6-Y)
730 R(Y)=INT(B/10^(5-Y))
740 NEXT Y
750 RETURN
760 END
```


EDTST6

```
100 REM "EDTST6" SIXTH PROGRAM IN EDUTEST
110 REM COPYRIGHT 1972 DIGITAL EQUIPMENT CORPORATION
120 REM MAYNARD, MASSACHUSETTS
130 REM EDUSYSTEMS 30 AND 40
140 DIM N(60),T(60),Z(60)
150 FOR I=1 TO 65\PRINT ".";\NEXT I\PRINT\PRINT
160 READ S\READ T
170 FOR I=1 TO S\READ T(I)\NEXT I
180 FOR I=1 TO S\READ N(I)\NEXT I
```

```
190 PRINT
200 PRINT TAB(24);"STATISTICAL MEASURES"
210 PRINT
220 M=T/S
230 PRINT ".....THE MEAN (AVERAGE) SCORE IS";M
240 FOR I=1 TO S\Z(I)=T(I)\NEXT I
250 FOR I=1 TO S\FOR J=I TO S
260 IF Z(I)<=Z(J) THEN 280
270 D=Z(I)\Z(I)=Z(J)\Z(J)=D
280 NEXT J\NEXT I
290 IF INT(S/2)=S/2 THEN 310
300 M2=Z((S+1)/2+1)\GO TO 320
310 M2=(Z(S/2)+Z(S/2+1))/2
320 M2=INT((M2+.0005)*1000)/1000
330 PRINT ".....THE MEDIAN (MIDDLE) SCORE IS";M2
340 FOR I=1 TO S
350 V=V+(M-T(I))^2
360 NEXT I
370 V=V/S
380 PRINT ".....THE STANDARD DEVIATION IS";SQR(V)
390 PRINT\PRINT
400 FOR I=1 TO 65
410 PRINT ".";
420 NEXT I
430 PRINT\PRINT\PRINT
440 PRINT "DO YOU WISH TO SPECIFY GRADES (Y OR N)?";
450 INPUT $A\PRINT
460 IF A=$Y THEN 490\IF A<>$N THEN 440
470 PRINT\PRINT "THIS IS THE END OF THE EDUTEST RUN."
480 PRINT "THANK YOU."\STOP
490 PRINT\PRINT "A FOR THOSE SCORES EQUAL TO OR GREATER THAN";
500 INPUT G(1)\FOR G1=2 TO 4
```

This set of code sets the dummy $Z_i=T_i$. Sorts the Z_i into ascending order and renumbers the subscript i accordingly.

The median score is then the middle score if S is odd, or halfway between the two innermost scores if S is even.

V=variance,
i.e. $\sigma^2=\sum_{x_S} \frac{(u-x)^2}{S}$

Begin letter grading

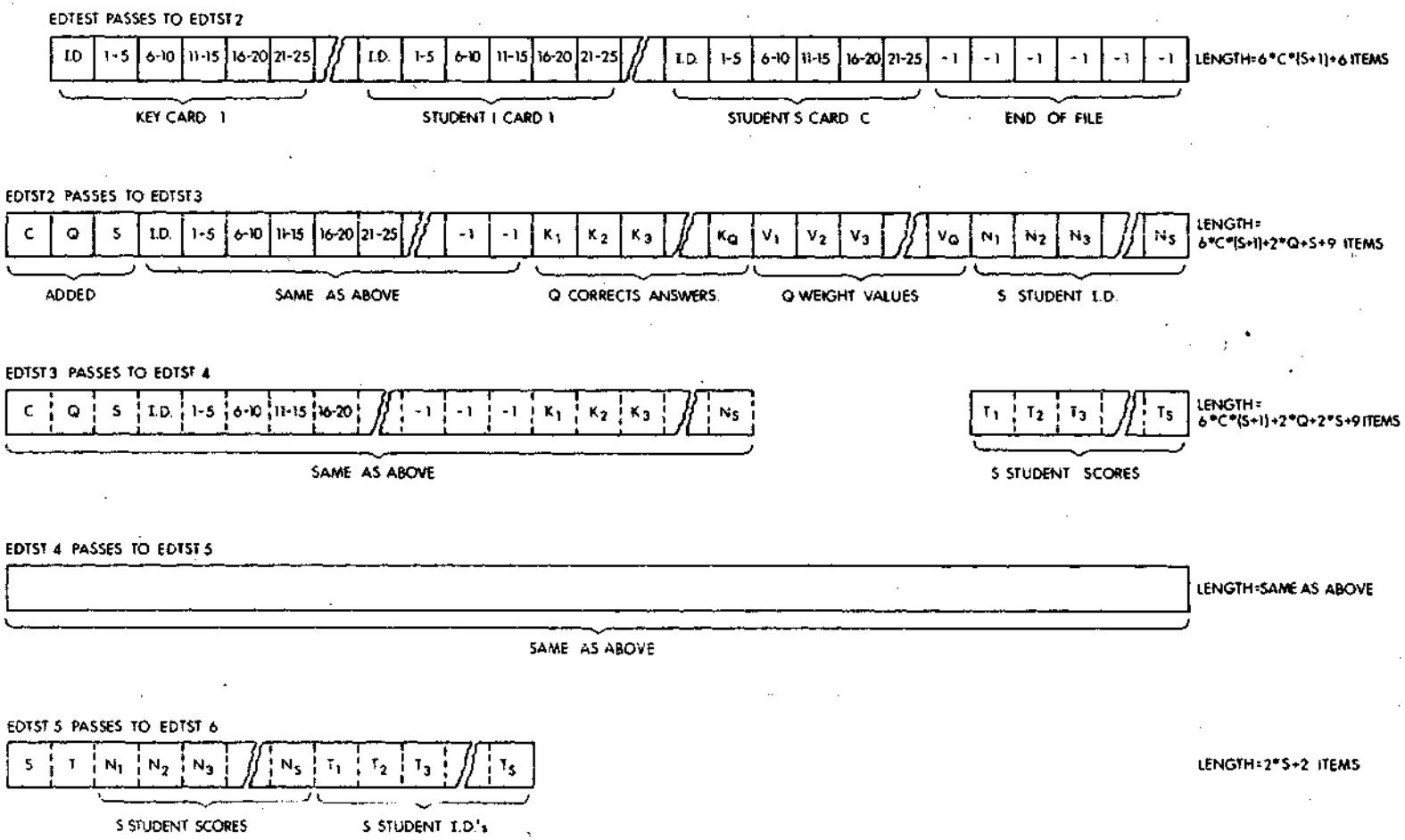
```
510 PRINT CHR$(G1+64);" FOR REMAINING SCORES EQUAL TO OR GREATER THAN";
520 INPUT G(G1)\NEXT G1\LET G(5)=0
530 PRINT\PRINT\FOR I=1 TO 5\F(I)=0\NEXT I
540 FOR I=1 TO S
550 FOR G1=1 TO 5
560 IF T(I)<G(G1) THEN 580
570 F(G1)=F(G1)+1\GO TO 590
580 NEXT G1
590 NEXT I
600 PRINT\PRINT "GRADE","NUMBER"\PRINT
610 FOR I=1 TO 5
620 G1=64+I\IF I<5 THEN 630\G1=70
630 PRINT "   ";CHR$(G1)," ";F(I),"I";
640 FOR J=1 TO F(I)\PRINT "X";\NEXT J
650 PRINT\PRINT\NEXT I
660 PRINT\PRINT
670 PRINT "DO YOU WISH TO RESPECIFY THE GRADES (Y OR N)?";
680 INPUT $A\PRINT
690 IF A=*Y THEN 490\IF A<>*N THEN 670
700 PRINT\PRINT
710 FOR G1=1 TO 5
720 IF G1<5 THEN 730\PRINT "F";\GO TO 740
730 PRINT CHR$(G1+64);
740 PRINT " GRADES ARE:"
750 FOR I=1 TO S
760 IF T(I)<G(G1) THEN 790
770 PRINT "STUDENT #";N(I)
780 T(I)=-5
790 NEXT I\PRINT
800 NEXT G1
810 PRINT\PRINT\GO TO 470
820 END
```

Sets F(1)=# A grades
F(2)=# B grades
etc.

Prints bar chart for distribution of letter grades.

This set of code prints the I.D. numbers of those students who received an A grade, etc.

EduTest Data File Layout



index/glossary

D

E

F

M

T

digital

DIGITAL EQUIPMENT CORPORATION, Maynard, Massachusetts, Telephone: (617) 897-5111 • ARIZONA, Phoenix • CALIFORNIA, Sunnyvale, Santa Ana, Los Angeles, Oakland, San Diego and San Francisco (Mountain View) • COLORADO, Denver • CONNECTICUT, Meriden • DISTRICT OF COLUMBIA, Washington (Riverdale, Md.) • FLORIDA, Orlando • GEORGIA, Atlanta • ILLINOIS, Chicago • INDIANA, Indianapolis • LOUISIANA, New Orleans • MASSACHUSETTS, Cambridge and Waltham • MICHIGAN, Ann Arbor and Detroit (Southfield) • MINNESOTA, Minneapolis • MISSOURI, St. Louis • NEW JERSEY, Englewood, Metuchen, Parsippany and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Centereach (L.I.), Manhattan, Syracuse and Rochester • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland and Dayton • OKLAHOMA, Tulsa • OREGON, Portland • PENNSYLVANIA, Philadelphia and Pittsburgh • TENNESSEE, Knoxville • TEXAS, Dallas and Houston • UTAH, Salt Lake City • WASHINGTON, Seattle • WISCONSIN, Milwaukee • ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BRAZIL, Rio de Janeiro, São Paulo and Porto Alegre • CANADA, Calgary, Alberta; Vancouver, British Columbia; Ottawa and Toronto, Ontario; and Montreal, Quebec • CHILE, Santiago • DENMARK, Copenhagen • FRANCE, Grenoble and Paris • GERMANY, Cologne, Hannover, Frankfurt, Munich and Stuttgart • INDIA, Bombay • ITALY, Milan • JAPAN, Tokyo • MEXICO, Mexico City • NETHERLANDS, The Hague • NEW ZEALAND, Auckland • NORWAY, Oslo • PHILIPPINES, Manila • PUERTO RICO, Miramar • SPAIN, Barcelona and Madrid • SWEDEN, Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Edinburgh, London, Manchester and Reading • VENEZUELA, Caracas

083 • 00173 • 2306
H-09-25

Printed in U.S.A.