PROBLEM-SOLVING ON THE TRS-80™ POCKET COMPUTER

A SELF-TEACHING GUIDE

DON INMAN
JIM CONLAN

More than a million people have learned to program, use, and enjoy microcomputers with Wiley paperback guides. Look for them all at your favorite bookshop or computer store:

BASIC, 2nd ed., Albrecht, Finkel, & Brown
BASIC for Home Computers, Albrecht, Finkel, & Brown
TRS-80 BASIC, Albrecht, Inman, & Zamora
More TRS-80 BASIC, Inman, Zamora, & Albrecht
ATARI BASIC, Albrecht, Finkel, & Brown
Data File Programming in BASIC, Finkel & Brown
Data File Programming for the Apple Computer, Finkel & Brown
ATARI Sound & Graphics, Moore, Lower, & Albrecht
Using CP/M, Fernandez & Ashley
Introduction to 8080/8085 Assembly Language Programming, Fernandez & Ashley
8080/Z80 Assembly Language, Miller
Personal Computing, McGlynn
Why Do You Need a Personal Computer? Leventhal & Stafford
Problem-Solving on the TRS-80 Pocket Computer, Inman & Conlan
Using Programmable Calculators for Business, Hohenstein
How to Buy the Right Small Business Computer System, Smolin
The TRS-80 Means Business, Lewis
ANS COBOL, 2nd ed., Ashley
Structured COBOL, Ashley
FORTRAN IV, 2nd ed., Friedmann, Greenberg, & Hoffberg
Job Control Language, Ashley & Fernandez
Background Math for a Computer World, 2nd ed., Ashley
Flowcharting, Stern
Introduction to Data Professing, 2nd ed., Harris

PROBLEM-SOLVING ON THE TRS-80 POCKET COMPUTER®

DON INMAN

JIM CONLAN

Dymax Corporation
Menlo Park, California

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Library of Congress Number 81-10358

ISBN: 0 471 09270-3
ISBN: 0 471 86808-6 RS

Printed in the United States of America

82 83 10 9 8 7 6 5 4 3 2 1

Contents

TO THE READER

Microcomputers continue to appear in more and more places. As they become smaller, they become more portable and, hence, useful in more places. The Radio Shack TRS-80 Pocket Computer is the ultimate in portability. It can be tucked away in your pocket and used wherever you go. Due to its portability, new uses will be continually discovered for this versatile tool.

This book shows a wide variety of problems that are easily solved with the Pocket Computer. Although some problems may seem to be presented with tongue in cheek, don't let the light-hearted approach fool you. All demonstration programs are practical in some way and, hopefully, will coax you to further applications. The problems cover a wide range of difficulty, but hints are provided that may be either used or ignored as your needs demand.

We assume that you have some familiarity with computing and with BASIC language as used on other computers. If you have never encountered BASIC language, we suggest *TRS-80 BASIC;* Albrecht, Inman, and Zamora; John Wiley & Sons, Inc., 1980. (It is also sold at Radio Shack stores under the title *TRS-80 Level II BASIC*.)

Our book has been organized to meet two main objectives:

1. Teaching the Tool – The TRS-80 Pocket Computer BASIC language and the use of the Pocket Computer's keyboard dominate the early chapters in the book. Problems are introduced as demonstrations to facilitate this objective.
2. Teaching Problem Solving Techniques – The techniques of problem solving dominate the discussions in the later chapters. Here the computer is regarded as a precision tool that is used to solve problems.

After covering the first five chapters, you will be able to make full use of the Pocket Computer. You will then be ready to solve the numerous problems that follow. Many different types of problems demonstrating a wide variety of applications are given throughout the book. Hints that are provided for the solutions to the problems are often quite extensive. You may choose not to use them, but they are there if you need them.

Often, more than one solution is given for a problem. This is true to life. There are many ways to solve most problems, and we do not all solve our problems in the same way.

The TRS-80 Pocket Computer is the tool used in this book. It is a small, but sophisticated, tool. You will find that the more that you use it, the more useful it is to you. This book is designed to help you explore the Pocket Computer in depth so that you can use it successfully in solving problems that you encounter outside of this book.

CHAPTER ONE

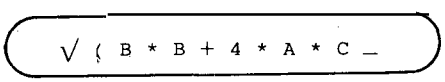
Pocket View of the Pocket Computer

The TRS-80 Pocket Computer is a unique mixture of a programmable calculator and a general-purpose computer. Although this book's main purpose is to illustrate procedures and techniques for solving problems with the Pocket Computer, some space must be allotted to a description of this unique problem-solving tool. Like any artisan, the problem solver is dependent on the tools that are available, and the ability to make use of fine tools is dependent on one's understanding of them.

This chapter is, therefore, devoted to the introduction of the capabilities of the TRS-80 Pocket Computer. In this chapter you will:

- become acquainted with the general characteristics of the Pocket Computer;
- learn how characters appear on the display;
- discover what kind of variables can be used in a program;
- learn a few BASIC commands and statements that are unique to the Pocket Computer or are used in unique ways;
- discover four different operating modes;
- learn how to use some keys in two different ways;
- learn something about error codes and how to clear them;
- discover PAUSE, a new form of the PRINT statement;
- learn all about the NEW statement;
- learn how to DEBUG programs;
- learn how to check for the amount of unused memory; and
- learn how to continue an interrupted program.

The TRS-80 Pocket Computer can be used as a calculator, as described in the *TRS-80 Pocket Computer Software Manual* (Catalog Number 26–3501) that comes with the computer. There are many wondrous features to be exploited in the calculator mode — you can even find the solution to an algebraic equation. For example, if you have previously entered a value for the variables A, B, and C, the Pocket Computer is capable of solving the square root of the following expression.



Each of the symbols shown in the example is entered from the keyboard of the Pocket Computer. When you press the ENTER key, the result is displayed.

Description of the Pocket Computer

Imagine carrying around the power of a BASIC-speaking computer in your pocket. No longer are you tied to a computer room or office desk — or even an electric company. Just tuck the computer into your shirt pocket and take it with you wherever a problem exists.

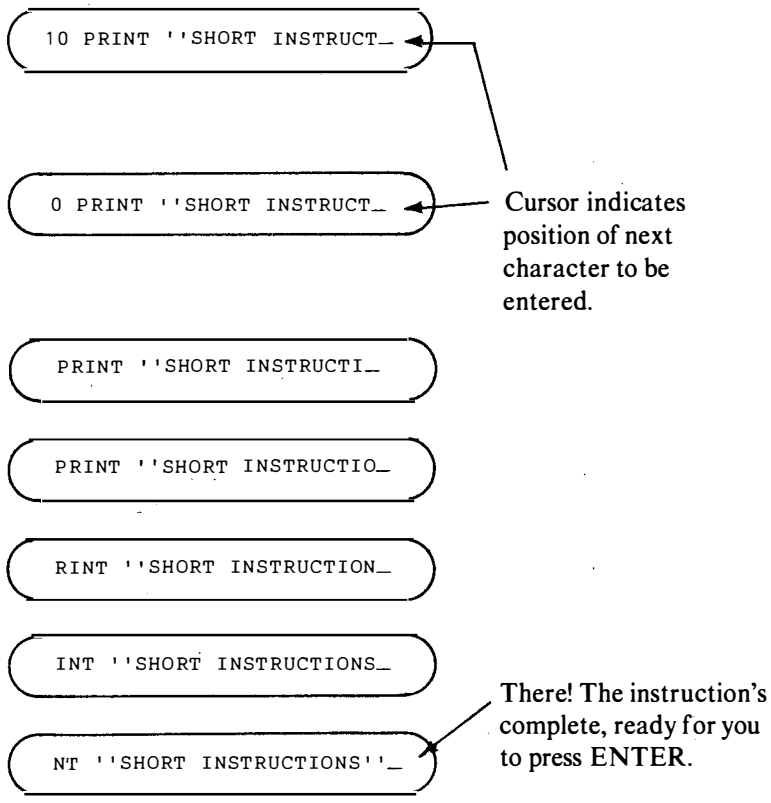
> Note: We didn't say tuck the computer into your *back* pants pocket. The liquid crystal display (LCD) is made of glass. If you sit down with it in your back pocket, you may crunch the LCD.

As long as the Liquid Crystal Display has been mentioned, we might as well talk about it first. If you're used to a 64-character, 16-line display of the TRS-80 Model I or Model III (or something similar), the Pocket Computer's 24-character, 1-line display is going to take a little adjusting to. However, within a few minutes, you'll find yourself writing shorter instructions. You may even find out that your programs become easier to read and understand because of the limit of the display size.

```
10 PRINT ''SHORT INSTRU_
```

The cursor is shown at the 24th position. What happens when the next character is typed?

Although only 24 characters are displayed, a BASIC line may contain a maximum of 80 characters. Computer buffs would say, "The size of the input buffer is 80 characters." Let's not get so technical. What they mean is that you can only input 80 characters on one program line. The computer will hold all of them even though you can't see them all at the same time on the display. If you continue to type the previous line, you will see the letters move to the left on the display and off the other end.



Yes, the TRS-80 Pocket Computer has an ENTER key that works just like the ENTER key on other TRS-80 computers (or like the RETURN key on some other computers). The Pocket Computer ENTER key looks like this:

The keys of the Pocket Computer are small and closely spaced, but they have a good feel and make clean contact. Numeric and arithmetic operation keys are in a separate area on the right side of the keyboard. Your entries may not go as fast as if you were touch-typing, but you'll soon get up to a respectable speed.

The computer can hold a maximum of 1,424 program steps. There are 26 fixed memories (used for variables) and 178 flexible memories (shared between program steps and variables). You'll learn all about the use of memory in Chapter 2. Calculations are carried out to 10-digit accuracy.

The computer also has editing functions, which include shifting the cursor from left or right, inserting characters, deleting characters, and shifting up or down one line at a time. We'll discuss these functions in Chapter 3.

Calculating capabilities include four arithmetic operations, power calculations, trigonometric and inverse trigonometric functions, logarithms and exponential functions, angular conversions, square roots, sign functions, absolute values, and integer and logic functions.

Power is supplied by mercury batteries. Memory content is *not* lost when the computer is turned off. Memory is protected (or saved) by the batteries when the rest of the computer is turned off.

As interface is available for a cassette recorder to save programs and data on tape and to retrieve them at a later time. There is also a 16-column printer that Radio Shack sells, which includes an interface for both a recorder and the printer.

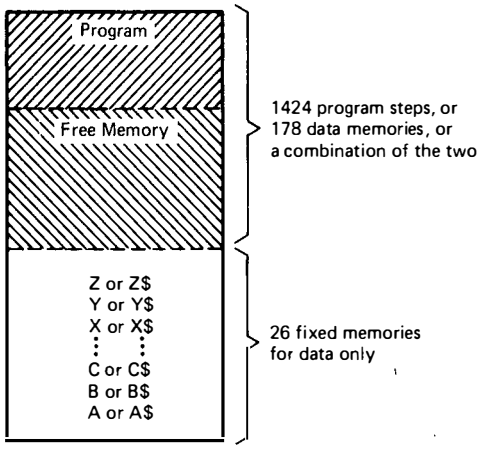
The computer itself is in a small package, 6⅞ inches long, 2¾ inches wide, and 19/32 inches thick. Its weight is just over one-third of a pound. A carrying case, two keyboard templates (more about them later), and a *User's Manual* are all included with the computer.

TRS-80 Pocket Computer BASIC

The BASIC language used by the Pocket Computer can recognize both numeric and string variables. A data memory is called a numeric variable when it stores a numeric value. Numeric variables use labels, or names, such as A, B, C, A(1), A(28), etc. A data memory is called a string variable when it contains a string of characters (limited to seven letters, numbers, or special symbols). String variables are distinguished by the $ symbol and are labeled as A$, B$, C$, A$(1), etc.

> CAUTION: There is a limit of 26 fixed memories used to hold variables. A (a numeric variable) and A$ (a string variable) both assign data to the same fixed memory. This must not be done at the same time. Only one value may be held in a given memory at a given time.

The Pocket Computer's memory is used in much the same way as that of a programmable calculator. Here is how the fixed and flexible memories may be assigned if no arrays are used. The use of memory for arrays is discussed in Chapter 2.



Any memory not used for program steps can be used for data memory, in addition to the 26 fixed memories. This may be done when using arrays. The number of data memories beyond the 26 fixed memories therefore depends upon how many steps are used in the program. The free memory in the previous diagram is used for additional program steps or data memories. In general, you may find out how much memory is not being used by typing in the MEM command in any of the Pocket Computer's operating modes. The display will then show how much data memory and how many program steps are unused. This is more thoroughly discussed in Chapter 2.

Basic Functions, Statements, Commands, and Tape Control Statements

Now let's take a look at the functions, statements, and commands available in Pocket Computer BASIC.

BASIC Functions

ABS	Absolute value
ACS	Arc cos
ASN	Arc sin
ATN	Arc tan
COS	Cosine
DEG	Degree/minute/second to decimal
DMS	Decimal to degree/minute/second
EXP	Exponential function
INT	Integer
LN	Natural logarithm
LOG	Common logarithm
SGN	Sign (positive or negative)
SIN	Sine
TAN	Tangent
√	Square root
π	Pi

BASIC Statements

AREAD	Contents displayed at start of definable program are read into the specified variable
BEEP	Beep sound is generated as many times as specified
CLEAR	Data memory clear
DEGREE	Degree is designated as angle measure
END	Indicates the end of a program
FOR	Beginning of FOR-NEXT loop
GOSUB	Execution is shifted to specified line or label where a subroutine begins
GOTO	Specified line or label is executed
GRAD	Grad is designated as angle measure
IF	Based on the specified condition, a branch is either taken or not taken
INPUT	Allows data to be input
LET	Assigns a value to a variable
NEXT	Ends a FOR-NEXT loop. Increments the step
PAUSE	Displays an output for approximately .85 seconds before going on
PRINT	Displays the specified contents and waits for ENTER key to be pressed
RADIAN	Radian is designated as angle measure
REM	Designates a non-executable statement
RETURN	Returns from subroutine
STEP	Increments a FOR-NEXT loop
STOP	Stops the execution of a program
THEN	Used only with IF statement as a jump instruction
USING	Format designation for display

BASIC Commands

CONT	Restart an interrupted program
DEBUG	Used to debug a program line by line
LIST	For listing a program in memory
MEM	Display amount of unused memory
NEW	Clear memory for a new program
RUN	Execute a program

Tape Control Statements

CHAIN	Program recorded on tape is read and executed
CLOAD	Transfer a program from tape to computer
CLOAD?	Check contents of program with those placed on tape
CSAVE	Record a program on tape
INPUT #	Transfer data from tape to memory of computer
PRINT #	Record data memory contents on tape

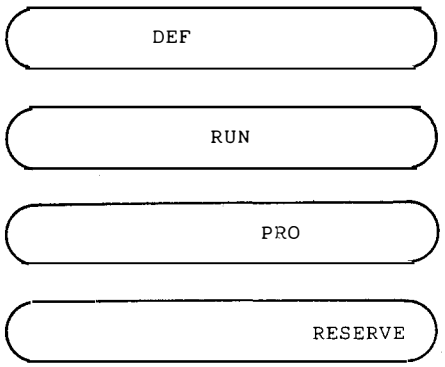
In writing this book, we assume that you have some knowledge of BASIC language. If you do not, we suggest you read *TRS-80 BASIC*; Albrecht, Inman, and Zamora; Wiley & Sons, Inc., 1980. As you look through the Pocket Computer's BASIC commands and statements, you will recognize many of them. We will not dwell on those statements and commands that are common to other computers. However, the Pocket Computer has some new ones, as well as some familiar ones that are used in different ways, which we *will* explain.

Modes of Operation

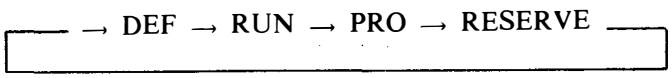
Before you are introduced to any more BASIC commands or statements, you should take a look at the four available operating modes of the Pocket Computer.

1) Program mode (PRO)
2) Run mode (RUN)
3) Definable mode (DEF)
4) Reserve program mode (RESERVE)

The mode currently being used is displayed at the top of the display.



A mode key, MODE, changes the operating mode from left to right. The mode moves clockwise, as indicated in the following diagram, each time the MODE key is pressed.



The operations carried on in each mode are briefly summarized as follows:

1) PRO – The program mode is used to write, enter, and edit programs.
2) RUN – The run mode is used to execute programs or for direct calculations.
3) DEF – The definable mode is used to execute programs that have been defined by a label.
4) RESERVE – The reserve program mode is used to write, enter, and edit programs or functions for reserved keys that will be frequently used.

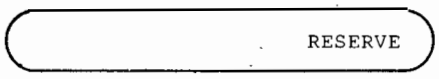
The use of BASIC commands and statements depends upon the operating mode selected. In this chapter, you'll find out how to use the PRO and RUN modes. In actual usage, the operating modes interact with each other and should not be considered isolated from each other. Let's investigate the PRO mode first.

The Program Mode

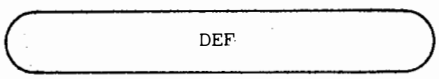
Access the program mode by pressing the MODE key a number of times (depending on the mode that you are presently in) until you see the PRO prompt on the top of the display. When you turn the Pocket Computer on, it is always in the mode in which it was last used.

Example:

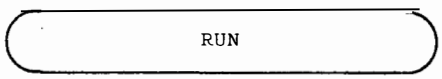
Suppose you last used the RESERVE mode, and you want the PRO mode now.



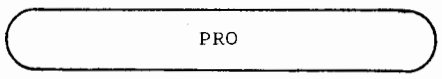
Press the MODE key once, and you see:



Press the MODE key again, and you see:



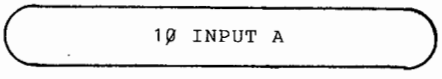
Press the MODE key once more, and you see:



You are now in the mode normally used to enter programs into the computer's memory. Lines of a program are numbered using integers ranging from 1 through 999. When a line is complete, it is entered into memory by pressing the ENTER key. One line may contain one or more statements. If more than one statement is used within a line, the statements must be separated by a colon.

Example:

One statement per line:



First line

```
20 B=A+1_
```

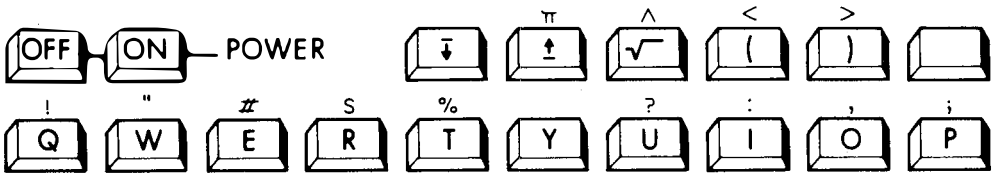
Second line

Or two statements per line:

```
10 INPUT A: B=A+1_
```

colon cursor

You may have trouble discovering how to display the colon. Notice that some of the top two rows of keys have symbols above them.



The colon is the symbol above the I key.

To enter the colon, you must first press the SHIFT key and then the I key. This selects the colon for entry. A shift designator appears on the display when the shift key is depressed: It tells the user that the next key stroke will be shifted.

SHIFT I

1st then next

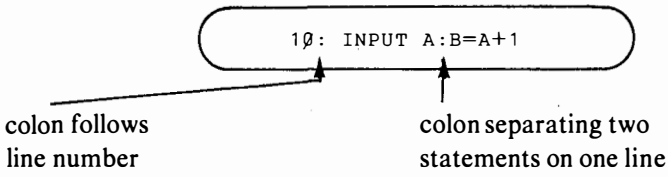
```
SHFT
```

When the shift key is pressed, this appears on the display.

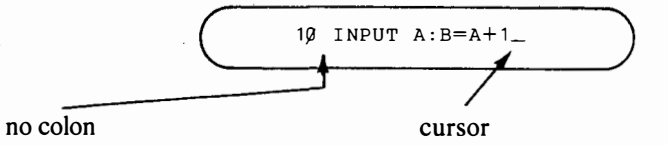
> If you should ever press the SHIFT key by mistake, press it a second time. This will cancel the affect of the first shift.

When the ENTER key has been pressed following the completion of a program line, the line is entered into memory, and the line is displayed once again with a colon following the line number.

> We will not always remind you to press the ENTER key. Just remember that it *must* be pressed at the end of each program line.

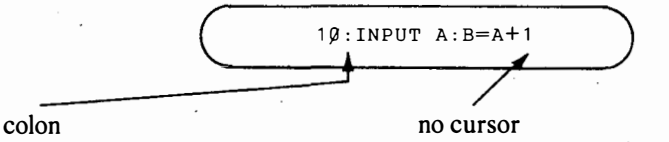


If you should ever forget whether you have pressed the ENTER key or not, *look* at the display. If the cursor shows at the end of the line and a colon *does not* show after the line number,



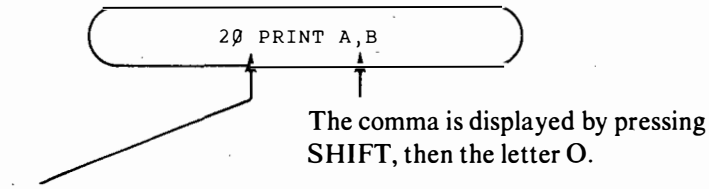
the ENTER key *has not been pressed.*

If the cursor does not show and there is a colon following the line number,

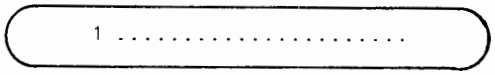


the ENTER key *has been pressed.*

Suppose that you have pressed the ENTER key after the first statement at line 10. The computer waits for you to type in the next line of the program. It doesn't give you any visual indication, but it is waiting for you to go on. Type in the next line. For example, type



If you put a colon of your own after the line number, the computer will add another. You'll then get an error message when you run the program.



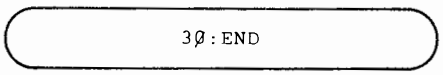
To enter line 20, the sequence of keys pressed should be:

2 0 SPC P R I N T SPC A SHIFT O B ENTER

The display would then show

Now finish off the program by adding an END statement. Type 30 END, and (of course) press ENTER.

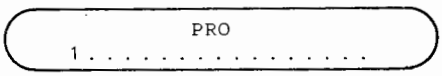
The display:



The END statement is optional If it is not there, the computer will stop after the last executable line. However, sometimes the END statement must be used to separate the end of the program from data or other information that might follow.
We know that you are just itching to run the program that you just entered, so . . .

The RUN Mode

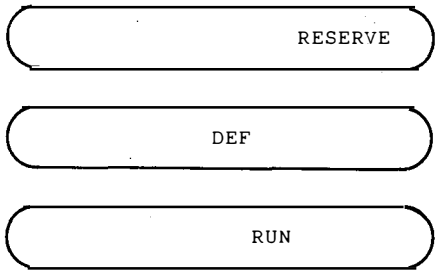
Suppose that you tried to run the program while you were in the PRO (Program) mode. If you did, this is what you probably saw:



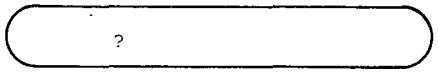
As you know, this is an error code.

To clear an error code: press the red CL key.

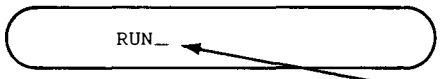
You can't RUN a program while you are in the PRO mode. Press the MODE key three times.



Now, you can type RUN.
If you remembered to press the ENTER key after typing RUN, you saw this:

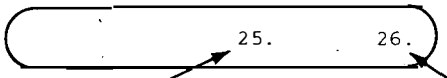


The question mark means ready for an INPUT. However, if you forgot to press the ENTER key after typing the word RUN, the computer is still waiting at this point.



Press the enter key to see the question mark prompt.

Try 25 for an input. Don't forget to press the ENTER key following the input. This is what you will see:



There's A, your input

There's B, (B = A + 1)

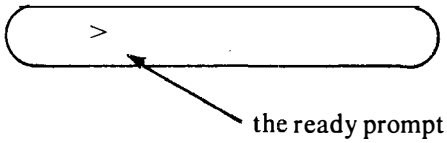
> When the Pocket Computer encounters a PRINT statement, it PRINTS the value specified and waits and waits and waits. It will not do anything else until you press the ENTER key.

When you press the ENTER key, the next statement:

```
30:END
```

is executed, and the ready prompt shows at the beginning of an otherwise empty line.

The ready prompt indicates that the Pocket Computer is ready and waiting for the operator to input a command for it to do something.



the ready prompt

Most computers go on after a PRINT statement, but the Pocket Computer can display only one line at a time. If it didn't stop and another PRINT statement were executed, the first results would disappear from the display before you had a chance to read them.

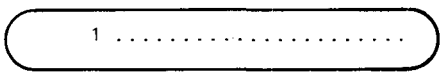
Let's change the program to print each variable on a separate line.

```
1Ø INPUT A: B=A+1
2Ø PRINT A
3Ø PRINT B
4Ø END
```

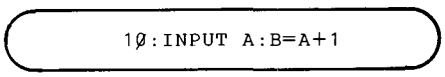
change these lines

Before you enter the changes, go to the PRO mode. Since you were last in the RUN mode, you should only have to press the MODE key once. Then enter the new lines (20, 30, and 40) and LIST the new program to make sure that it is correct.

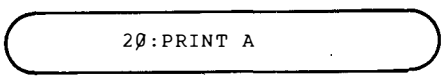
The LIST command works only in the PRO mode. If you try it in any other mode, you'll get an error code.



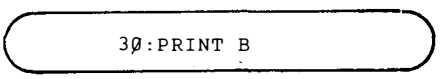
When you are in the PRO mode, you can successfully type LIST, and press the ENTER key.



Press the down arrow key [▼] once. This scrolls the program down to the next line.



Press [▼] again.



Press ↓ again.

```
40:END
```

If you press ↓ again, nothing else will happen. That's the end of the program.

Now, press the MODE key until the computer is in the RUN mode (three times should do it). Type RUN and press ENTER. The program begins execution and the input prompt appears.

```
?
```

Type 25 (your input).

```
25.
```

The value for A is printed by line 20.

Press the ENTER key to go on.

```
26.
```

The value for B is printed by line 30.

Press the ENTER key to END the program.

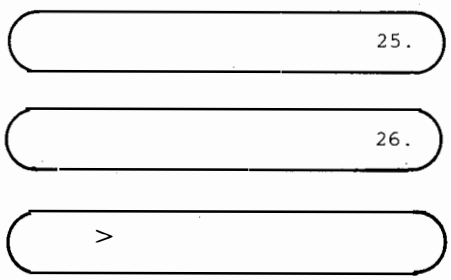
```
>
```

The ready prompt appears.

Now you know how the PRINT statement works. You might not want the computer to stop after it has printed something. Wouldn't it be nice if it would just flash the answer on the display for a brief time and go right on to the next statement? Well, go back to the PRO mode once more and make these changes to your last program.

```
20 PAUSE A
30 PAUSE B
```

Once you have entered the changes, go back to the RUN mode. Run the program. After you've typed in your input (25) and pressed ENTER, watch the right end of the display closely.



ON, then OFF

ON, then OFF

The program's over.

Did you see the computer display the 25 for almost a second, blink off, display the 26 for about a second, blink off, and then display the prompt (>)?

> PAUSE works the same way as PRINT, except execution of the program is halted for 0.85 seconds. The computer then continues.

So, if your eyes are quick enough, you can flash results on the display without stopping the computer. It merely makes a PAUSE before going on to the next statement. What would happen if you said PAUSE in line 20 and PRINT in line 30? Try it by changing these lines:

```
2Ø PAUSE A
3Ø PRINT B
```

Once you've discovered what happens, write a program that will count from 1 through 10. Each time, have the computer flash the counting number, then display 2π times the counting number, then stop to wait for you to press the ENTER key before going on.

Your Program

10

20

30

40

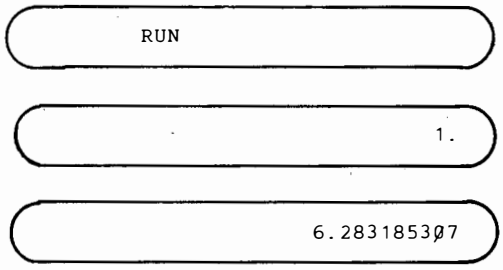
50

60

First type NEW.

> In the DEF, RUN, OR PRO modes, NEW clears all program and data memories, but *not* reserve memories. In the RESERVE mode, it clears all reserve memories, but *not* program and data memories.

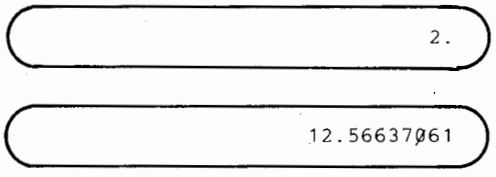
Then enter your program. Go to the RUN mode to execute it. This is what the display shows:



Flashes

Stops

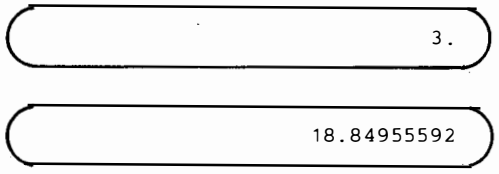
Press ENTER.



Flashes

Stops

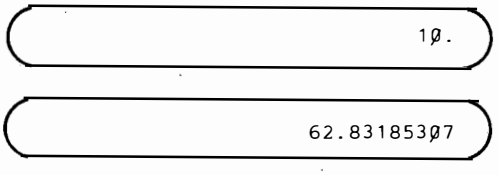
Press ENTER.



Flashes

Stops

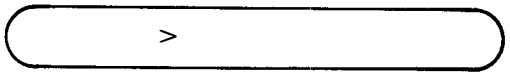
Press ENTER.



Flashes

Stops

Press ENTER.



End of program

Here's how we wrote our program:

```
10 FOR A=1 TO 10
20 B=2*π*A
30 PAUSE A
40 PRINT B
50 NEXT A
60 END
```

Your program may look entirely different; but if it does the job, it is OK. No two programmers do the same thing in the same way. If the program works, that's all that's necessary.

Notice line 20 in our program.

> The Pocket Computer has a built-in value of 10-digit accuracy. The π key is in the top row of keys and must be "shifted."

To enter, press SHIFT and then π.

Other than the π key, this program looks just like a BASIC program on most other computers. Notice the accuracy displayed in the run.

> The TRS-80 Pocket Computer displays numbers to 10 digits.

Of course, you knew that you were finding the circumference of a circle (B) with the given radius (A) in the last program.

$$C = 2\pi r$$

Most BASIC languages require the multiplication symbol (*) to be entered between each of the values being multiplied. Do you think the previous program would work if you changed line 20 to:

20 B=2π A

without the multiplication operator sign? Change the line and try it.

> The Pocket Computer will do this as long as there's no chance for misunderstanding. If you want to multiply two constants, such as 3 times 5, 35 will not work — it has to be 3*5; but 2A will work as well as 2*A since one value is a constant and one a variable.

Let's change the program slightly again. This time make both PRINT statements PAUSE. Also extend the upper limit of the FOR-NEXT loop to 100.

```
10 FOR A=1 TO 100
20 B=2*π*A
30 PAUSE A
40 PAUSE B
50 NEXT A
60 END
```

Suppose that you want to see the program execute each step. Maybe you're not sure it is operating as it should.

> The Pocket Computer has a DEBUG command that allows you to watch the program execute line by line. The DEBUG command is only effective in the DEF and RUN modes.

1) Enter the program in the PRO mode.
2) Access the RUN mode.
3) Now DEBUG using the following steps:

You	Display	
type DEBUG	1Ø:	Stops
press .	2Ø:	Stops
press .	1.	Flashes
	3Ø.	Stops
press .	6.283185307	Flashes
	4Ø:	Stops
press .	5Ø:	Stops
press .	2Ø:	Stops
press .	2.	Flashes
	3Ø:	Stops
press .	12.56637Ø61	Flashes
	4Ø:	Stops

.
.
.
etc.
.
.
.

If the program had not been operating properly, you could have seen where the error was occurring. The line number is displayed for each step. The ▪ key is pressed to go on to a new step in the program. Notice how the FOR-NEXT loop worked as the program went back to line 10 after line 50 had been executed.

Do you wonder how much memory the program has used? You can find out. Remember, the Pocket Computer has memory for 1,424 program steps or data memories. If you are familiar with Radio Shack's Level II BASIC, you have probably used the MEM command. You can also use it with the Pocket Computer.

> The MEM command functions in all modes. It will display the number of program steps and flexible memories *not* being used.

So, regardless of the mode that you are in, type MEM and press the ENTER key.

```
1386STEPS    173MEMORIES
```

There were 1,424 program steps or 178 data memories available before the program was entered. The display now shows that there are 1,386 program steps or 173 data memories unused. Therefore, you have used 38 program steps or 5 flexible memories, whichever way you want to look at it. You have also used two fixed memories (A and B). There is lots of room for much longer programs.

Let's get back to the last program. RUN the program this time instead of using DEBUG. Suppose the phone should ring or the coffee pot starts to boil over when you have it running. You can press the ON key ON . Press it only once, and the program will "break." A message appears on the display to let you know where the program sequence was broken, such as:

```
BREAK AT 20
```

The program was stopped at line 20.

Now you can go answer the phone or turn off the coffee pot. When you come back:

a) If you've been gone less than seven minutes, type CONT, and the program will continue from the point where the break occurred.
b) If you've been gone longer than seven minutes, the computer automatically turns off to save its batteries. When you press ON, the data memories are cleared, but the program is still there. You'll have to RUN the program from the beginning again, however.

Summing Up Chapter One

You now have some first-hand knowledge of the TRS-80 Pocket Computer. The following table shows commands and statements used in this chapter.

COMMANDS

Command	Accepted Abbreviation	Remarks	Used in These Modes
CONT	C. CO. CON.	Restarts a program from the point at which interrupted.	DEF or RUN
DEBUG	D. DE. DEB. DEBU.	Starts execution of a program line by line. Execute each line by pressing.	DEF or RUN
LIST	L. LI. LIS.	Lists programs line by line. Press [↓] to see next line. [↑] to see previous line.	PRO
MEM	M. ME.	Shows remaining unused memory.	Any mode
NEW		Clears program and data memories.	DEF, RUN, or PRO
		Clears reserve memories.	RESERVE
RUN	R. RU.	Begins execution of a program.	DEF or RUN

Note: Although it has not been mentioned, some commands and statements may be abbreviated when entered. They will be displayed here in their unabbreviated form, but accepted abbreviations, shown above, can be used.

STATEMENTS

Statement Introduced	Accepted Abbreviation	Remarks
END	E. EN.	Indicates end of program.
FOR	F. FO.	Starts a FOR-NEXT loop.
INPUT	I. IN. INP. INPU.	Stops a program for data input. ? is displayed.
NEXT	N. NE. NEX.	End of FOR-NEXT loop. Increments the loop variable.
PAUSE	PA. PAU. PAUS.	Specified value is displayed for 0.85 seconds.
PRINT	P. PR. PRI. PRIN.	Specified value is displayed. Program execution is halted until the ENTER key is pressed.

Note: Periods in abbreviations must be included.

Many facts were presented to you in this chapter. Here is a list of the most important ones.

- Twenty-four characters may be displayed at one time.
- Numeric variables are displayed with ten digits.
- Memory consists of:
  - 26 fixed memories, and
  - 178 flexible memories, or
  - 1,424 program steps.
- Numeric or string variables (up to seven characters) may be used.

- BASIC statements and commands are similar to BASICs of larger systems.
- Four modes of operation offer flexible use:
  (1) PROgram mode – for entering normal programs.
  (2) RUN mode – for executing programs.
  (3) DEF mode – to execute programs defined by a label.
  (4) RESERVE mode – for writing functions for reserved keys.
- A special key, MODE is used to change the operation mode.
- More than one statement may be placed on one program line.
- Some keys have two functions. The SHIFT key is used to access the second function.
- The ENTER key must be pressed at the end of each program line.
- Error codes are given for incorrect operations.
- The clear key CL clears an error message from the display.
- The Pocket Computer stops operation each time a PRINT statement is executed.
- Programs may be LISTed, one line at a time in the PRO mode.
- A PAUSE statement works like a PRINT statement, except the program pauses for about a second to display the result and then goes on.
- The NEW command clears all program and data memories when used in the DEF, RUN, or PRO mode. It clears all reserve memories when used in the RESERVE mode.
- A built-in π key is available with 10-digit accuracy.
- A DEBUG command can be used in the DEF and RUN modes to display the execution of a program, one line at a time.
- The MEM command can be used in any mode to display the amount of unused memory.

Chapter One Self-Test

1. You used a program that calculated the circumference of a circle in this chapter. Expand that program so that it will also compute the area of the circle.

   HINT: $A = \pi r^2$

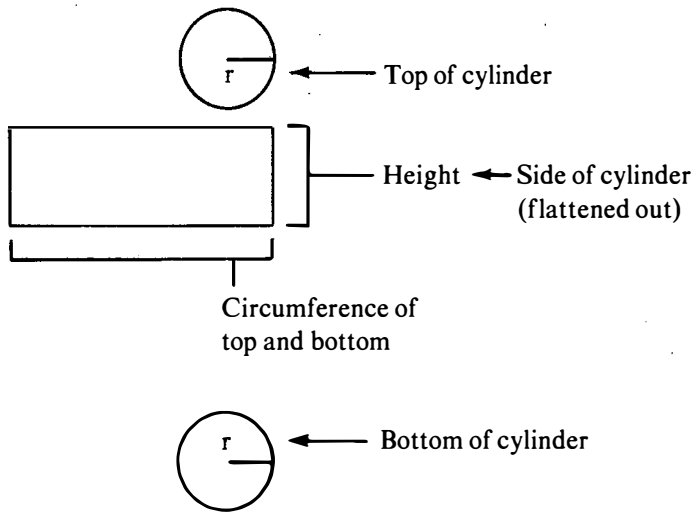
2. Modify your answer to Problem 1 so that the results will be displayed as in this example:

`RADIUS = 3`	Flash and go on
`CIRCUMF. = 18.84955592`	Print and stop
`AREA = 28.27433388`	Print and stop

   HINT: Strings can be printed by enclosing characters to be printed in quotation marks (`PRINT ''RADIUS =`").

3. Now change the program so that you can also input the height and radius of a cylinder. Compute and print the total surface area of the cylinder along with previous information.

HINT:



Answers To Chapter One Self-Test

These solutions are merely one way to solve the problem. Yours may differ. As long as yours produce the desired results, they may be considered correct.

1. From the program on page 17.

```
10 FOR R = 1 TO 10
20 B = 2*π*R
30 PAUSE R
40 PRINT B
50 A =π*R*R          <-- or A= *R 2
60 PRINT A
70 NEXT R
80 END               <-- Optional
```

2. We modified lines 30, 40, and 60.

```
.
.
30 PAUSE ''RADIUS =';R
40 PRINT ''CIRCUMF. =';B
.
60 PRINT ''AREA =';A
```

3. Added lines have been inserted as needed.

```
10 FOR N = 1 TO 10   <-- This could be any range.
13 INPUT ''R=?';R
16 INPUT ''H=?';H
20 B = 2*π*R
30 PAUSE ''RADIUS =';R
40 PRINT ''CIRCUMF. =';B
50 A =π*R*R
60 PRINT ''AREA =';A
63 S = (2*A)+(B*H)
66 PRINT ''SURF. AREA =';S
70 NEXT N
80 END
```

CHAPTER TWO

Applications, Memory Use, and Definable Mode

There have been many questions asked about the personal use of computers. One of the most frequently posed questions is, "What do you do with a computer after you get tired of playing computer games?" Well, computers have many practical functions too, ranging from keeping a shopping list for your family to running an inventory program for a business.

Computers serve three functions: recording information, processing information, and transmitting information. The TRS-80 Pocket Computer adds new dimensions to these functions. In the past, the size of computers confined their use to a limited number of places. A source of power was needed, and even the desk-top computer was heavy and cumbersome to carry around. But because the TRS-80 is portable, it is possible to record information at any place and at any time, and to have any previously recorded information transmitted to you by the computer for immediate use. Like larger computers, the Pocket Computer has a memory that is nonvolatile, which means it can store information permanently. (Information from the Pocket Computer can also be stored a second way, on cassettes, which we will discuss in a later chapter.) The portability of the Pocket Computer also enables you to process information immediately, for instant review.

In this chapter you will:

- learn about possible uses of the Pocket Computer;
- learn more about how memory is used;
- learn how to use the Pocket Computer in the DEFinable mode; and
- apply some problem solving techniques to practical applications.

Application Examples

Let your imagination run loose as you think of ways to use your Pocket Computer. We'll plant a few seeds to start your "garden of ideas."

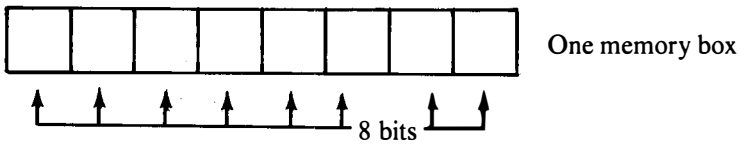
- Up-to-the-second stock market analysis
- Horse-race handicapping
- Shopping list
- Shopping price
- Letter reader and writer
- Short story reader
- Speed-reading machine
- Cue for speeches
- Memo pad
- Record job time for accounts
- Record medical data, such as temperature throughout day, foods eaten, heart rate, blood pressure, and activity level
- Compute food allotment given present weight, to aid in diet for weight gain or loss
- Time payment schedules
- Compound interest
- Amortization
- Annuities
- New games
- 100 most misspelled words
- Poker game
- Dice games
- ESP games
- Record people contacted during day
- Record reference tables
- Record class activity for later dump to cassette tape (or printer)
- Arithmetic practice

There are many others which will apply to your own needs.

Pocket Computer Memory

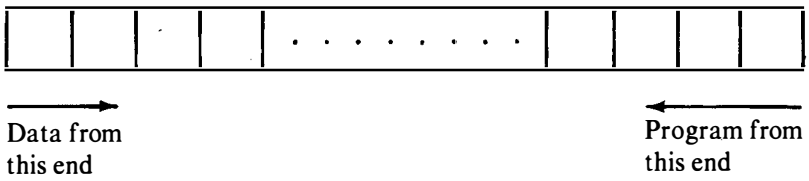
One of the things that any self-respecting computer can do is to store information. The TRS-80 Pocket Computer stores an amazing amount of information for its tiny size. And when information is stored, it doesn't go away when the computer is turned off. The information you store stays in memory until the day that you clear or replace it. To use this memory power effectively, you will need to understand how data and program steps compete for memory space.

It is useful to think of the TRS-80 memory as a long sequence of 1,632 boxes. Each box contains 8 bits of information. In more technical language, each box will hold an 8-bit byte.



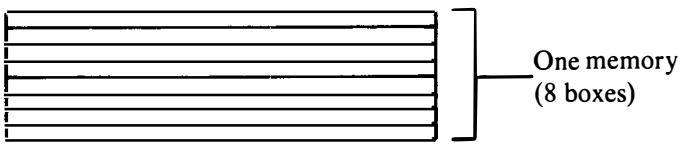
8 bits, combined in a group, make one byte

Program steps are stored in the memory boxes starting at one end, while data items (consisting of numbers and words) are stored starting at the other end.

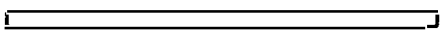


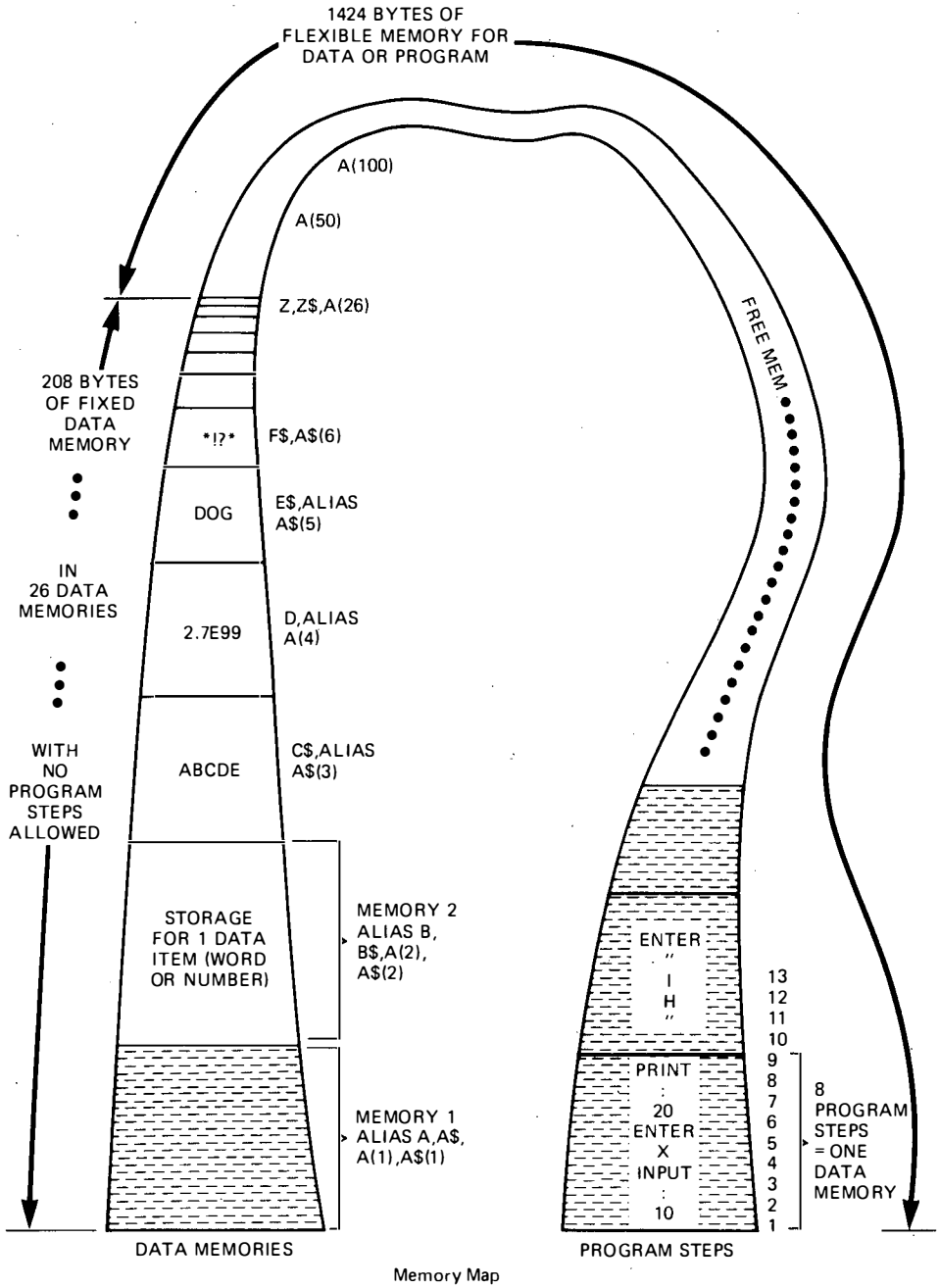
When data and program meet in the middle, the memory is full.

More space is needed to store an item of data than to store a program step. One memory, consisting of 8 boxes, is required to store one data item.



A single box is required to store one program step.




1424 BYTES OF FLEXIBLE MEMORY FOR DATA OR PROGRAM
A(100)
A(50)
Z,Z$,A(26)
208 BYTES OF FIXED DATA MEMORY
!?
F$,A$(6)
DOG
E$,ALIAS A$(5)
IN 26 DATA MEMORIES
2.7E99
D,ALIAS A(4)
WITH NO PROGRAM STEPS ALLOWED
ABCDE
C$,ALIAS A$(3)
STORAGE FOR 1 DATA ITEM (WORD OR NUMBER)
MEMORY 2 ALIAS B, B$,A(2), A$(2)
MEMORY 1 ALIAS A,A$, A(1),A$(1)
DATA MEMORIES
FREE MEM
ENTER " I H "
PRINT : 20 ENTER X INPUT : 10
13 12 11 10 9 8 7 6 5 4 3 2 1
8 PROGRAM STEPS = ONE DATA MEMORY
PROGRAM STEPS

Memory Map

The first 26 data memories (208 boxes) are used only for data storage and are not available for program storage. This leaves 1,424 of the original 1,632 boxes for storage of other data and program steps. The longest possible program could therefore consist of 1,424 program steps. Since each data item occupies 8 boxes, the 1,424 boxes could be used to store 178 data items (of course there would then be no room for program steps).

Data memories have names. In fact, each data memory has many names. The name, or names, you use will depend on whether the data item is a number or a word, and on your particular preference.

Example:

If the data item in memory number 1 is a number, then the following names can refer to it:

A, or A(1)

If the data item in memory number 1 is a string of symbols, then the following names can refer to it:

A$, or A$(1)

String variables have a $ sign, number variables do not.

The data in memory number 2 is referred to by the names:

B, or A(2), or B$, or A$(2)

The data in memory 3 is referred to by the names:

C, or A(3), C$, or A$(3)

Other memories follow the same pattern.

This is a very versatile naming system. It allows you to use the alphabetic order A, B, C, . . . Z for number variables; or A$, B$, C$, . . . Z$ for string variables. If the occasion demands, you can use subscripted variables such as A(1), A(2), A(3), . . . A(204), or A$(1), A$(2), A$(3), . . . A$(204).

To see how memory is filled as you enter a program in the PRO mode, you can look at the unused memory size by the MEM command. Go through the following program on your Pocket Computer as we check MEM after each entry. Do the same on your computer.

The program that we'll use will compute the volume of a cylinder, given the height and radius of the cylinder.

Equation: $v = r^2h$ where r = radius
h = height
v = volume

Program:

```
10 INPUT ''RADIUS='';R
20 INPUT ''HEIGHT='';H
30 V= *R 2*H
40 PRINT ''VOLUME='';V
50 GOTO 10
```

Notice that INPUT statements can contain strings. One data memory can hold a maximum of seven characters.

Also notice line 30, the volume equation.

The TRS-80 Pocket Computer has a key for raising a value to a power. The ^ key is in the top row of keys and must be "shifted."

Press SHIFT, then 

Before you enter any of the program, access the PROgram mode and type NEW. Then type MEM.

```
1424STEPS 178MEMORIES
```

All memories are clear.

Type this much.

```
10 INPUT
```

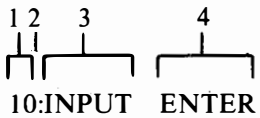
Press ENTER.

```
10: INPUT
```

Type MEM.

```
1420STEPS 177MEMORIES
```

You have used four program steps:



Four program steps equal a part of one memory. Therefore, there are 1,420 steps left, or 177 memories.

Type:

```
1Ø INPUT ''
```

Press ENTER.

```
1Ø:INPUT ''
```

Type MEM.

```
1419STEPS 177MEMORIES
```

Now you've used five program steps — that's still only one memory.

Type:

```
1Ø INPUT ''R
```

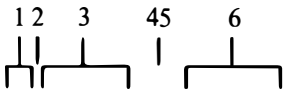
Press ENTER.

```
1Ø:INPUT ''R
```

Type MEM.

```
1418STEPS 177MEMORIES
```

Now six steps have been used — still one memory. This is how it's stored so far:



Let's go on and finish the first line.
Type:

```
1Ø INPUT ''RADIUS='';R
```

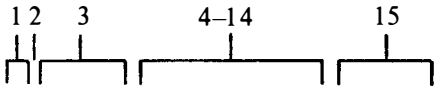
Press ENTER.

```
1Ø:INPUT ''RADIUS='';R
```

Type MEM.

```
1409STEPS 176MEMORIES
```

This is the result from the first line of the program:



1424 original memories
− 15 steps used

1409 unused program steps
(or 176 complete memories left since the 15 steps only take 2 memories at 8 steps per memory)

Now enter the second line.

Type:

```
2Ø INPUT''HEIGHT='';H
```

Press ENTER.

```
2Ø:INPUT ''HEIGHT='';H
```

Type MEM.

```
1394STEPS 174MEMORIES
```

Stored as 20: INPUT"HEIGHT=";H ENTER.

15 more steps
15 + 15 = 30 steps so far

1424 − 30 = 1394 steps unused

30 ÷ 8 = 3 full memories and
a part of the fourth; therefore
174 complete memories
unused

Finish entering the program, then check the amount of memories unused.

```
1361STEPS 17ØMEMORIES
```

1424 originally
−1361 left

63 program steps used

63 ÷ 8 = 7 full memories used and a part of the eighth memory; therefore 170 full memories unused

Step	Data
1	10
2	:
3	INPUT
4	"
5	R
6	A
7	D
8	I
9	U
10	S
11	=
12	"
13	;
14	R
15	ENTER
16	20
17	:
18	INPUT
19	"
20	H
21	E
22	I
23	G
24	H
25	T
26	=
27	"
28	;
29	H
30	ENTER
31	30
32	:
33	V
34	=

Step	Data
35	π
36	*
37	R
38	^
39	2
40	*
41	H
42	ENTER
43	40
44	:
45	PRINT
46	"
47	V
48	O
49	L
50	U
51	M
52	E
53	=
54	"
55	;
56	V
57	ENTER
58	50
59	:
60	GOTO
61	1
62	0
63	ENTER

Fixed Memories Used

Memory Number	Variable Stored
8	H
18	R
22	V

Memory Use for the 63 Program Steps in the Cylinder Volume Program

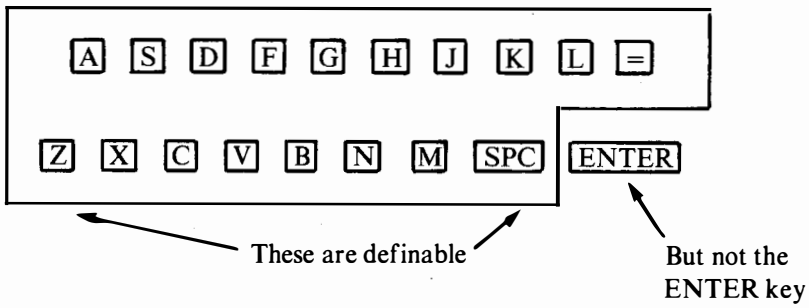
You can see that each BASIC statement such as INPUT, PRINT, and GOTO occupies only one program step. Each time the ENTER key is pressed at the end of a line, a program step is used. A line number occupies one program step, and the colon following the line number occupies a separate step. Otherwise, each keyboard character requires its own program step whether it is a letter, number, or arithmetic operation. Variables are stored in fixed memory. Multiple statements in a line save memory, but sacrifice the clarity of your program.

Now that you have a better understanding of memory use, let's move on to another Pocket Computer operating mode.

The DEFinable Mode

The DEF mode is used in place of the RUN mode when your programs have been *defined* by a label. It is an execution mode, like the RUN mode.

The lower two rows of gray keys are definable keys.



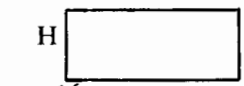
When more than one program is in memory at the same time, it is convenient to label each program with one of the definable keys. Then you can execute any one of the programs by pressing the SHIFT key, followed by the defined key that is used to label the program. It is much easier to press the appropriate defined key than to try to remember the beginning line number of each program and to type in RUN, line number, ENTER.

Recalling equations that you used in earlier programs and using two new ones, let's set up three programs to solve them.

$C = \pi R^2$ — Area of circle (top or bottom of a cylinder)

$A = 2\pi RH$ — Surface area of side of cylinder

Top of cylinder



Side of cylinder (flattened out)

Bottom of cylinder

$V = \pi R^2 H$

Volume of cylinder

$S = 2\pi R^2 + 2\pi RH$ — Total surface area of cylinder

or

$= 2\pi R\,(R + H)$

We will use three separate programs to compute and print the surface area and volume of a cylinder given the radius and height.

First program – input the radius and height
Second program – compute and print the surface area
Third program – compute and print the volume

The Cylinder Program

First program with "=" label inputs R and H

```
10 ''='':INPUT ''RADIUS='';R
20 INPUT ''HEIGHT='';H
30 END
```

Second program with "S" as label computes surface area

```
40 ''S'':A=2*π*R
50 B=R+H
60 S=A*B
70 PRINT''R='';R;''H='';H
80 PRINT ''SURFACE AREA='';S
90END
```

Third program with "V" as label computes volume

```
100 ''V'':V=π*R  2*H
110 PRINT''R='';R;''H='';H
120 PRINT ''VOLUME='';V
130 END
```

In the first program, the radius and height are input. The program is executed in the DEF mode and establishes the values for the radius and height of the cylinder.

Press the MODE key until the DEF mode is found.

Press SHIFT, = To access the first program labeled "="

```
RADIUS=
```

Press 2, ENTER

```
HEIGHT=
```

PRESS 3, ENTER

```
>
```

← The program has ended

Now, you may select either program "S" if you want the surface area of the cylinder or program "V" if you want the volume of the cylinder.

a) To select the surface area program:

Press SHIFT, S to access the second program labeled "S."

```
R=2.H=3.
```

Reminding you of the initial values

Press ENTER.

```
SURFACE AREA=62.83185305
```

Press ENTER.

```
>
```

End of program "S"

b) To select the volume program:
Press SHIFT, V To access the third program labeled "V."

```
R=2.H=3.
```

Press ENTER.

```
VOLUME=37.699111.84
```

Press ENTER.

```
>
```

End of program "V"

To access the first program again, so you can input new values for r and h, press SHIFT, [=], then input new values as requested. Use programs "=", "S", and "V" to complete the exercises in Problem 1. (Answers at end of chapter.)

Inputs		Surface Area	Volume
R	H		
15	10		
14.5	11.5		
14	12		
13.5	12.5		
13	13		
12.5	13.5		
12	14		

The TRS-80 has a statement called AREAD, which will automatically store a numeric value or character in a specified variable that has been displayed *before* the start of program execution. In our last example, you used a program labeled "=" which allowed you to input the values for radius and height. Let's replace program "=" with AREAD statements to see how they work.

In the PRO mode, replace the first three lines of the previous program. Leave programs "S" and "V" just as they are.

Replace lines 10, 20, and 30 only.

```
10 ''A'':AREAD R
20 ''B'':AREAD H
30 END
```

Now go to the DEF mode and press in succession:

5, SHIFT, A ← Enters 5 for R
then 6, SHIFT, B ← Enters 6 for H
then SHIFT, S ← Runs program "S"

```
R=5.H=6.
```

Press ENTER.

```
SURFACE AREA=345.5751919
```

Press SHIFT, V. ← Runs program "V"

```
R=5.H=6.
```

Press ENTER.

```
VOLUME=471.238898
```

The AREAD statement allows for a neat and quick way to enter variables.

Suppose you want to leave the radius at 5 and explore various values for the height. Just continue as before — but ignore the AREAD R statement, which is labeled A.

In the DEF mode:

Press in succession:

7, SHIFT, B. ← Changes H to 7

Press SHIFT, S.

R=5.H=7. — See H was changed

Press ENTER.

SURFACE AREA=376.9911185

Press SHIFT, V.

R=5.H=7.

Press ENTER.

VOLUME=549.7787144

Press 10, SHIFT, B. ← Changes H to 10
Press SHIFT, S.

R=5.H=10.

Press ENTER.

SURFACE AREA=471.2388981

Press SHIFT, V.

R=5.H=10.

Press ENTER.

VOLUME=785.3981634

Continue on in this manner until you have used as many different heights as you desire. You could go through a similar procedure to change only the radius.

> Note that when using a program with an AREAD statement, a value must be displayed *before* the labeled line with the AREAD statement is executed.
>
> 15, SHIFT, A in example

Using Arrays

A more versatile program can be written by using an array to hold a series of results produced by a program such as the previous one.

Problem to be solved:

> Compute the surface area of ten cylinders having a given radius but varying heights. Input the radius, beginning height, and the amount (increment) that the height is to change each time. Store the results in an array. After all areas have been computed, print them.

In planning a solution to such a problem, state the necessary actions in general statements.

1) Recording: Input initial values
2) Processing: Make necessary computations
3) Transmitting: Print results

You must also plan how you are going to use memory space so that no conflict arises between variables or between space for variables and space for program steps.

1) You need ten memories for the array that will hold the computed surface areas. You might use A(1) through A(10).
2) The FOR-NEXT loop variable to be used in computing and storing the surface areas will occupy one memory. N might be used.
3) You will need three variables for the radius, original height, and the increment to the height. Use A(20), A(19), and A(18). A(17) is used for the original height also since you will be changing A(19) as the different surface areas are computed.
4) The equation used to solve the surface area is long and is therefore broken into two parts:

   Z = R + H
   Y = 2 * π * R
   Surface area = Z * Y
   Z and Y take two more memories.

In all, it looks like you will use 17 memories. These will all fit into the fixed memory space. Therefore, they will not interfere with the memory used for program steps.

Memory Number	Variable Stored	Remarks
1	A(1)	The first ten hold the computed areas
2	A(2)	
3	A(3)	
4	A(4)	
5	A(5)	
6	A(6)	
7	A(7)	
8	A(8)	
9	A(9)	
10	A(10)	
11		
12		
13		
14	N	← FOR-NEXT loop variable
15		
16		
17	A(17)	← Original height
18	A(18)	← Height increment
19	A(19)	← Changing height
20	A(20)	← Radius

Memory Number	Variable Stored	Remarks
21		
22		
23		
24		
25	Y	← 2 * π * Radius
26	Z	← Radius + height

Fixed Memories Used for Surface Area Program

It is a good idea for you to make a table of memories used for your programs. Then when you wish, you can examine the memories to see what values are there; if there are errors in a program, this is a great aid to find where they may be occurring. We will cover error codes, editing, and debugging programs in Chapter 3.

To conform with the way the program has been planned, it would be convenient for you to separate the program into three parts and label each part. The DEF mode would then be used to execute the program. Here is a three-part program for surface area.

```
10 '':INPUT ''RADIUS='';A(20)
20 INPUT ''FIRST HEIGHT='';A(19)
30 INPUT ''INC FOR H='';A(18)
40 A(17)=A(19)
50 END

100 ''S'';FOR N=1 TO 10
110 Z=A(20)+A(19)          <-- radius + height
120 Y=2*π*A(20)            <-- 2πR
130 A(N)=Z*Y               <-- 2πR(R + H)
140 A(19)=A(19)+A(18)      <-- new height
150 NEXT N
160 BEEP(3)
170 END

200 ''L'':A(19)=A(17)      <-- set height back to
210 FOR N=1 TO 10              original value
220 PRINT A(19),A(N)
230 A(19)=A(19)+A(18)      <-- next height
240 NEXT N
250 END
```

Notice that each part of the program has a separate label. Enter each part in the PRO mode. Execution will be performed in the DEF mode. Look over the program carefully and see if you notice anything new. How about BEEP (3) — a surprise that we won't give away yet. You'll probably figure out what BEEP will do, but no matter. We still won't tell you yet.

All right, you're now ready to RUN. Get into the DEF mode and let's go.

You Do This	Computer Display	Remarks
	> DEF	When in DEF mode
SHIFT, [=]	RADIUS=_	
5, ENTER	FIRST HEIGHT=_	
1, 0, ENTER	INC FOR HEIGHT=_	
3, ENTER	>	End of "=" program
SHIFT, S	RUN	Computations being made
Wait until	→	BEEP! BEEP! BEEP!
	>	End of "S" program

All calculations are finished. The 3 BEEPS were caused by `160 BEEP (3)`. It gives you an audible indication that the program has finished its chore.

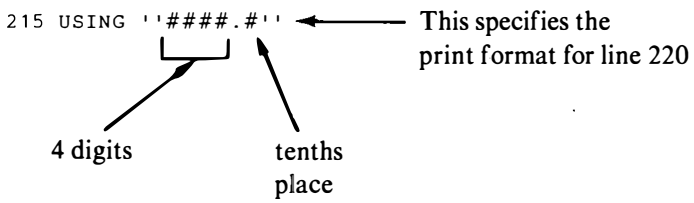
Now let's search for the results in program "L".

SHIFT, L	1Ø. 471.2388981	Height followed by surface area
ENTER	13. 565.4866777	
ENTER	16. 659.7344573	
ENTER	19. 753.982237	
ENTER	22. 848.23ØØ166	
ENTER	25. 942.4777962	
ENTER	28. 1036.725576	
ENTER	31. 113Ø.973355	
ENTER	34. 1225.221135	
ENTER	37. 1319.468915	
ENTER	>	End of "L" program

Isn't that neat? Look at all those numbers — that kind of accuracy is ridiculous for most measurements. See if you can be more reasonable. Suppose each measurement, R and H, had been made to the nearest tenth of a unit. Then you could expect the answer to be no more accurate than to the nearest tenth. You can force the computer to print the answer to tenths (or other places) with the USING statement.

> The USING statement is an instruction to specify a PRINT or PAUSE display format for *numerical* data.

To demonstrate this statement, go back to the PRO mode and type in this line:



Now go to the DEF mode and do it all over again. Input a radius of 5, original height of 10, and an increment of 3. Then, on to program "S" to compute the surface areas. When the computer beeps 3 times, type SHIFT L to see the following results.

Press	See this result	
SHIFT, L	10.0 471.2	Height, then area
ENTER	13.0 565.4	
ENTER	16.0 659.7	
ENTER	19.0 753.9	
ENTER	22.0 848.2	
ENTER	25.0 942.4	
ENTER	28.0 1036.7	
ENTER	31.0 1130.9	
ENTER	34.0 1225.2	
ENTER	37.0 1319.4	

Use the same program to complete the table in Problem 2. Use a radius of 2.35, original height of 5.50, with increments of 0.25. Change line 215 to print the surface areas to the nearest hundredth. (Answer at end of chapter.)

Try #	Height	Surface Area
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Summing Up Chapter Two

Three new Pocket Computer BASIC statements were introduced in this chapter.

Statement	Accepted Abbreviations	Remarks
AREAD X	A. AR. ARE. AREA.	The contents displayed at the start of a defined program are automatically read into the specified variable.
BEEP(X)	B. BE. BEE.	A short tone (beep) is generated as many times as specified by value specified (X).
USING	U. US. USI. USIN.	Designates the format to be used for numerical data in a PRINT or PAUSE statement.

Other material presented includes:

- Information on the Pocket Computer's Memory:
  One memory box holds 8 bits of data.
  Eight memory boxes are used for each alphanumeric character.

  Names that may be used for memories:

A,B,C,D ..... Z
A$,B$,C$,D$ ..... Z$
A(1),A(2),A(3) ..... A(204)
A$(1),A$(2) ..... A$(204)

- Prompt messages can be used in INPUT statements:

```
INPUT ''NAME=?'';A$
```

- How memory is used as you enter a program:

Program steps are stored at the opposite end of memory from variables with eight program steps filling one memory location.

One BASIC line may use several program steps.

Variable values may be assigned to memory locations at the opposite end of memory.

Care should be taken in assigning variables so that memory used for variables does not overlap that used for program.

- Definable mode – The lower two rows of keys can be defined to execute subprograms that have been labeled by shifting a definable key.
- Arrays are stored in memory as subscripted variables such as:

A(1), A(2), A(3), ...A(204) for numeric variables,
or A$(1), A$(2), A$(3), ...A$(204) for string variables.

A(1) or A$(1) occupy fixed memory #1,
A(2) or A$(2) occupy fixed memory #2,
etc.

If more than one array is to be used in a program, they must occupy *different* memory locations as denoted by their subscripts.

Example: A(1) through A(10) for one array,
A(11) through A(20) for another array,
etc.

Chapter Two Self-Test

1. A cylinder is to have a volume of 3,000 cubic centimeters. Write a program that will let you input the radius and compute the height. Complete the chart for the inputs shown.

   HINT: $V = \pi R^2 H$ What does H equal if you know R and V?

   Volume Radius Height

R	Height
11	
10	
9	

2. Enlarge your program for Problem 1 to calculate the total surface area of the cylinder after the radius has been input and the height calculated.

   HINT: $S = 2\pi R^2 + 2\pi RH$

   Complete the chart.

R	Height	Surface Area
9		
8.5		
8		
7.5		
7		

3. A soup company wants to package soup in cans that will hold 3,000 cubic centimeters. They send an order to a can manufacturer for cans that will hold that amount, *but* they stipulate that the can must use the least amount of material possible. Several radii were investigated to find which radius–height combination would give the correct volume and still have the smallest surface area. Write a program that they might have used. Use your program and search until you find the best radius to the nearest .001 centimeter.

   HINT: Use your program for Problem 2. Also study the results of that problem for a clue as to where to start looking.

Best radius =_____

Best height = _____

Smallest surface area = _____

Ratio R/H = _____

Answers to Chapter Two Self-Test

1.

Program

```
10 INPUT ''RADIUS='';R
20 H='3000/(  *R*R)
30 PRINT R,H
40 GOTO 10
```

R	Height
11	7.891980649
10	9.549296586
9	11.78925504

2.

Program

```
10 INPUT ''RADIUS='';R
20 H = 3000/  *R*R)
30 PRINT R,H
40 S = 2*  *R*(R+H)
50 PRINT S
60 GOTO 10
```

r	Height	Surface Area
9	11.78925504	1175.604676
8.5	13.2170195	1159.842492
8	14.92077591	1152.123859
7.5	16.97652726	1153.429173
7	19.48836038	1165.018937

3. We first tried these:

R	Height	Surface Area
8.3	13.8616587	1155.740202
8.2	14.20180932	1154.188697
8.1	14.55463586	1152.980529
8	14.92077591	1152.123859
7.9	15.30090784	1151.627266
7.8	15.69575376	1151.499763
7.7	16.10608296	1151.750836

Then we tried these:

R	Height	Surface Area
7.84	15.53600158	1151.505877
7.83	15.57571019	1151.498705
7.82	15.61557124	1151.495288
7.81	15.6555855	1151.495638
7.8	15.69575376	1151.499763

Last of all these:

R	Height	Surface Area
7.821	15.61157825	1151.495461
7.82	15.61557124	1151.495288
7.819	15.61956576	1151.495154
7.818	15.62356181	1151.495057
7.817	15.62755939	1151.494997
7.816	15.63155851	1151.494976
7.815	15.63555917	1151.494992

Best radius = 7.816

Best height = 15.63155851

Smallest surface area = 1151.494976

Ratio R/H = 0.5000141218 or 2R H

Answers to Problems in Chapter

1. Cylinder Exercises

Inputs		Surface Area	Volume
R	H		
15	10	2356.19449	7068.58347
14.5	11.5	2368.760861	7595.978337
14	12	2287.079452	7389.025921
13.5	12.5	2205.398043	7156.940764
13	13	2123.716634	6902.07906
12.5	13.5	2042.035225	6626.797003
12	14	1960.353816	6333.450789

2. Surface Area for Cylinders of Radius 2.35

Try #	Height	Surface Area
1	5.50	115.90
2	5.75	119.60
3	6.00	123.29
4	6.25	126.98
5	6.50	130.67
6	6.75	134.36
7	7.00	138.05
8	7.25	141.74
9	7.50	145.44
10	7.75	149.13

CHAPTER THREE

Error Codes, Editing, and Cassette Use

The TRS-80 Pocket Computer performs a wide variety of operations and functions, but there are limits to what any computer can do. The Pocket Computer has ways to let you know when you are asking it to execute a statement or command that it cannot perform. There are a great many ways that such situations can arise. You have probably already found a few of them.

In this chapter you will learn:

- what the Pocket Computer's six error codes mean;
- how to use editing features to insert and delete segments of program lines;
- how to save and load programs via cassette tape;
- how to load a program from tape without disturbing other programs that are already in memory;
- how to generate random numbers; and
- how to load and use a random number-generating subroutine in other programs.

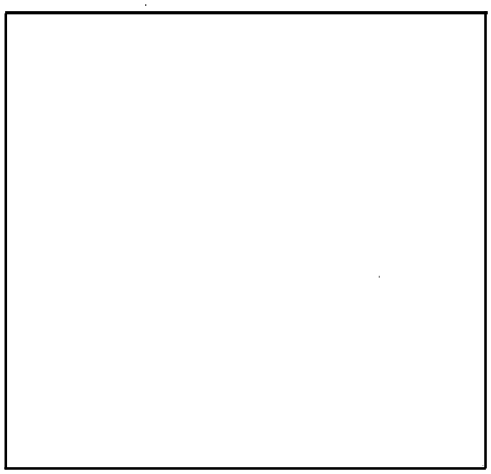
We'll use a solution to the following problem to cause some error codes to appear. Then we'll see if we can correct them with some of the Pocket Computer's editing capabilities.

The Problem - CLIP CLOCK

The ComputerTown Engine Company has a large, round, 24-hour clock on the wall which everyone uses to synchronize their watches before important events. The latest event led to this problem: It was almost midnight, and the stalwart crew loaded up a long program to run while they rested. Big George observed that the program they were about to run consisted of a single subroutine that was called exactly 1,001 times. Each call of the subroutine took exactly one hour. The obvious question is, "What time will the program finish, as measured by the 24-hour clock?" Since this problem may recur, a program is needed that inputs the number of times the subroutine is called, and then prints out the time of day that the program will finish. (You should know the truth about the ComputerTown Engine Company — they drink coffee, play tennis,

talk, and generally carry on until late in the evening.) The ComputerTown Engine Company always start running their programs at midnight. Before looking at our demonstration program, see if you can conjure up a program of your own. Write it in the box provided.

Your Program



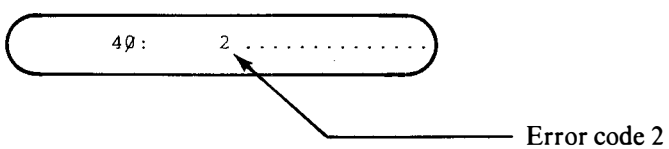
Problem 1 Subroutine
(answer at end of chapter)

Now let's take a look at the program that Big George came up with. These thoughts ran through his mind. "The obvious thing to do is start subtracting 24 from 1,001, since the clock will be back where it started each time it performed 24 subroutine calls. I'll write a program to keep subtracting 24 and test each time to see if fewer than 24 hours remain. When that happens, I'll know when the job is over."

Big George's Solution

```
10 T=1001
20 IF T<24 THEN PRINT T:END
30 T=T-24
40 GOTO 15
```

Well, as you probably realize from looking at the program, Big George is not the best programmer in the world. Enter his program in your Pocket Computer and see what happens when you try to run it. Here's what happened to Big George when he tried.



George held down the [↓] key to see what was wrong.

```
40:GOTO 15
```

Something's wrong about the line number 15

> Error Code 2 indicates a *line error.* It occurs when you try to GOTO, GOSUB, RUN, DEBUG, or LIST a statement that doesn't exist.

Big George pressed the MODE key until he was in the PROgram mode. Then he tried to list line 15.

```
LIST 15_
```

```
2 .............
```

Since we get error code 2 again, we know that line 15 must not exist.

Big George really meant to say: `40 GOTO 20`. He didn't have to retype all of line 40 to correct it. You can use the Pocket Computer's editing ability, just as he did:

Clear the display with the CL key.

Type

```
LIST 40_          PRO
```

Must be in PRO mode

Then press ENTER.

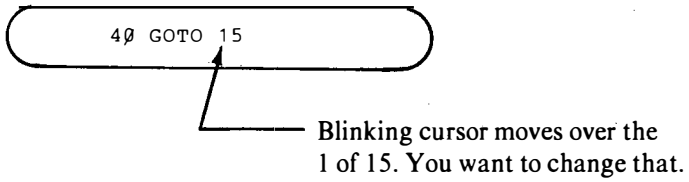
```
40:GOTO 15
```

Press the [▷] key.

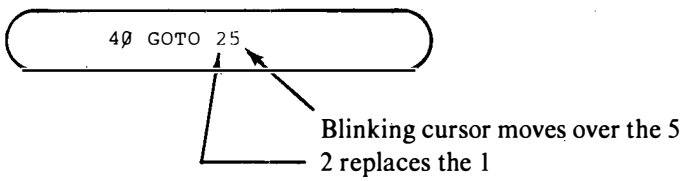
```
40 GOTO 15
```

Cursor blinks over the letter G of GOTO

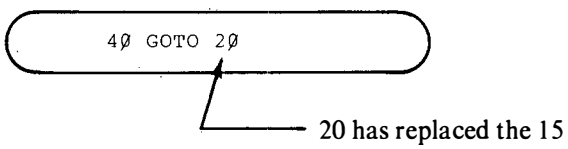
Press the ▷ key again.



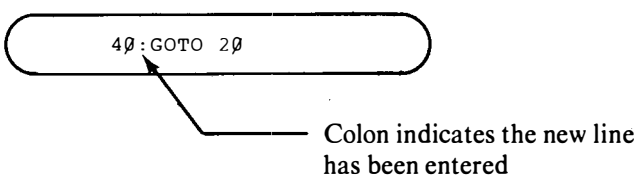
The number 15 must be changed to 20. Therefore, press the 2 key.



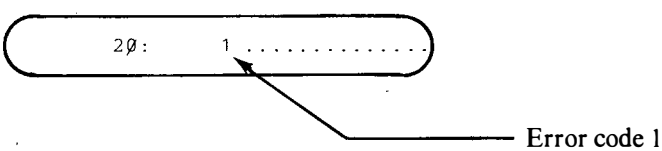
Now press the zero key 0.



Press the ENTER key to complete the entry of your new line 40.



Big George goes back to the RUN mode and tries again. After the program runs awhile, it suddenly stops again.



He presses the [ ] key.

```
20:IF T<THEN PRINT T;
```

Blinking cursor over the P in PRINT

> Error code 1 indicates one of several things:
> a) A grammatical (or syntax) error
> b) Operational error (division by zero, absolute value of result is greater than 1×10^{100} etc.)
> c) Error in memory specification (numerical value assigned to a string variable, or vice versa)

George's error appears to be a grammatical error since it doesn't fit b or c. There must be something wrong with the syntax of THEN PRINT. By closely reading your *TRS-80 Pocket Computer Manual* (page 61), you will probably come to the conclusion that THEN has the same function in an IF statement as GOTO. In other words, if you use IF-THEN, the computer expects a *line number* to follow the word THEN. Page 60 of the manual indicates that George should use IF-PRINT, *not* the word THEN.

Big George should delete the word THEN in line 20. He goes to the PROgram mode and LISTs line 20.

```
20:IF T<24THEN PRINT T;
```

He presses [▷] once,

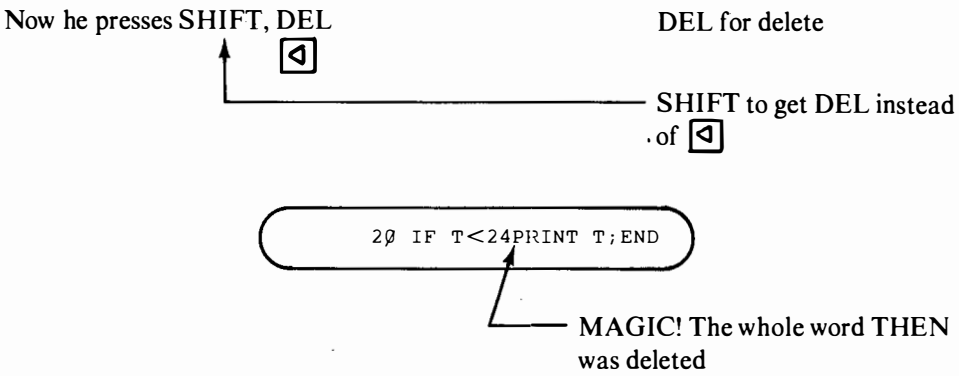
```
20 IF T<24THEN PRINT T;
```

[▷] again,

```
20 IF T<24THEN PRINT T;
```

[▷] several more times until the blinking cursor is over the letter T of THEN.

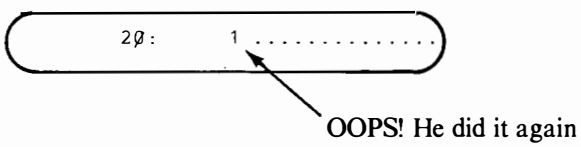
```
20 IF T<24THEN PRINT T;
```



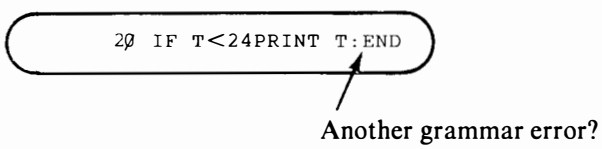
By pressing ENTER the altered line will be entered.

```
2Ø;IF T<24PRINT T:END
```

Big George RUNS again!
The program runs, George waits. Finally . . .



He holds down the [•] key again to see what is wrong this time.



Look at the line carefully. Big George meant line 20 to be a multiple statement line consisting of the two statements:

```
IF T<24 PRINT T
```
and
```
END
```

He can't find anything wrong with either part, so it must be the way that he joined them. Notice what he used to separate the two statements — a *semi*colon. The semicolon is used to separate two variables to be printed. The computer thinks he wants to print T and END. But variables can only be a single letter or a single letter followed by a subscript or $ sign — not three letters. Therefore, the computer is confused and gives a syntax, or grammar, error message. That semicolon should be changed to a colon.

Big George goes to the PROgram mode once again. How many times must he press the ▷ key (after LISTing line 20) to get the blinking cursor over the semicolon? We counted eight times as we watch over his shoulder. Then he presses SHIFT, I to change it to a colon.

```
20 IF T<24PRINT T:END
```

He then presses ENTER.

```
20:IF T<24PRINT T:END
```

Big George runs once more: He waits awhile, then

```
17.
```

Success maybe?

But how does he know if he's right? Little Fred comes to the rescue. He leaves the computer in the RUN mode and types:

```
T=1001
```

Then he presses ENTER and types

```
T-INT(T/24)*24
```

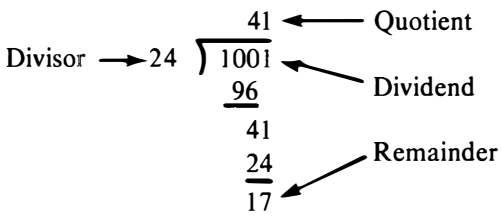
and presses ENTER.

```
17.
```

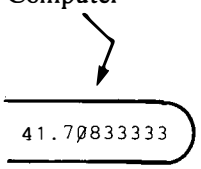
Same result

What does Little Fred know that Big George doesn't? His result sure came a lot faster than Big George's.

Well, Little Fred remembered what he learned in school. When you divide one integer by another, you get a quotient (Q) and a remainder (R).



By the Pocket Computer



Little Fred reasons that if he subtracted 24 times the integer part (41) of the quotient, from the original number of subroutine calls, the difference would be the remainder. The remainder would be the time the clock would show after it makes all of the complete revolutions.

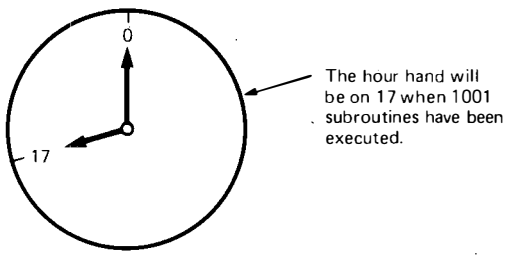
41 complete revolutions of the clock. During that time 41 × 24 (or 984) calls were made to the subroutine.

1001/24 = 41.70833333

41 = INT(1001/24)

INT(1001/24)*24 = 984

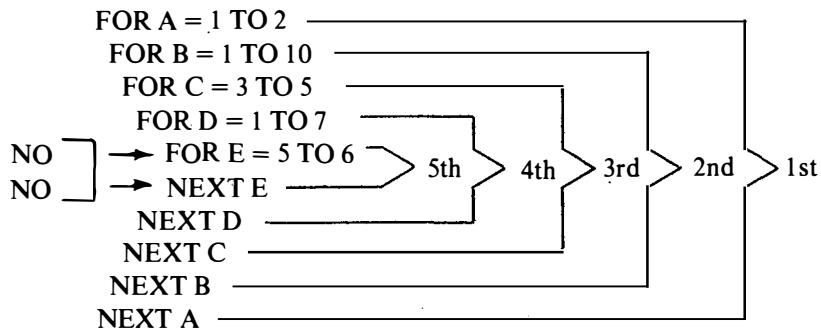
1001−984 = 17 remainder



Even though Big George didn't do a very good job on his original program, he *learned* something about error codes and the editing features of his Pocket Computer.

Other error codes that Big George, and you, should learn are:

- Error code 3 is called a *level error.*
  a) It occurs when you *exceed 4* levels of a GOSUB or FOR-NEXT statement.



  b) It occurs when you try to execute a RETURN without a previous GOSUB.
  c) It occurs when you try to execute a NEXT without a previous FOR.
- Error code 4 indicates that you have insufficient memory for either a program step, reserve memory, or a dimensioned memory.
- Error code 5 indicates a control error when using cassette tape (verify error, check sum error, etc.).
- Error code 6 is a PRINT or PAUSE format error. It occurs when a display of numerical data is not in the format that has been specified.

Editing Features

- Correcting: Use the ◁ or ▷ keys to move the cursor to the position where the correction is needed. Then enter the correction.
- Inserting: Use the ◁ or ▷ keys to move the cursor to the position where you want the insertion to be made. Then press SHIFT ▷ to insert the character desired. The contents of the position under the cursor and all after it will be shifted one place to the right. Press SHIFT and ▷ before each successive insertion if there is more than one to be made.
- Deleting: Use the ◁ or ▷ keys to move the cursor to the position you want to delete. Then press SHIFT ◁ to delete that character, numeral, or statement. The contents of that position will be deleted and all following will be shifted to the left.

Using the Cassette Recorder

The optional cassette interface (Radio Shack Catalog Number 26–3503) makes the Pocket Computer an even more useful tool; programs may be SAVEd permanently for future use. One of its practical uses is to save frequently used subroutines. The subroutines can then be added from the cassette to any program that you wish. The Minisette–9 Cassette Recorder (Radio Shack Catalog Number 14–1812) is an excellent companion to the Pocket Computer because it is both portable and operates on batteries.

One subroutine that you will use frequently is a random number generator. Random numbers are used for the occurrence of unpredictable events in recreational or business simulations. Such unpredictable events can be programmed to occur randomly during program execution.

Some computers have such a function built in, but due to the size of the Pocket Computer, this feature was not incorporated. However, a short subroutine can be written to generate numbers randomly. There is such a program on page 112 of the *TRS-80 Pocket Computer Software Manual*. We have modified that program to fit our particular needs at this time, starting it with a high line number (900) that wouldn't interfere with lower numbered programs.

Random Number Subroutine

```
900 ''A'':INPUT'' INITIAL VALUE (1-9)?'';Z
910 X=ABS(439147+X+Z)
920 Y=E2:1
930 W=23*X
940 X=W-INT(W/Y)*Y
950 IF X=0 THEN 910
960 RETURN
```

E is the Exp key

> WARNING! The subroutine uses variables W,X,Y, and Z. If you use any of these variables in your main program, the subroutine will alter your values.

When the computer returns from the subroutine, a random number will be assigned to X. Use it in any way you wish. In line 920, Y is assigned to the value E2 + 1. That's just another way to express $10^2 + 1$, or 101. Notice line 940 of the subroutine. Doesn't that look like Little Fred's check of the 24-hour clock subroutine call problem? It sure does. The random number generator is using the remainder of a division to get its random number.

Save the Subroutine on Cassette

If you have the optional cassette interface and a recorder, you will want to save the subroutine on tape.

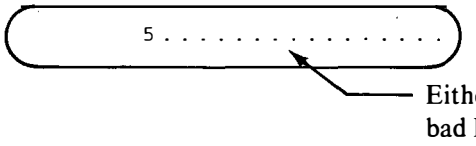
1. Enter the subroutine in the Pocket Computer. A program may be saved in the DEF, RUN, or PRO mode. You can leave the computer in the PRO mode used to enter the program.

2. Connect the cassette interface and recorder as described in the Appendix of your *TRS-80 Pocket Computer Software Manual.*
3. Insert a new, blank cassette in the recorder and rewind it. If it has a clear leader on it, wind the tape past the clear leader so that data can be recorded.
4. Note the index setting of the recorder. Set the controls of the recorder so that it is ready to record. Then type CSAVE "RANDOM" and press ENTER. RANDOM will be the name of the cassette file which contains the subroutine.

As the program is being recorded, you may notice that it goes to the RUN mode. When the program has been completely recorded, the cassette stops and the computer returns to the PRO mode. You should now check to see if the program was successfully loaded.

5. Reset the cassette tape to the beginning of the previous recording. Press the PLAY button on the recorder.
6. Type CLOAD?"RANDOM" and press ENTER.

> CLOAD? compares the information on the tape to the information in memory. If they are not the same, error code 5 will be displayed.

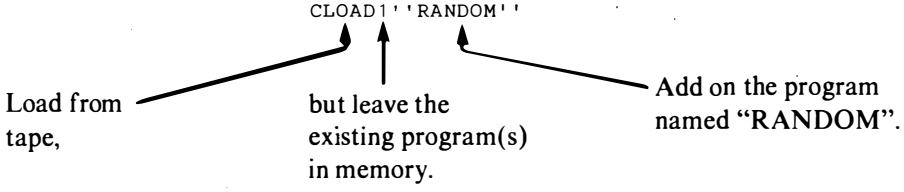


Reset the tape to the beginning of the recording and try again. You may have to experiment with the volume control of your recorder.

When you have a successful CLOAD? the computer will return to the PRO mode and display the usual prompt [>].

LOAD the Subroutine from Cassette

In the future, you will want to add this subroutine to other programs. Ordinarily, when you LOAD a program from tape, all programs currently in the computer are erased. The Pocket Computer has a feature to avoid this. The command:



We will use this command to load the subroutine *after* we enter the main program. The main program calls the subroutine, prints the resulting random number, and then calls the subroutine again. This process repeats until you decide to stop it by pressing the ON key to break into the program.

Main Program – Subroutine Call

```
1Ø GOSUB ''A''
2Ø PRINT X
3Ø GOTO 1Ø
```

The subroutine is labeled "A"

Type NEW and enter the three-line, main program. Then set the controls of your recorder to PLAY and the tape to the beginning of the subroutine that you saved. Load the subroutine with the CLOAD1 command.

```
CLOAD1''RANDOM''
```

Press ENTER. When the subroutine is loaded, LIST it line by line.

You Type.	Computer Displays
LIST, ENTER	`1Ø:GOSUB ''A''`
	`2Ø:PRINT X`
	`3Ø:GOTO 1Ø`
	`9ØØ:''A'':INPUT ''INITIAL V`
	.
.	.
.	etc.
etc.	
	`96Ø:RETURN`

Remember, the display only shows 24 characters. However, the rest is in the computer.

It's all there. RUN the program to check a few random numbers. Fill in the chart in Problem 2. The computer will ask you to input an initial value each time that the subroutine is called. We used 5 for our input each time. Use any number you wish. (The answer is at the end of the chapter.)

Initial Value		Random Number	
We	You	Ours	Yours
5		50	
5		30	
5		75	
5		100	
5		69	
5		63	
5		26	
5		84	
5		4	
5		83	
5		82	
5		59	
5		35	
5		89	
5		18	

Problem 2. Random Number Chart

Applying the Random Number Generator

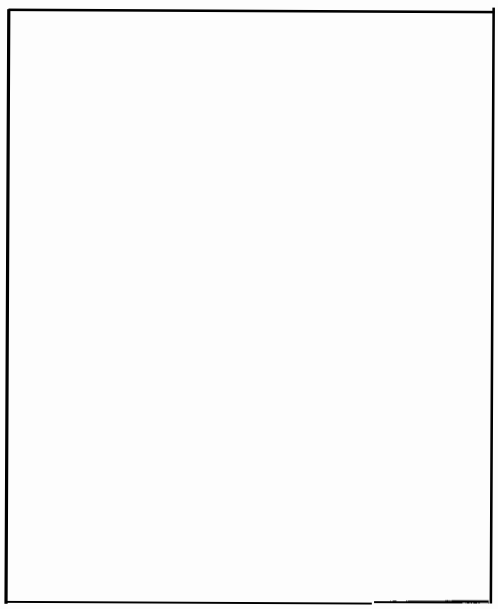
The Popup Toy Company, which manufactures ten different kinds of toys, has recently automated their factory. As a toy comes off the production line it is placed in one of ten different bins. One worker sits at a display console where he has control of a

robot that travels down the line of bins. The worker presses one of ten buttons to tell the robot which of the ten toys to search for. The robot looks in each bin in turn and reports whether that bin contains the particular toy that it is searching for. If the robot doesn't find the toy in the first bin, he reports, "X NOT HERE." (X is the number of the particular toy.) If he finds the toy, he reports, "X FOUND IN BIN Y." (Y is the number of the bin where the toy is found.)

The company wants a computer program that will handle the messages. Each toy is numbered so that the robot can compare the toy's number with the number it is searching for.

Can you write such a program for the Pocket Computer, using the Random Number subroutine to fill the bins with toys? Record your program in the box provided.

Your Toy Program



Problem 3. Toy Program

Here's the program that POPUP TOYS used.

Popup Toys Program

```
10 FOR N=1 TO 10
20 GOSUB ''A''
30 A(N)=X
40 NEXT N
50 INPUT ''PICK A TOY (1-10)'';A(11)
60 FOR N=1 TO 10
70 IF A(N)=A(11) BEEP(1):PRINT A(11);'' IS IN BIN '';N
80 IF A(N)<>A(11) PAUSE A(11);''NOT HERE''
90 NEXT N
100 BEEP(3):GOTO 50
```

Get a random number for each toy

Spaces separate numbers from text

Look in each bin and report

Try their program as well as your own. After you enter their program, CLOAD1 the Random Number subroutine. Change line 920 to:

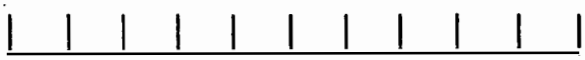
```
920 Y=INT(10*RND(1))+1
```

This will keep the random numbers in the range of 1 through 10.

When you run the program, the Random Number subroutine loads the bins randomly with toys numbered from 1 through 10. Remember, you must type in an initial value each time to "seed" the random number generator. The toy's number will not coincide with the number that you type in for the initial value. The computer then displays:

```
PICK A TOY (1-10)_
```

The worker would then type in the number of the toy that he desired. The robot searches the bins from 1 through 10, reporting back "X NOT HERE" or "X IS IN BIN Y," depending on whether he spots that toy or not. Put yourself in the place of the worker at the console. RUN the program and fill in the toy number found in each bin.



Problem 4. TOYS in Bins

Here is a typical result that we obtained after going to the subroutine ten times to load the boxes.

```
PICK A TOY (1-10)1
```

BEEP!

```
1. IS IN BOX 1.
```

Then for nine times in a row as it searches boxes 2 through 10.

```
1. NOT HERE
```

Boxes so far:


```
PICK A TOY (1-1Ø)2
```

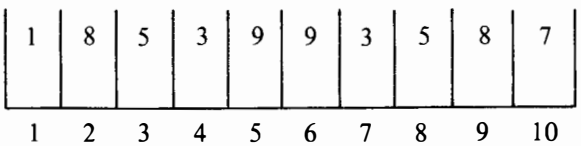
No BEEPS this time. Just ten times in a row:

```
2. NOT HERE

PICK A TOY (1-1Ø)3
```

BEEPS at boxes 4 and 7.

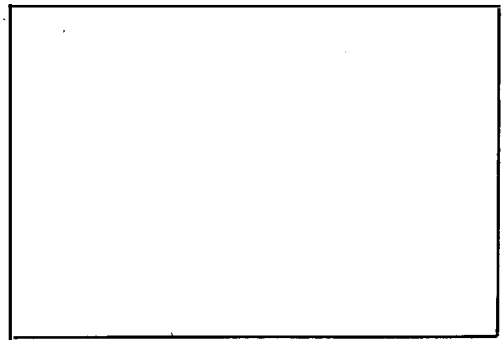
Boxes so far:

We kept going until we had all boxes full. Our distribution looked like this:



Notice that the robot must look in each bin, even if it has already found a toy in one or more of the bins. Can you modify our program so that the robot will pass by the bins where he has previously reported a toy found? (Answers are at end of chapter.)

Your Modified Toy Program



Problem 5. Popup Toys Modification

Summing Up Chapter Three

New Pocket Computer BASIC Statements and Functions

Statement or Function	Accepted Abbreviations	Remarks
INT	—	Find the largest integer contained in the specified expression.
ABS	AB.	Take the absolute value of the specified expression.
GOSUB	GOS. GOSU.	Go to the subroutine starting at the specified line or label.
RETURN	RE. RET. RETU. RETUR.	Return from the subroutine to the main program.
CSAVE	CS. CSA. CSAV.	Record a program from memory onto a cassette tape.
CLOAD	CLO. CLOA.	Load a program from cassette into memory (other programs in memory are erased).
CLOAD?	CLO.? CLOA.?	Check contents of program in memory with those placed on tape by previous CSAVE.
CLOAD1	CLO.1 CLOA.1	Load a program from cassette into memory (adds it to program(s) already there).

Note: Periods in abbreviations must be included.

Error Codes

Number	Cause
1	Syntax error Operating error Error in memory specification
2	Line number error
3	Level error (GOSUB or FOR-NEXT)
4	Insufficient memory
5	Control error of magnetic tape
6	Error in PRINT or PAUSE format

Editing Features

Correcting: [▷] or [◁] key used to move the cursor to the position where correction is to be made.

Inserting: Same keys used as in correcting to position cursor. SHIFT, [▷] INS for INSERT keys used before inserting a new character.

Deleting: Same keys as for correcting used to position cursor. SHIFT, [◁] DEL for DELETE keys used before deleting a character, numeral, or statement.

Other New Items:

- Exp key is used to raise 10 to the specified power. The symbol displayed is E.
- A Random Number generator was given for future use.
- Subroutines were introduced using GOSUB and RETURN.
- The IF-THEN statement requires a line number following the word THEN. For other uses of an IF statement, a direct statement such as PRINT, PAUSE, or LET should follow the specified condition.

Chapter Three Self-Test

1. The Popup Toy Company needs a program to display the inventory of each of its ten toys after monthly sales are reported. The inventory must be known at the beginning of each month. The monthly sales for each toy is input and the resulting inventory is displayed. Write the program to obtain the original inventory from the Random Number subroutine (values 1 through 1,000) for each toy. The monthly sales of each toy must be input from the keyboard. The ending inventory is calculated and displayed.

HINT: Line 920 of the subroutine must be changed to give values I through 1,000.

Your Program (including the subroutine)

2. Fill in the following chart when you run your program.

Toy #	Starting Inventory	Monthly Sales	Ending Inventory
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

3. Add a section to your (or our) program for Problem 1 so that the month's production of toys can be added to the ending inventory. Then have the program repeat itself for a full year. The first month's inventory is random, but then Popup can build more toys.

Your Additions or changes

Answers To Chapter Three Self-Test

1.

```
1Ø FOR N=1 TO 1Ø
2Ø GOSUB ''A''
3Ø A(N)=X
4Ø NEXT N
1ØØ REM * DISPLAY AND INPUT *
11Ø FOR N=1 TO 1Ø
12Ø PRINT''TOY #'';N;'' '';A(N)
13Ø INPUT ''MONTHLY SALES ='';S
14Ø A(N)=A(N)-S
15Ø IF A(N)<Ø PRINT''NOT ENOUGH INVENTORY'':A(N)=A(N)+S:GOTO 12Ø
16Ø NEXT N
17Ø BEEP(3)
18Ø REM * DISPLAY NEW INVENTORY *
19Ø FOR N=1 TO 1Ø
2ØØ PRINT''TOY #'';N;'' '';A(N)
21Ø NEXT N
22Ø END
```

2. Answers will vary, but here is ours.

Toy #	Starting Inventory	Monthly Sales	Ending Inventory
1	514	26	488
2	285	85	200
3	978	155	823
4	878	413	465
5	511	450	61
6	193	103	90
7	818	500	318
8	155	140	15
9	36	10	26
10	233	133	100

3. Additions or Changes

```
1Ø5 FOR M=1 TO 12
1Ø7 PAUSE ''MONTH #'';M
2Ø5 INPUT ''NEW TOYS PRODUCED =?'';T
2Ø7 A(N)=A(N)+T
215 NEXT M
```

A typical two month's run of the program would produce results similar to these:

	Toy #	Starting Inventory	Montly Sales	Ending Inventory	Added Production
First Month	1	852	55	797	0
	2	28	14	14	200
	3	3	3	0	330
	4	383	85	298	200
	5	275	77	198	100
	6	656	500	156	100
	7	502	251	251	0
	8	918	490	428	0
	9	499	88	411	0
	10	964	555	409	0
Second Month	1	797	350	447	0
	2	214	100	114	200
	3	330	133	197	100
	4	498	205	293	0
	5	298	95	203	100
	6	256	190	66	250
	7	251	101	150	150
	8	428	99	329	0
	9	411	100	311	0
	10	409	255	154	150

Answers to Problems in Chapter

1. Subroutine Problem

A general program to solve the problem would look similar to this:

```
10 REM * FIND R FOR T/Q *
20 INPUT ''T='';T
30 INPUT ''Q='';Q
40 PRINT ''REMAINDER= '';T-INT(T/Q)*Q
50 END
```

(50 could be GOTO 20 if there are more than 1 T and/or Q.)

2. Random Numbers

Initial Value		Random Number	
We	You	Ours	Yours
We can't tell what your answers will be, but your random numbers should be in the range of 1 through 100.			

3. Toy Program

   Compare your program with the one on page 65.

4. Toys in Bins

   Again, your answers are random and unpredictable.

5. Popup Toys Modification

Solution 1

In this solution, the robot looks in each bin, but does not bother to report the bins where the desired toy does not exist.

ELIMINATE LINE 80

This cuts down the GOTO loop executions of lines 50 through 100 from about 4 minutes to approximately 2 minutes because the PAUSE statement takes about 0.85 seconds for each execution.

Solution 2

In this solution, a light on the front of the bin is turned on when a toy has been found by an addition to line 70. A new line 65 is added to allow the robot to check the light. The GOTO loop looks like this:

```
5Ø INPUT ''PICK A TOY (1-1Ø)'';A(11)
6Ø FOR N=1 TO 1Ø
65 IF A(N)=Ø THEN 9Ø
7Ø IF A(N)=A(11) BEEP(1):PRINTA(11);'' IS IN BIN '';N:A(N)=Ø
8Ø IF A(N)<>A(11) PAUSE A(11);'' NOT HERE''
9Ø NEXT N
1ØØ BEEP(3):GOTO 5Ø
```

Turn on light when a toy is found.

This program takes approximately 3 minutes to execute the loops.

Solution 3

This solution makes use of a logic statement in line 70 to report which toy is found in a particular bin. It reports only when the toy is found. The GOTO loop:

```
5Ø INPUT ''PICK A TOY (1-1Ø)'';A(N)
6Ø FOR N = 1 TO 1Ø
7Ø Y=(A(N)=A(11))
8Ø IF Y=1 BEEP(1):PRINT ''TOY #'';A(11);'' IS IN BIN '';N
9Ø NEXT N
1ØØ BEEP(3):GOTO 5Ø
```

Don't forget the space to separate numbers from text.

If the statement at line 70 is FALSE, line 80 is *not* executed, and the next A(N) is tested. This solution takes about 2 minutes to complete the search if you are fast in recording the results.

CHAPTER FOUR

Data Files

You learned how to save and load programs in Chapter 3. You worked with an inventory program in the problem section that created and kept track of Popup Toys' supply. However, the data were stored in the computer's memory and were not saved for future use. This is not very practical unless you are not going to use the computer for anything else. But chances are that you are going to want to record a lot of personal and business data once you start using your Pocket Computer.

In this chapter, you will learn:

- how to store a data file on cassette tape;
- how to retrieve a data file from cassette tape;
- how to use a program that:
  a) inputs inventory data from the keyboard or from cassette tape,
  b) allows you to modify the data
  c) allows you to save the resulting data file on tape for future use.

The Popup Toy inventory program introduced in Chapter 3 can be made more useful by revising the program to include saving the inventory data from month to month. Programs are saved by the CSAVE statement, but data is saved with a PRINT # statement in one of two formats.

a) To save a specified memory and all that follow:

```
PRINT # ''DATA 1'';A(5)
```

save on cassette tape	a data file named DATA 1	save the data in memory A(5) and all memories following A(5)

b) To save all data memories in sequence:

```
PRINT # ''DATA 2''
```

save on cassette tape	a data file named DATA 2	no memories specified

These statements may be executed from within a program or in the immediate (manual) mode. Of course you must have your cassette interface and recorder connected as shown in the *TRS-80 Pocket Computer Software Manual*. The recorder

must be turned on and ready to record the data when the PRINT # statement is executed.

If you save data on a cassette tape, you must be able to load the data back into the computer. TRS-80 Pocket Computer BASIC uses the INPUT # statement to do this. It too may be used in two forms.

a) To input data from a specified memory in sequence:

```
INPUT #''DATA 1'';A(5)
```

Input from cassette tape — the data file named DATA 2 — put into memory starting from A(5)

b) To input a whole file:

```
INPUT # ''DATA 2''
```

Input from cassette tape — the data file named DATA 2 — no memory specified

The computer searches the cassette tape for the specified file name. It loads only the data from the named file and ignores data contained in other files with different names. The data are loaded in sequence starting from memory 1 in format b, or from the memory specified in format a.

Cassette Use — Immediate Mode

Use the following program to input data from the keyboard to create a data file.

Load Inventory Program

```
10 REM *INPUT DATA*
40 FOR Y = 1 TO 10
50 PAUSE ''TOY #'';Y
60 INPUT ''HOW MANY?'';A(Y)
70 NEXT Y
```

You'll notice that we skipped some line numbers between 10 and 40. Later we'll add some lines there to expand this section as we develop a more complete inventory program.

Enter the above program and RUN it using the following data:

Toy Number	Input Inventory
1	30
2	40
3	50
4	60
5	70
6	80
7	90
8	100
9	110
10	120

During each pass through the FOR-NEXT loop the computer will flash the toy number briefly and then ask you how many. Wait for it to PRINT:

```
HOW MANY?_
```

before typing in the inventory number. Then press ENTER to move on to the next toy.

After you've finished putting in the data, insert the Pocket Computer in the cassette interface (if it's not already there). Connect the recorder and insert a blank cassette. Saving a data file takes lots of tape space. Don't try to save data on a tape containing other data files or programs unless you are positive that there is enough room. Turn the recorder on and make the necessary settings to prepare it for *recording*.

Type:

```
PRINT #''INVEN.''_
```

Press ENTER

Be prepared to wait awhile. It takes some time for all data memories to be saved.

After the data have been saved on the tape, the recorder stops and the computer displays its ready prompt.

```
>
```

Type CLEAR to clear the computer's memories.

```
CLEAR_
```

Press ENTER.

Just to make sure that the memories have been cleared,

Type

```
A(1)_
```

and press ENTER.

```
Ø.
```

Yes, A(1) has been cleared.

Make sure your recorder is ready to input. Press the PLAY button and type:

```
INPUT #''INVEN.''_
```

and press ENTER.

The data is read from the tape. Then the > prompt appears.

```
>
```

Type A(1) and press ENTER.

```
3Ø.
```

← The same value you entered and saved

Check the other memories — A(2) through A(10) — in the same way. Make sure that they are the same as those used in Memory-Use Chart on page 81.

Cassette Use — Program Mode

When you are sure the data are correct, it's time to write a complete program that Popup Toys can use to keep a record of its inventory.

They need:

1. An input section that will either input data from the keyboard or from cassette tape,
2. A section to enter monthly sales and compute the modified inventory,
3. A section to display the modified inventory and record the month's production, and
4. A section to record the month-ending inventory on tape so that it will be ready to input next month.

Let's look at the program one section at a time.

Popup Toy Inventory Program

1–Input the Data

```
10 REM *INPUT DATA*
20 INPUT ''DATA TAPE (YES OR NO)?'';Z$
30 IF Z$ = ''YES'' THEN 100
40 FOR Y = 1 TO 10
50 PAUSE ''TOY #'';Y
60 PRINT ''HOW MANY?'';A(Y)
70 NEXT Y
80 BEEP(1)
90 GOTO 200

100 REM *READ DATA TAPE*
110 PRINT ''READY YOUR RECORDER''
120 INPUT #''INVEN.''
130 BEEP(2)
```

Skip 40 through 90 if data are from tape

From keyboard

Skip tape section if data are from keyboard

Read data tape

Either lines 40 through 90 or 100 through 130 are used, depending on whether the data to be input are coming from the keyboard or tape.

2 – Enter Sales

```
200 REM *ENTER SALES*
210 FOR Y = 1 TO 10
220 PRINT ''TOY #'';Y;'' '';A(Y)
230 INPUT ''MONTHLY SALES='';S
240 A(Y)=A(Y)-S
250 IF A(Y)<0 PRINT ''NOT ENOUGH INVENTORY'':A(Y)=A(Y)+S:
    GOTO 230
260 NEXT Y
270 BEEP(3)
```

To use this section:

Line 220 prints the toy number and the amount in inventory.

Press ENTER to go on.

Line 230 prints ''MONTHLY SALES?'' and waits for you to type in how many of that particular toy were sold. Enter sales and press ENTER.

Line 240 subtracts the toys sold from the amount on hand to compute the modified amount of that toy on hand.

Line 250 checks to make sure you have enough of the particular toy to make the requested number of sales. If there is not enough, a new number of sales is requested.

Line 270 beeps three times to signal the end of this section.

3 – New Inventory and Production

```
300 REM *DISPLAY NEW INVENTORY AND PRODUCTION*
310 FOR Y = 1 TO 10
320 PRINT'' TOY #'';Y;'' '';A(Y)
330 INPUT ''NEW TOYS PRODUCED='';T
340 A(Y)=A(Y)+T
350 PRINT ''SUPPLY NOW='';A(Y)
360 NEXT Y
```

To use this section:

Line 320 prints the toy number and the supply on hand following the sales of section 2.

Press ENTER to continue.

Line 330 prints NEW TOYS PRODUCED= and waits for you to input the number of new toys.

Input the number of new toys produced.

Press ENTER.

Line 340 computes the resulting inventory.

Line 350 prints the new supply of that particular toy. Press ENTER to continue.

The lines above are repeated for each toy (1 through 10).

4 – Save Data File on Tape

```
400 REM *SAVE DATA FILE*
410 INPUT ''SAVE FILE ON TAPE?'';Z$
420 IF Z$ = ''NO'' THEN 440
430 PRINT #''INVEN.''
440 BEEP(4)
450 END
```

Line 410 allows you to make a choice as to whether or not you want to save the data. If you do, type YES. The data will be saved on tape by line 430. BE SURE YOUR RECORDER IS SET TO RECORD THE DATA. If you type NO, line 420 will cause line 430 to be skipped.

Line 420 checks your response to line 410. If your answer was no, line 430 is skipped.

Line 430 records the data file on tape.

Line 440 beeps four times to signal the end of the program.

Line 450 is the END.

The Memory-Use Chart shows which memories have been used and for what purpose. A tabulation of memory use should be made to ensure that you are not trying to use a memory location for more than one quantity.

Memory	Use	
1	A(1)	Inventory by toy number
2	A(2)	
3	A(3)	
4	A(4)	
5	A(5)	
6	A(6)	
7	A(7)	
8	A(8)	
9	A(9)	
10	A(10)	
11		Unused
12		
13		
14		
15		
16		
17		
18		
19	S	Monthly Sales
20	T	New toys produced
21		Unused
22		
23		
24		
25	Y	Loop counter
26	Z$	Response to question

Memory-Use Chart for Inventory Program

Enter the complete program. You have a data file named INVEN. on cassette tape. Run the program and enter the data from your cassette tape. Then fill in the blanks in Problem 1 as you go along.

Your first response is requested at line 20.

```
DATA TAPE?_
```

Since you have the data on tape, type YES.

The computer will go to line 100 and display:

```
READY YOUR RECORDER
```

Be sure your recorder is ready to PLAY the tape. Then press ENTER. The data will be input from the tape. This will be followed by two beeps to let you know when to go on. Lines 200 through 260 then allow you to enter the sales from Problem 1. When this has been finished, three beeps will sound. The computer then moves to line 300. You will first see:

```
TOY #1. 2Ø.
```

This informs you that there are now 20 toy #1's in stock after the month's sales. Press ENTER to continue. The computer then displays:

```
NEW TOYS PRODUCED=_
```

From Problem 1, type 30

The computer adds this value to the stock on hand and displays:

```
SUPPLY NOW= 5Ø
```

This is the month's ending inventory. Type ENTER to continue. Then the process is repeated for toys 2,3,4,5,6,7,8,9, and 10. Type in the appropriate new production from Problem 1.

Toy #	Input Inven.	Monthly Sales	New Inven.	New Produc.	Ending Inven.
1	30	10		30	
2	40	20		20	
3	50	25		15	
4	60	31		30	
5	70	20		0	
6	80	14		0	
7	90	75		25	
8	100	60		10	
9	110	80		20	
10	120	85		20	

Problem 1. Inventory Program Results
(answers at end of chapter)

When all has been completed, the computer will ask:

```
SAVE FILE ON TAPE?_
```

Type YES to save the ending inventory. *Before* pressing ENTER, *be sure your recorder is ready to record.* Then press ENTER and the data will be saved. Four beeps will be heard after the data is successfully saved. Then the > prompt is shown as the program ends.

```
>
```

Have your data been saved correctly?
To find out:

1) Type CLEAR to clear the existing data memories.
2) Set the cassette back to the beginning of the inventory data file just saved. Set your recorder to PLAY the data back in. Then type `INPUT #''INVEN.''` and press ENTER.
3) After the data are typed in, type A(1) and press ENTER. Fill in the result in Problem 2.
4) Repeat step 3 for each memory A(2) through A(10).

Type	Toy #	Inventory
A(1) ENTER	1	
A(2) ENTER	2	
A(3) ENTER	3	
A(4) ENTER	4	
A(5) ENTER	5	
A(6) ENTER	6	
A(7) ENTER	7	
A(8) ENTER	8	
A(9) ENTER	9	
A(10) ENTER	10	

Problem 2. Inventory Check After Reloading Data
(answers at end of chapter)

Now that you know how to use the Popup Toy Inventory Program, RUN the program and keep a three-month account using the data in Problem 3. You're on your own this time. Fill in the data as the program is executed.

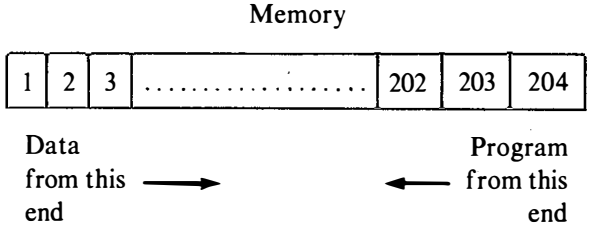
Month	Toy Number	Starting Inventory	Sales	Production	Final Inventory
1	1	50	20	30	
	2	35	15	40	
	3	70	50	60	
	4	83	45	20	
	5	42	22	40	
	6	79	38	20	
	7	66	0	0	
	8	42	28	32	
	9	15	7	50	
	10	100	44	0	
2	1		20	30	
	2		28	20	
	3		52	30	
	4		35	20	
	5		42	30	
	6		28	20	
	7		42	40	
	8		33	30	
	9		20	0	
	10		48	40	
3	1		35	20	
	2		40	50	
	3		22	40	
	4		38	50	
	5		42	50	
	6		22	30	
	7		32	40	
	8		28	40	
	9		22	30	
	10		20	30	

Problem 3. Three-Month Record (answers at end of chapter)

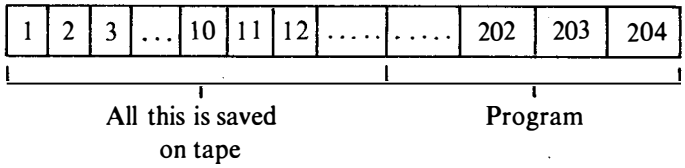
How Data Files Are Recorded

You probably noticed that it took a long time and a lot of tape to record the ten items of the Popup Toys Inventory Program. To make use of cassette tapes efficiently, you should understand and make use of the method used to record data on the TRS-80 Pocket Computer.

Remember, first of all, how the memories in the Pocket Computer are used from both ends.



When data are recorded by the statement: PRINT #"INVEN." the recorder saves the data from all memories from the DATA end up to the end of the flexible program memory used.

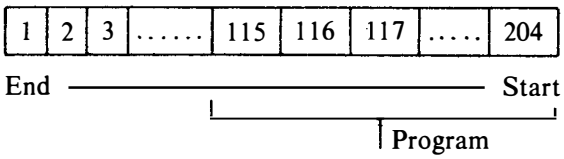


In other words, the shorter your program is, the more data are saved by the PRINT # statement. All memories are recorded from A(1) up to the end of the program, even though the memories may not contain data. A zero is recorded as the contents of memories not in use.

If you use memories A(1) through A(10) and the PRINT # format that records all memories, much tape is wasted. If you knew where your program ended, you could choose subscripts closer to the end of program memory, and thus save the total amount of tape used to save the file.

Example:

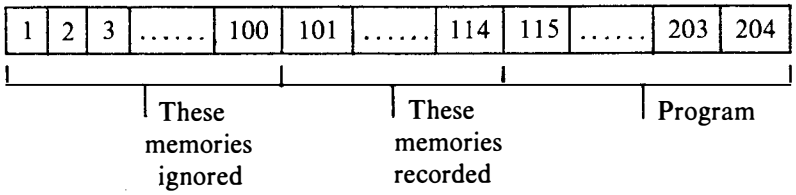
Suppose the program ran to memory number 115.



You might choose A(101) through A(110) for your data storage. Then use the print format:

```
PRINT #''INVEN.'';A(101)
```

This would record only memories 101 through 110, thus saving the tape normally used to record memories 1 through 100.

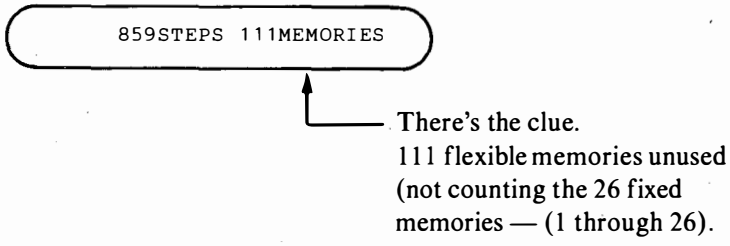


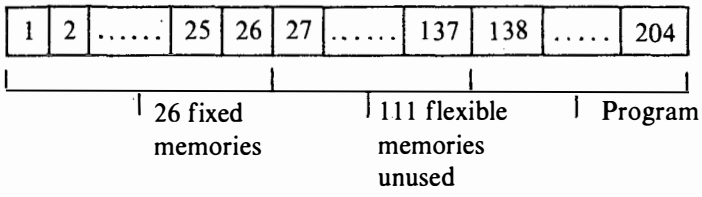
Now, the problem is how do you find out where the program ends in memory? Remember the MEM command? It will give you a clue. It tells you how many flexible memories are free for your use. Here's what we did.

The Popup Inventory Program is in the computer.

We typed MEM.

When we pressed ENTER, we saw:





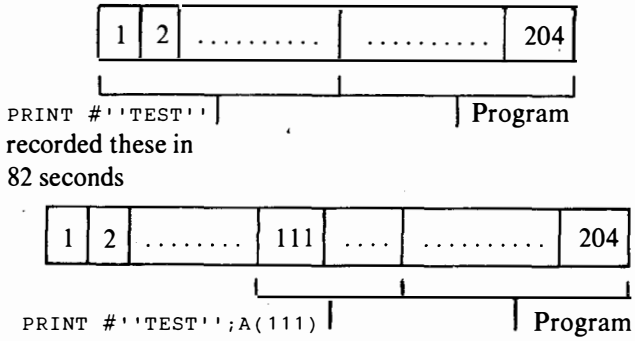
It looks like we could use A(121) through A(130) for our toy counters instead of A(1) through A(10). However, we will drop down 10 more memories to be saved and use A(111) through A(120). Much less tape will then be used for recording the values.

Just to get a feeling of the time saved in this approach, we disconnected the recorder and tested:

1. `PRINT #''TEST''`, and
2. `PRINT #''TEST'';A(111)`

Here are their recording times:

`PRINT #''TEST''`	82 seconds
`PRINT #''TEST'';A(111)`	16 seconds



The second method used 16/82 or .1951219512 as much tape as the first method (less than 1/5 as much). It would be to your advantage to change the Popup Toys Inventory Program to incorporate a change in the use of memories. See if you can change the following Popup lines in Problem 4 so that you can use memories 111 through 120 for holding the toy inventory.

Hint: Change FOR-NEXT loop start and end values. Add a new variable X = Y − 110.

Line #	New (or Modified) Line
40	
45	
50	
120	
210	
215	
220	
310	
315	
320	
430	

Problem 4. Modifications to Popup Toys Inventory Program

Summing Up Chapter Four

New Pocket Computer BASIC Statements

Statement	Accepted Abbreviation	Remarks
CLEAR	CL. CLE. CLEA.	Clears all data memories.
INPUT #	I. # IN. # INP. # INPU. #	Transfers data from cassette tape to data memories.
PRINT #	P. # PR. # PRI. # PRIN. #	Transfers data from computer's memories to cassette tape.

Other New Items:

- PRINT # can be used in two formats:
  a) `PRINT # ''NAME''` — Records all data memories onto cassette tape.
  b) `PRINT # ''NAME'';A(111)` — Records all data memories from 111 upward onto cassette tape.
- INPUT # can be used in two formats:
  a) `INPUT # ''NAME''` — Transfers data from cassette to computer's data memories in sequence beginning with memory #1.
  b) `INPUT# ''NAME'';A(111)` — Transfers data from cassette to computer's data memories in sequence beginning with memory #111.
- The amount of tape used to record a data file depends upon the PRINT # format used.
- PRINT # and INPUT # can be used from either the manual mode (DEF or RUN) or from within a program.
- After all the practice in this chapter, you should now be proficient in the use of the cassette recorder for data files.

Chapter Four Self-Test

1. You have a list of ten people that you frequently call on the telephone. Write a program that will input their last names and seven-digit telephone numbers from the keyboard. Include a section that will save the names and phone numbers in a cassette data file.

2. Write a program to input the cassette data file of Problem 1 and display the names and corresponding phone numbers.

3. Write a program to search the file created in Problem 1 and print only the name and phone number of the name that is input.

4. Write a program similar to the solution to Problem 3. Only this time, input the phone number desired and print the name and phone number.

5. Add a section to your (or to our) solution to Problem 2 that will let you search for a name, delete that name and phone number, and then move all names that follow the deleted name up one position.

Answers to Chapter Four Self-Test

1. Your variables, FOR-NEXT parameters, line numbers, format, etc. may be different from our solution. Ours is merely given as a guide.

```
10 FOR Z=101 TO 110:Y=Z+10
20   INPUT ''NAME?'';A$(Z)
30   INPUT ''PHONE NUMBER?'';A(Y)
40 NEXT Z
50 PRINT #''PHONE #'';A(101)
```

2. Again, your program may be different. Try yours and ours with the file created in Problem 1 to see if they work.

```
10 INPUT #''PHONE #'';A(101)
20 FOR Z=101 TO 110:Y=Z+10
30   PRINT A$(Z),A(Y)
40 NEXT Z
```

3. We are using the phone file data from Problem 1.

```
10 INPUT #''PHONE #'';A(101)
20 INPUT ''NAME DESIRED?'';X$
30 FOR Z=101 TO 110 Y=Z+10
40   IF A$(Z)=X$ PRINT A$(Z),A(Y)
50 NEXT Z
```

4.

```
10 INPUT #''PHONE #'';A(101)
20 INPUT ''PHONE # DESIRED?'';P
30 FOR Z=101 TO 110:Y=Z+10
40  IF A(Y)=P PRINT A$(Z),A(Y)
50 NEXT Z
```

5. This is the way we did it:

```
10 INPUT #''PHONE #'';A(101)
20 FOR Z=101 TO 110:Y=Z+10
30   PRINT A$(Z),A(Y)
40 NEXT Z
```

From Problem 2, you can examine the names to see which to delete from the printout (line 30).

```
100 INPUT ''NAME TO BE DELETED?'';X$
110 FOR Z= 101 TO 110
120   IF A$(Z=X$ GOSUB 200
130 NEXT Z
140 END
200 W=Z
210 FOR Z=W TO 109:V=Z+1:U= Z+10:T=Z+11
220   A$(Z)=A$(V):A(U)=A(T):A$(V)='' '':A(T)=0
230 NEXT Z
240 Z=Z+1:RETURN
```

Type name

Search for the name

Move all names, and numbers below the one to be deleted, up one position.

When you run the program, lines 10 through 40 allow you to look at the names and numbers in the file. Lines 100 through 130 conduct the search for the name that you want to delete. The subroutine (lines 200 through 240) deletes the requested name, if found, and moves all the names up one position. The last position is blanked out.

Here are the names and numbers in our phone file.

ABLE 1111111
BAKER 2222222
CARTER 3333333
DOUGLY 4444444
EASY 5555555
FARMER 6666666
GOODE 7777777
HARRY 8888888
INMAN 9999999
JONES 1212121

This is what a typical run looks like from line 100 onward.

```
NAME TO BE DELETED?_
```

We typed FARMER

```
>
```

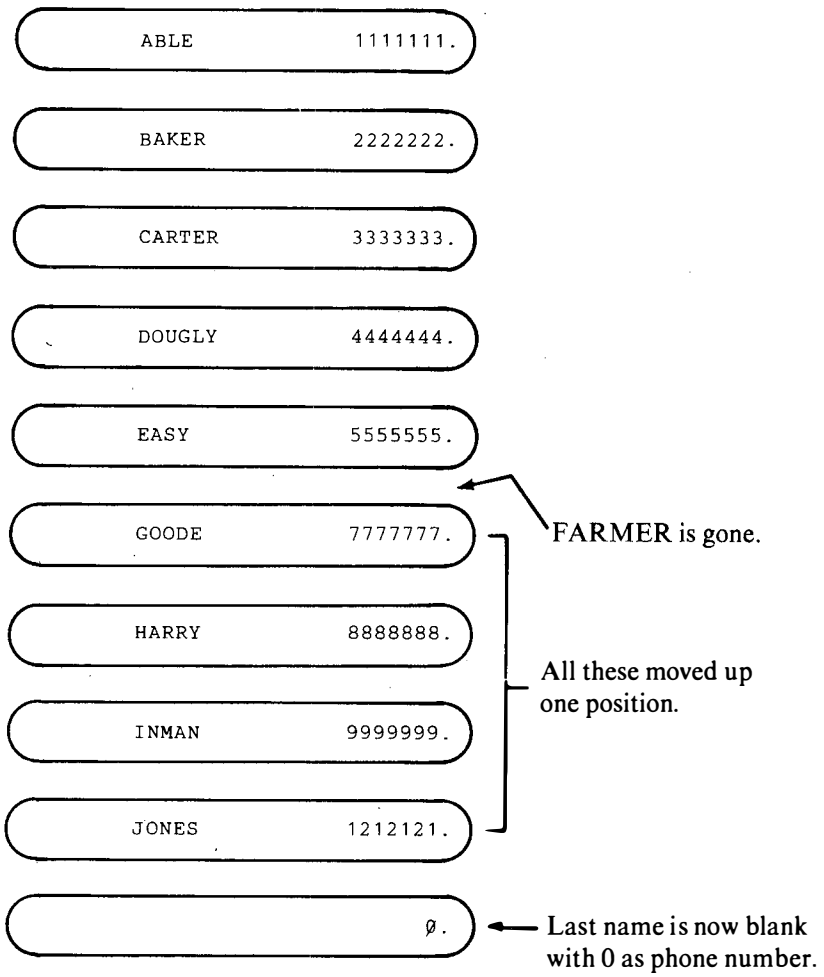
The END of the program

To see whether the name had been deleted, we typed:

```
RUN 20
```

(so that the original file was not input again)

In order, we saw the file that was left.



Answers to Problems in Chapter

1. Inventory Program Results

Toy #	Input Inven.	Monthly Sales	New Inven.	New Produc.	Ending Inven.
1	30	10	20	30	50
2	40	20	20	20	40
3	50	25	25	15	40
4	60	31	19	30	59
5	70	20	50	0	50
6	80	14	66	0	66
7	90	75	15	25	40
8	100	60	40	10	50
9	110	80	30	20	50
10	120	85	35	20	55

2. Inventory Check After Reloading Data

Toy #	Inventory
1	50
2	40
3	40
4	59
5	50
6	66
7	40
8	50
9	50
10	55

3. Three-Month Record

Month	Toy Number	Starting Inven.	Sales	Produc.	Final Inven.
1	1	50	20	30	60
	2	35	15	40	60
	3	70	50	60	80
	4	83	45	20	58
	5	42	22	40	60
	6	79	38	20	61
	7	66	0	0	66

Month	Toy Number	Starting Inven.	Sales	Produc.	Final Inven.
	8	42	28	32	46
	9	15	7	50	58
	10	100	44	0	56
2	1	60	20	30	70
	2	60	28	20	52
	3	80	52	30	58
	4	58	35	20	43
	5	60	42	30	48
	6	61	28	20	53
	7	66	42	40	64
	8	46	33	30	43
	9	58	20	0	38
	10	56	48	40	48
3	1	70	35	20	55
	2	52	40	50	62
	3	58	22	40	76
	4	43	38	50	55
	5	48	42	50	56
	6	53	22	30	61
	7	64	32	40	72
	8	43	28	40	55
	9	38	22	30	46
	10	48	20	30	58

4. Modification to Popup Toys' Inventory Program

Line #	New (or Modified) Line
40	`FOR Y=111 TO 12Ø`
45	`  X=Y-11Ø`
50	`  PAUSE ''TOY #'';X`
120	`INPUT #''INVEN.'';A(111)`
210	`FOR Y=111 TO 12Ø`
215	`  X=Y-11Ø`
220	`  PRINT ''TOY #'';X;'' '';A(Y)`
310	`FOR X=Y-11Ø`
315	`  X=Y-11Ø`
320	`  PRINT ''TOY #'';X;'' '';A(Y)`
430	`PRINT #''INVEN.'';A(111)`

CHAPTER FIVE

Trigonometric Functions

The TRS-80 Pocket Computer has many functions that are built into its BASIC. Some of them that you have already seen are INT and ABS (Chapter 3). In this chapter, you will be working with the Pocket Computer's trigonometric functions. You will learn to solve geometric problems using right triangles. The sides and the angles of a right triangle have special relationships that form trigonometric functions. Those available on the TRS-80 Pocket Computer are:

- sine
- cosine
- tangent
- arcsin
- arccos
- arctan

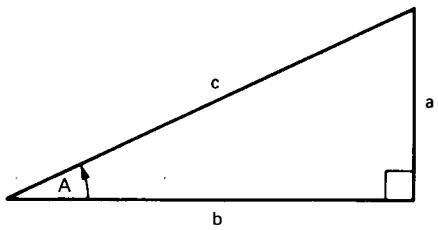
You have learned the major features of the TRS-80 Pocket Computer in the first four chapters. Therefore, Chapter 5 and those that follow are largely devoted to solving a wide variety of problems. Those encountered in Chapter 5 involve:

- A boat crossing a stream
- A dog walking in circles
- A ship passing a lighthouse
- The height of a tall building
- Surveying measurements
- Area measurements
- Measuring an immeasurable lake
- Fourier series

The solutions of all these problems are reached through the trigonometric functions of the Pocket Computer, plus a little brain work on your part.

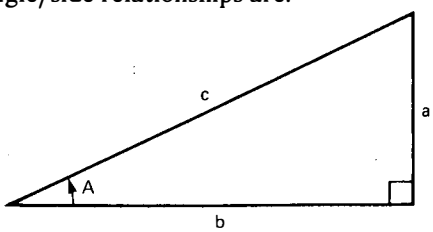
Angle Measurement Systems

A right triangle is composed of the following parts:



A = angle
a = side opposite A
b = side adjacent to A
(c is also adjacent to A, but it has a special name.)
c = hypotenuse

The TRS-80 Pocket Computer is ideal for solving unknown parts of right triangles. Some of the angle/side relationships are:

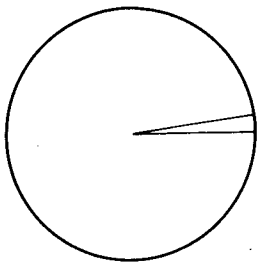


$c^2 = a^2 + b^2$

1. Degree – an angular measurement based on the division of a circle into 360 equal angles — each being one degree.

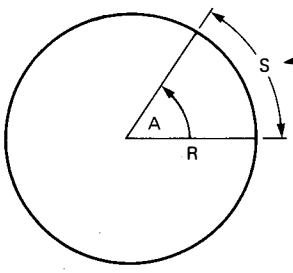
$\sin A = a/c$	$A = \sin^{-1}(a/c)$
$\cos A = b/c$	$A = \cos^{-1}(b/c)$
$\tan A = a/b$	$A = \tan^{-1}(a/b)$

The Pocket Computer can work with three different angle-measurement systems.



1 degree = 1/360 of a circle

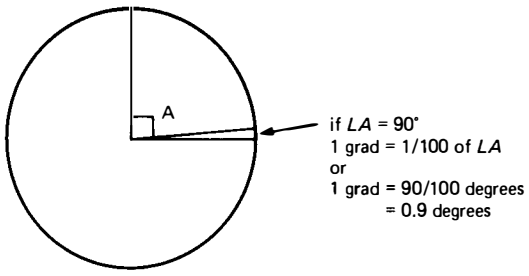
2. Radians – an angular measurement defined as the angle formed when the length of the radius equals the length of the subtended arc.



when are S = radius R,
$\angle A$ = 1 radian.

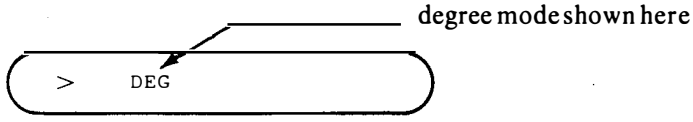
There are 2 radians in a circle.
There are 360 degrees in a circle.
Therefore, 1 radian = $\frac{360}{2}$ degrees, or
1 radian $\approx$ 57.2958 degrees.

3. Grad – 1/100 of a right angle.

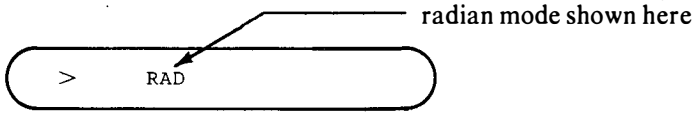


You may choose the angular measurement system that you wish to work in.

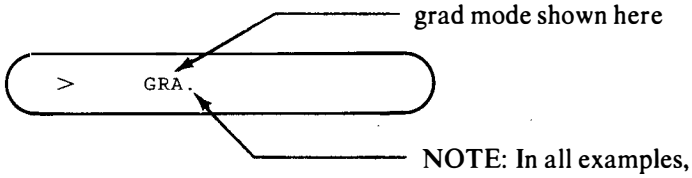
1. To designate the degree mode type:
   DEG.
   DEGR.
   DEGRE.
   or DEGREE



2. To designate the radian mode type:
   RA.
   RAD.
   RADI.
   RADIA.
   or RADIAN

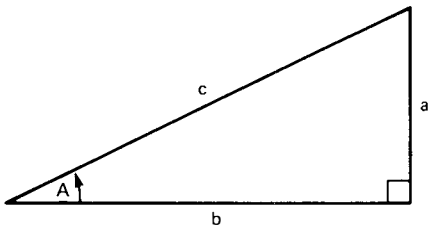


3. To designate the grad mode type:
   GR.
   GRA.
   or GRAD



We will usually be operating in the degree mode and will not show the angle mode on the display unless we change modes.

The Pocket Computer has statements for sin, cos, tan, arcsin ($\sin^{-1}$), arccos ($\cos^{-1}$), and arctan ($\tan^{-1}$) to aid in the solution of trigonometric problems.



- sin (sine of an angle) is the ratio of the side opposite the angle to the hypotenuse (a/c).
- cos (cosine of an angle) is the ratio of the side adjacent to the angle to the hypotenuse (b/c).
- tan (tangent of an angle) is the ratio of the side opposite the angle to the side adjacent to the angle (b/c).
- arcsin (sin $^{-1}$) is the angle whose sine is specified.
- arccos (cos $^{-1}$) is the angle whose cosine is specified.
- arctan (tan $^{-1}$) is the angle whose tangent is specified.

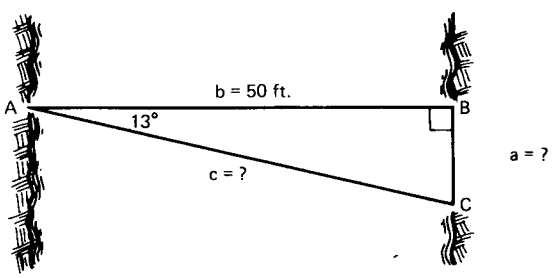
Boat Problem

A girl in a boat is crossing a river 50 feet wide. The planned course runs perpendicular (at a right angle) to the shore from point A to point B, but the current carries her downstream at an angle of 13 degrees.

How far downstream is her destination (C) from the point she was steering for (B)?_____

How far does the boat travel (from A to C)?_____

HINT: In solving problems of this type, always draw a picture first. If you can sketch the conditions given, the problem is half solved.



Use the calculator mode of the Pocket Computer to compute the distances. Write your answers in the box.

$\overline{BC}$ =
$\overline{AC}$ =

Which trig functions did you use:

1. to get $\overline{BC}$? _____
2. to get $\overline{AC}$? _____

Solution to the Boat Problem

To solve a problem of this type, pick out one of the trigonometic function where two parts of the triangle are known. In this problem you know that side b is 50 feet. You also know that angle A is 13 degrees. You want to find side a. Therefore, you would pick the tangent function.

$$\tan 13 \text{ degrees} = a/50 \text{ feet}$$

or

$$a = 50 \times \tan 13 \text{ degrees}$$

On the computer

```
50*TAN13_
```

ENTER

```
11.54340956
```

feet downstream $\overline{BC}$

To find side c, pick the cosine function.

$$\cos 13 \text{ degrees} = 50 \text{ feet}/c, \text{ or } c = 50/\cos 13 \text{ degrees}$$

On the computer

```
50/COS13_
```

ENTER

```
51.31520539
```

feet traveled $\overline{AC}$

Of course, you should check your solutions each time a problem is solved. Check the solutions:

$$c = \sqrt{50^2 + 11.5434^2}$$

On the computer

```
√(50  2+11.5434  2)_
```

ENTER

```
51.31520324
```

Pretty close

Another way to check the solution is by the known angle, 13 degrees. To check your result for side a, use the inverse trigonometric function for known angle A and known side b. That would be the arctan. You can also check the result for c by using the known angle A and the known side b. That would be the arccos.

Angle A = arctan(a/b)
Angle A = arccos(b/c)

computed side
known side

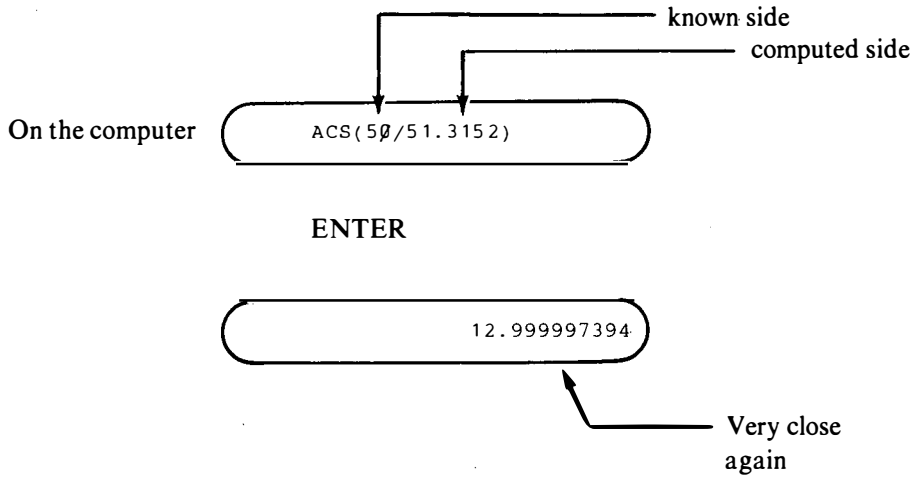
On the computer

```
ATN(11.5434/50)
```

ENTER

```
12.9999896
```

Pretty close
to 13 degrees

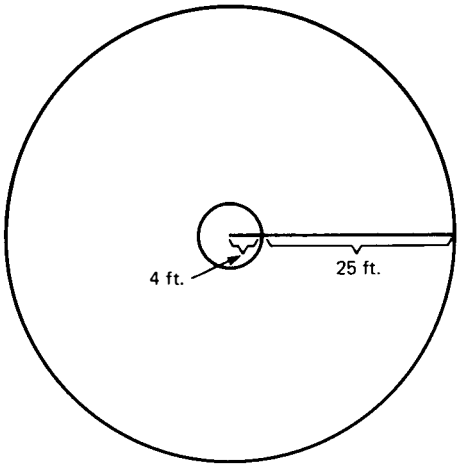


Walking The Dog Problem

A man is walking his dog in a circle. The dog is on a 25-foot rope. The man walks in a circle with a radius of 4 feet. The dog keeps the rope taut as he walks.

After walking through a 125-degree angle, how far has the dog walked? _____ How far has the man walked? _____

HINT: Think the problem through in radians. For each radian, the arc length walked equals the radius. We have drawn the problem for you.



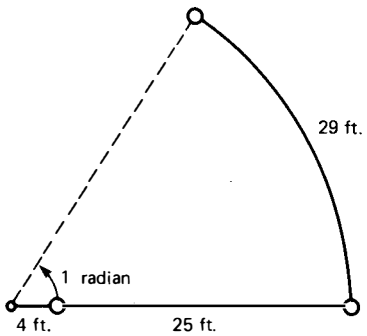
After 20 trips around the circle, how far has the man walked?

_____

The dog? _____

Solution to the Dog Walk Problem

For each radian, the dog walks a distance of 29 feet, the same as the radius of his circle.



Remember, a radian is approximately equal to 57.2958 degrees. Therefore, for 125 degrees, the dog walks (125/57.2938 radians) × (29 feet per radian).

```
125/57.2958*29_
```

ENTER

```
63.26816276
```

The man walks:

```
125/57.2958*4_
```

ENTER

```
8.72664314
```

After 20 trips around the circle:

The man has walked:

```
2*  *4*2Ø_
```

```
5Ø2.6548246
```

The dog has walked:

```
2*  *29*2Ø_
```

```
                3644.247478
```

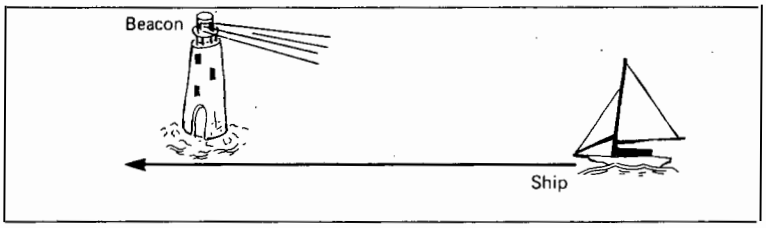
Lighthouse Beacon Problem

A ship at sea sights a lighthouse beacon at an angle of 16 degrees from the ship's heading. The ship maintains the same heading for 20 miles, and then takes a second sight on the lighthouse. This time the angle is 32 degrees.

How much farther will the ship travel before it is directly opposite the lighthouse? _____

How far from the lighthouse will the ship be at that time? _____

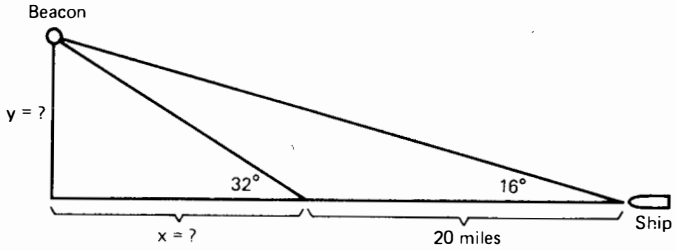
Make a sketch here



HINT: Set up two separate solutions; one for the 16 degree angle and one for the 32 degree angle. Use two variables for the unknowns. Solve the two variables for one unknown, then solve for that variable first. Then work on the second variable.

Solution to the Lighthouse Beacon Problem

Our sketch



$$y/x + 20 = \tan 16 \qquad y/x = \tan 32$$

$$(x + 20)*\tan 16 = x*\tan 32$$

$$x = \frac{-20*\tan 16}{\tan 16 - \tan 32}$$

On the computer

```
-20*TAN16/(TAN 16-TAN32)
```

```
16.96096192
```

about 17 miles to travel

$$y = x*\tan 32$$

On the computer

```
16.96096192*TAN32
```

```
10.59838525
```

about 10.6 miles from beacon

Check of solution

$$y = (x + 20)*\tan 16$$

On the computer

```
(16.96096192+20)*TAN16
```

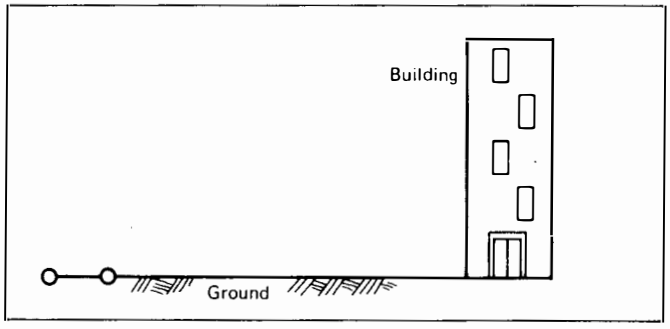
```
10.59838528
```

Very close

Building Height Problem

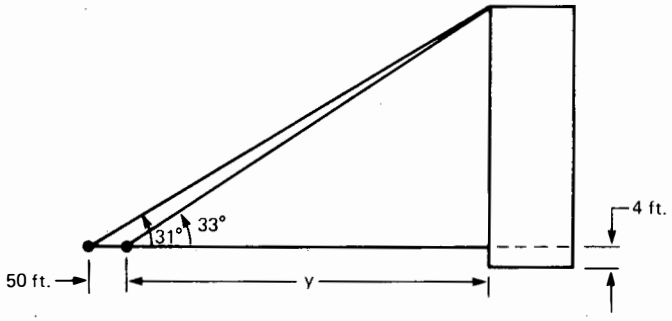
A surveyor on the ground sights the top of a building through an angle of elevation measuring 33 degrees. He then moves back 50 feet and resights the top of the building at 31 degrees. The eyepiece of the transit is 4 feet from the ground.
How high is the building? _____

Draw a sketch here



Solution to the Building Height Problem

Our sketch:



$x/y = \tan 33$ $\qquad$ $x/y + 50 = \tan 31$

$x = y*\tan 33$ $\qquad$ and $\qquad$ $y = \dfrac{50\ \text{TAN}\ 31}{\text{TAN}\ 33 - \text{TAN}\ 31}$

Therefore, $x = \dfrac{50*\text{TAN}\ 31*\text{TAN}\ 33}{\text{TAN}\ 33 - \text{TAN}\ 31}$

On the computer `(50*TAN31*TAN33)/(TAN33 - TAN 31)`
This won't all fit on the display, but after it is typed in you see:

```
31*TAN33)/TAN33-TAN31)_
```

Then, when you press ENTER:

```
401.8823574
```

Building − transit height in feet

401.8823574 + 4 = 405. 8823574

height of building in feet

Check of solution

y = distance from building to first sight
= x/tan33

On the computer

```
401.8823574/TAN33_
618.8445617
```

y = x/tan31 − 50

```
401.8823574/TAN31-50_
618.8445617
```

both the same

Sighting distance = hypotenuse
= x/sin33

```
401.8823574/SIN33
737.8875394
```

hypotenuse = $\sqrt{x^2 + y^2}$

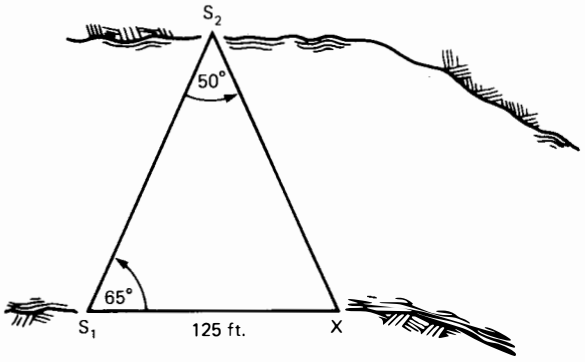
√(401.8823574 2 + 618)8445617 2)

737.8875393

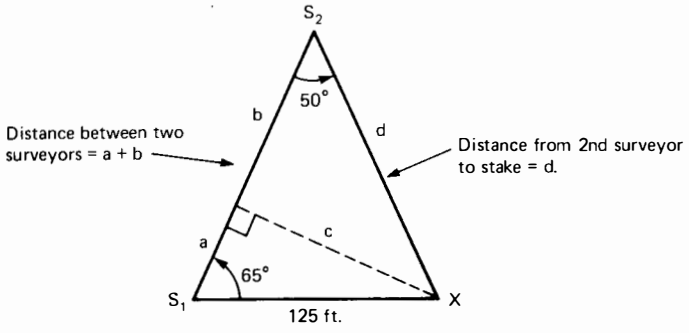
Both methods check.

River Problem

One surveyor, labeled S_1, places a stake on one side of a river. He then walks along the side of the river to a point 125 feet from the stake. He measures an angle of 65 degrees from the original stake to a second surveyor on the other side of the river. The second surveyor (S_2) measures an angle of 50 degrees from the first surveyor to the stake, X.
How far apart are the two surveyors? _____
How far is the second surveyor from the stake? _____



Our solution



A quick solution on the Pocket Computer can be found by arranging the partial solutions in this order:

a/125 = cos65	a = 125*cos65	← find a
↓		
c/a = tan65	c = a*tan65	← then b
↓		
c/b = tan50	b = c/tan50	← then c
↓		
b/d = cos50	d = b/cos50	← then d

On the computer

```
125*COS65
```

```
52.82728272
```
a

Press * and type tan65

```
52.82728272*TAN65
```

```
113.2884734
```
c

Press / and type tan50

```
113.2884734/TAN5Ø
```

```
95.Ø6Ø31625
```
b

Press / and type cos50

```
95.Ø6Ø31625/COS5Ø
```

```
147.887599
```

a + b = 52.83 + 95.06 = 147.89 feet apart (two surveyors)
d = 147.89 feet (second surveyor from stake)

Checks for solutions

$\sqrt{a^2 + c^2 = 125}$

On the computer

```
√(52.83  2+113.29  2)
                125.ØØ2532
```

Very close

$\sqrt{b^2 + c^2 = d = 147.887599}$

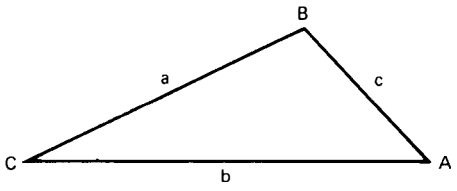
On the computer

```
√(95.Ø6  2+113.29  2)
                147.8885651
```

Close again

Equations for General Triangles

Equations for triangles that have no right angles can be derived from the right triangle angle/side relationships. Such relationships are given following the sketch of this general triangle.



Sides and angles for the general triangle may be found from the group of equations called the Law of Cosines and the Law of Sines.

Law of Cosines

$a = \sqrt{b^2 + c^2 - 2bc \cos A}$

$b = \sqrt{a^2 + c^2 - 2ac \cos B}$

$c = \sqrt{a^2 + b^2 - 2ab \cos C}$

Law of Sines

$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c}$$

Here are three different formulas that may be used to find areas of triangles.

Two angles and a side known:

$$\text{Area} = \frac{a^2 \sin C \sin(180 - (A + C))}{2 \sin A}$$

Two sides and included angle known:

$$\text{Area} = \tfrac{1}{2}ab \sin C$$

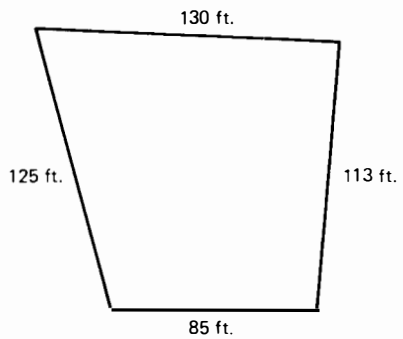
Three sides known:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$
$$\text{where } s = \tfrac{1}{2}(a + b + c)$$

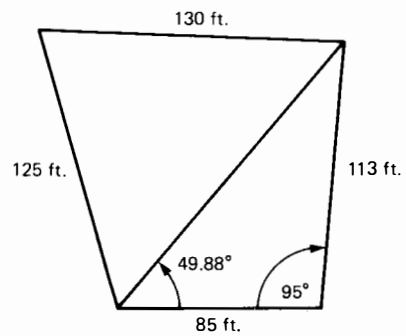
With these equations, you are equipped to solve all kinds of problems involving triangles. Use your Pocket Computer and the equations on the following problems.

The Irregular Lot Problem

My friend, Jennifer, owns an irregular shaped lot. She has a map that shows the boundaries and the length of each of the four sides. She wants to advertise the lot as for sale. However, the wording in her deed is so obscure that she can't make much out of it. This is the shape of her lot.

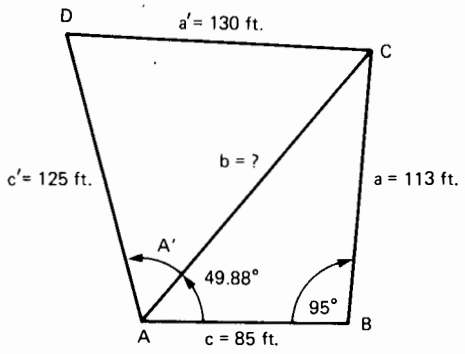


Jennifer put stakes at all four corners of her lot and measured the angles. Can you help her find the area of the lot? Here are the measurements:



Solution to Irregular Lot

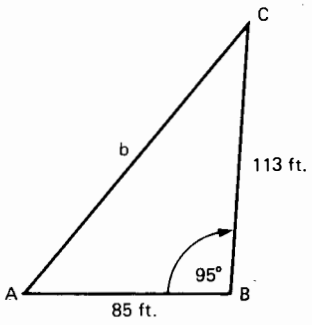
Here are two ways to solve the problem. Maybe you can think of others. In both solutions the lot is divided into two triangles, as shown.



Solution 1

Using the lower right triangle first:

Area of △ABC = ½ac sin B



= ½*113*85*sin 95

Computer shows

```
.5*113*85*SIN95

4784.225038
```

Area of △ABC ≈ 4784 sq. ft.

Use the law of cosines to solve for side b.

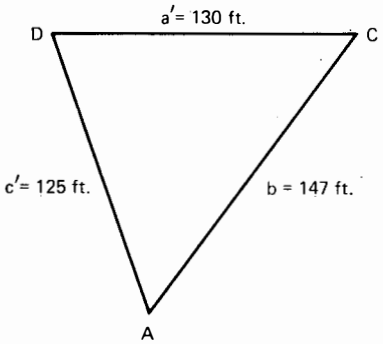
$b = \sqrt{a^2 + c^2 - 2ac \cos B}$

Computer shows

```
√(113  2+85  2-2*113*85*COS95)

                    147.2Ø14328
```

side b ≈ 147 ft.

Then solve for the triangle in the upper left.



Use the three sides to solve for the area of △ACD.

$s = \frac{1}{2}(a' + b + c')$

Area of $\triangle ACD = \sqrt{s(s-a')(s-b)(s-c')}$

Computer shows

```
S=.5*(13Ø+147+125)

                    2Ø1.

(S*(S-13Ø)*(S-147)*(S-125)

                    7652.9853
```

S=201

Area of △ACD ≈ 7653 sq. ft.

Total area of lot = area of △ABC + area of △ACD
≈4784 + 7653
≈12437 sq. ft.

Solution 2

$$\text{Area of } \triangle ABC = \frac{113^2 (\sin 95)(\sin(180-(49.88+95))}{2 \sin 49.88}$$

Computer shows

```
1132*SIN95*SIN(180(-(49.88+95))/(2*SIN49.88)
```

```
4784.86226
```

Area of △ABC
≈4785 sq. ft.

Solving for side b from Law of Sines:

$$\frac{b}{\sin B} = \frac{a}{\sin A}$$

$$b = \frac{a \sin B}{\sin A} = \frac{113 * \sin 95}{\sin 49.88}$$

Computer shows

```
113*SIN95/SIN49.88
```

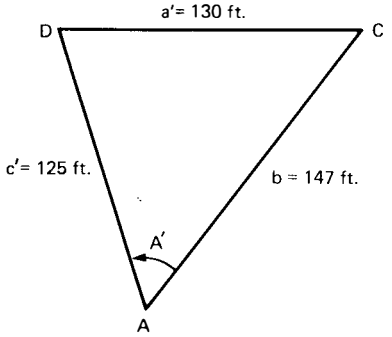
```
147.2087279
```

side b ≈ 147 ft.

Angle A′

$$a'^2 = b^2 + c'^2 - 2bc'\cos A'$$

$$\cos A' = \frac{b^2 + c'^2 - a'^2}{2bc'}$$



Computer shows

```
ACS((147^2+125^2-130^2)/(2*147*125))
```

```
56.0587622
```

$A' \approx 56.4$ degrees

Area of $\triangle ACD$ =
$\frac{1}{2}bc'\sin A'$

Computer shows

```
.5*147*125*SIN56.4
```

```
7652.463899
```

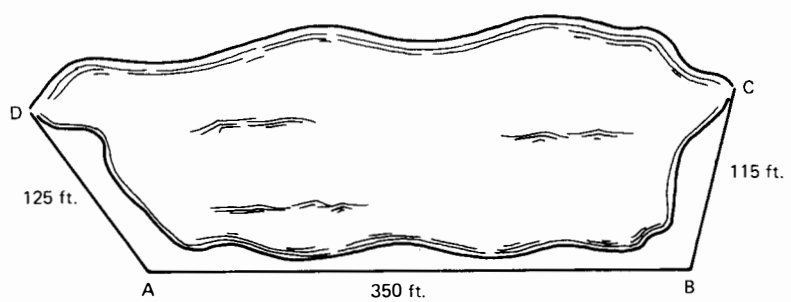
Area of $\triangle ACD$
$\approx$ 7652 sq. ft.

Total area of lot = area of $\triangle ABC$ + area of $\triangle ACD$
= 4785 sq. ft. + 7652 sq. ft.
= 12437 sq. ft.

Both methods give the same total area.

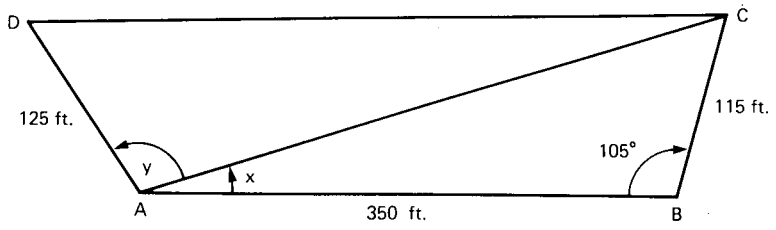
Width of Lake Problem

A surveyor wants to measure the width of Lake Deep Blue. He does it by measuring between points A and B, which are on the shore. He sights an angle of 105 degrees between the line AB and one end of the lake. He sights an angle of 125 degrees between line AB and the other end of the lake. He also measures from one end of the lake to point A and from the other end of the lake to point B. Using these measurements, as shown in the following sketch, and your Pocket Computer, find the width of Lake Deep Blue.



Solution 1

First we drew the following diagram, filling in the information given.



Solving for line segment AC, we have:

$$\overline{AC} = \sqrt{350^2 + 115^2 - 2(350)(115)(\cos 105)}$$

On the computer

```
√(35Ø^2+115^2-2*35Ø*115*COS 1Ø5)
395.6765511
```

$\overline{AC} \approx 395.7$ ft.

Solve for angle x by using the inverse cosine function.

x =arccos((350² + 395.7² − 115²)/(2*350*395.7))

y = 125 − x

On the computer

```
ACS((35Ø 2+395.7 2-115 2)/(2*35Ø*395.7)
16.3Ø219452
```

x ≈ 16.3 degrees

y ≈ 125 − 16.3 = 108.7 degrees

Solve for line segment $\overline{DC}$ by using the law of cosines.

$$\overline{DC} = \sqrt{125^2 + 395.7^2 - 2*125*395.7*\cos 108.7}$$

On the computer

```
√125^2+395.7^2-2*125*395.7*COS1Ø8.7
451.5751655
```

$\overline{DC} \approx 451.6$ ft.

Solution 2 (Check for Solution 1)

$$\overline{DB} = \sqrt{125^2 + 350^2 - 2(125)(350)(\cos 125)}$$

On the computer

```
√125^2+350^2-2*125*350*COS125
```

```
433.95038 67
```

$\overline{DB} \approx 433.95$ ft.

$$z = \arccos((350^2 + 433.95^2 - 125^2)/(2*350*433.95)$$
$$w = 105 - z$$

On the computer

```
ACS((350^2+433.95^2-125^2)/(2*350*433.95))
```

```
13.648150 22
```

$z \approx 13.65$ degrees
$w \approx 105 - 13.65 = 91.35$ degrees

$$DC = \sqrt{433.95^2 + 115^2 - 2*115*433.95*\cos 91.35}$$

On the computer

```
√(433.95^2+115^2-2*115*433.95*COS91.35)
```

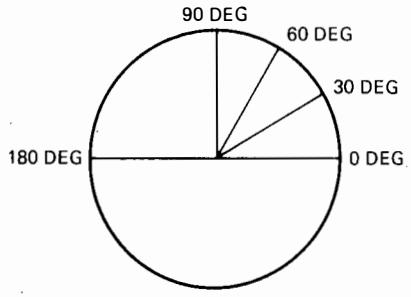
```
451.54077 03
```

$\overline{DC} \approx 451.5$ ft.

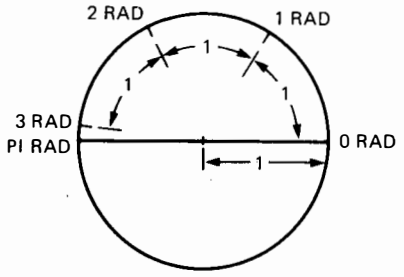
Both solutions give results within 0.1 feet.

Radii and Other Measures

Your TRS-80 Pocket Computer measures angles in two ways. The first way is the degree measure method that you learned in school. If you type the command DEG. in a program, the computer will use the degree measure until you tell it to change.



The second way of measuring angles is called radian measure.



Radian measure is used in the mathematical sciences. The radius of the circle is used as a measuring tape to measure along the arc of the circle. It takes slightly over 3 radius lengths to get halfway around the arc of the circle. In fact, it takes exactly pi = 3.14159 . . . radius measures to get halfway around the arc of the circle. That's where pi comes from.

Many beautiful and important formulas from mathematics and science depend on radian measure. One of the most surprising and important formulas was discovered by Isaac Newton.

Newton's Sin

Isaac Newton discovered this formula 350 years ago by exceedingly ingenious methods. It relates the SIN(X) function to an unending sum of powers of X.

$$\sin(X) = X\wedge 1/1 - X\wedge 3/(3*2*1) + X\wedge 5/(5*4*3*2*1) - \text{(you can guess the next terms)}$$

The Pocket Computer should be set to the RAD measure for angles before computing SIN(X). Just put the command RAD. as a program line. Now that you know so much about this wonderful formula, perhaps you'd like to check that it really works. Write a program that takes the number X as input and computes the partial sum out to the term X∧21/(21*20*. . .*2*1). The program should print out both SIN(X) and S.

Hint to Newton's Sin

The chief difficulty is the long and complicated sum S. S has too many terms to fit onto one line. What to do? An accumulator saves the day. We can start S empty, and then keep adding new terms into it.

S=0
S=S + X
S=S − X∧3/(3*2*1)

And so on, until done. Can we use the pattern of the formula to make the job easier? Each new term can be gotten from the one before it.

Solution to Newton's Sin

Here is the crude but simple version:

```
1Ø RAD
2Ø S=Ø
3Ø INPUT X
4Ø S=S + X
5Ø S=S − X∧3/(3*2*1)
6Ø S=S + X∧5/(5*4*3*2*1)
7Ø S=S − X∧7/(7*6*5*4*3*2*1)
ETC
14Ø S=S + X∧21(21*2Ø*19* ETC *2*1)
15Ø PRINT SIN(X), S
16Ø GOTO 1Ø
```

You may have noticed a few ETCs scattered about the previous program. It is a bit long and cumbersome. Let's see if we can avoid writing all those lines. Observe that each new term differs only slightly from the term before it. The sign changes. The power of X increases by 2. The denominator gets 2 more factors.

Suppose we had gotten as far as the order 3 term `−X∧3/(3*2*1)`. To get the next term, we multiply by −1*X*X and divide by 4 and by 5. We get `+X∧5/(5*4*3*2*1)`. Here is a program based on this observation:

```
1ØØ ''TOP''
11Ø RAD.: REM **REM SET RAD MODE
12Ø S=Ø: REM ** START ACCUMULATOR
13Ø N=1: REM ** START EXPONENT COUNTER
14Ø INPUT X
15Ø T=X : REM ** FIRST TERM IS X

2ØØ ''ADDTERM''
21Ø S=S+T
22Ø N=N+2: REM ** INCREMENT EXPONENT
23Ø REM: CHECK IF DONE
24Ø IF N>21 THEN ''PRINT''
25Ø T=T*(−1)*X*X/(N*(N−1))
26Ø GOTO ''ADDTERM''

3ØØ ''PRINT''
31Ø PRINT SIN(X),S
32Ø GOTO ''TOP''
```

Square Wave

The SIN and COS functions on your Pocket Computer are remarkably useful. The numbers they generate can be used to describe the vibration of strings, oscillations in electric lines, and other such matters. If you've never watched the sin numbers oscillate, then here is your chance. Try this little program and watch the Y numbers rise and fall as T(time) changes.

```
10 DEG: REM ** SET DEGREE MODE
20 T=0
30 T=T+10
40 Y=SIN(T)
50 PAUSE Y
60 GOTO 30
```

You can make the number oscillate twice as fast by changing line 40 to this:

```
40 Y=SIN(2*T)
```

The frequency of the oscillations has now doubled.

The great scientist Baron Jean Baptiste Joseph Fourier discovered an amazing fact 150 years ago. He found that any kind of changing number pattern can be built by adding sin and cos functions together. Here is an example of such a function built out of SIN functions of various frequencies:

Y = sin(1*T)/1 + sin(3*T)/3 + sin(5*T)/5 + sin(7*T)/7 + (You can guess the rest)

The numbers (Y) that are generated by this complicated formula are surprising. Write a program which takes T as input and then computes and prints Y. Since the formula goes on forever, it is best to stop somewhere. Compute the sum of the terms out to SIN(99*T)/99.

Hint to Square Wave

It looks like an accumulator is needed to collect the terms. The general term has this form:

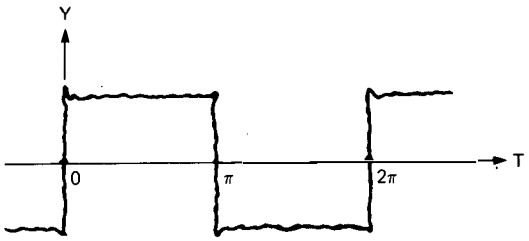
```
sin(N*T)/N
```

Where N increases by two each term. Would a FOR-NEXT loop do it?

Solution to Square Wave Problem

```
1Ø RADIAN
2Ø Y=Ø
3Ø INPUT T
4Ø FOR N=1 TO 99 STEP 2
5Ø Y=Y+SIN(N*T)/N
6Ø NEXT N
7Ø PRINT Y
8Ø GOTO 1Ø
```

The numbers are surprising. You might have expected that the numbers would vary widely. In fact, they vary hardly at all. If you graph the numbers, you will get a square wave that looks like this:



What Fourier discovered was that by putting SIN and COS functions together in the previous way, it is possible to get any function, with any graph. This lies at the heart of modern communication and systems analysis.

Summing up Chapter Five

Trigonometric functions were used in this chapter to solve problems involving angles and sides of geometric figures.

Pocket Computer Trigonometric Functions

Function	Accepted Abbreviations	Remarks
SIN	SI.	$\frac{\textit{Side opposite an angle}}{\text{hypotenuse}}$
COS	—	$\frac{\textit{Side adjacent to an angle}}{\text{hypotenuse}}$
TAN	TA.	$\frac{\textit{Side opposite an angle}}{\text{Side adjacent to an angle}}$
ASN	AS.	The angle whose sine is specified
ACS	AC.	The angle whose cosine is specified
ATN	AT.	The angle whose tangent is specified

Note: Periods in abbreviations must be included.

Other new items:

Angle Measure	Accepted Abbreviations	Remarks
DEGREE	DEG. DEGR. DEGRE.	An angular measure = 1/360 of a circle
RADIAN	RA. RAD. RADI. RADIA.	An angular measure. One radian is the angle which intercepts an arc equal to the radius of the circle.
GRAD	GR. GRA.	An angular measure equal to 1/100 of a right angle (0.9 degree)

Equations used:

- Law of Cosines:

$$a = \sqrt{b^2 + c^2 - 2bc \cos A}$$

$$b = \sqrt{a^2 + c^2 - 2ac \cos B}$$

$$c = \sqrt{a^2 + b^2 - 2ab \cos C}$$

- Law of Sines:

$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c}$$

- Areas of triangles:

Two angles and a side known:

$$\text{Area} = \frac{a^2 \sin C \sin(180 - (A + C))}{2 \sin A}$$

Two sides and included angle known:

$$\text{Area} = \tfrac{1}{2}ab \sin C$$

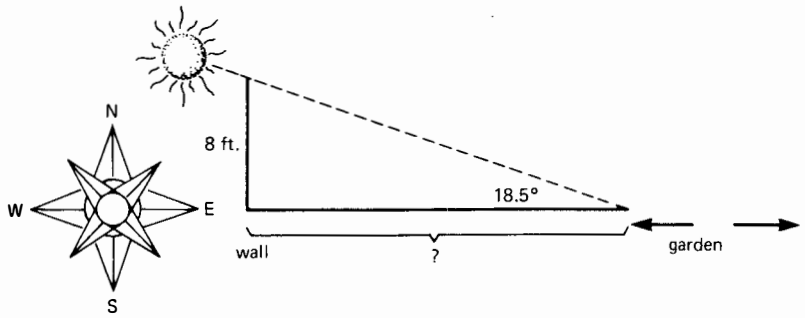
Three sides known:

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

where $s = \tfrac{1}{2}(a + b + c)$

Chapter Five Self-Test

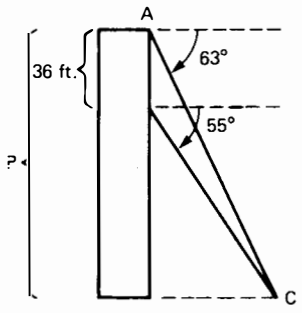
1. A year-round garden is to be planted near a retaining wall, 8 feet tall. On December 21, the shortest day in the year, the angle of elevation of the sun at 5 PM is 18.5 degrees. A shadow from the retaining wall is cast toward the garden. We want to keep the sun on the garden as long as possible. To keep the sun on the garden until 5 PM of December 21, how far east of the retaining wall should we plant the garden? _____



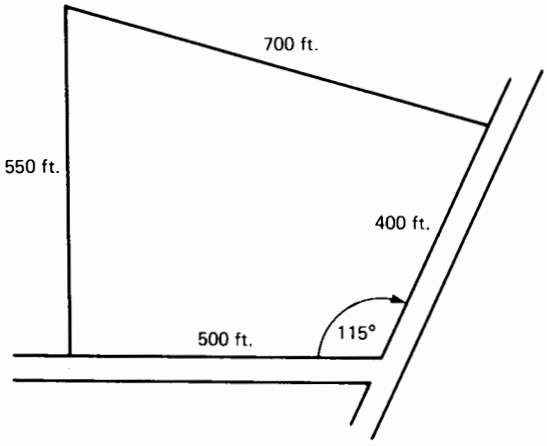
2. A ship sails due west for 120 miles from San Diego. Then it changes direction 24.7 degrees, from west to south. It sails 75 miles in the new direction. At that time, how far is it from San Diego? _____

Draw sketch here.

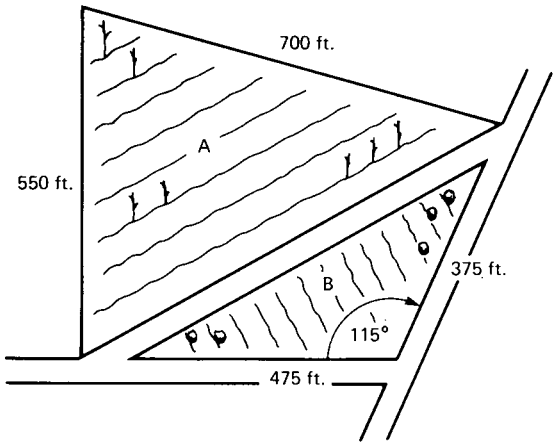
3. The angle of depression from the top of a building (point A in the figure) to point C is observed to be 63 degrees. From a window four floors (36 feet) directly below point A, the angle of depression is 55 degrees to point C. How high is the building? _____



4. A farm, bounded by two roads, has the following dimensions:



A new road is going to cross the farm, as shown here:



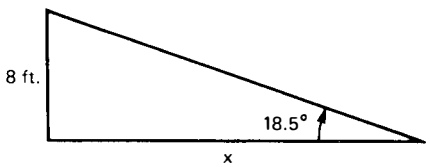
What are the areas of the two remaining parts of the farm after the new road goes in?

Area of ΔA = _____

Area of ΔB = _____

Answers to Chapter Five Self-Test

1.



$\frac{8}{x} = \tan 18.5°$

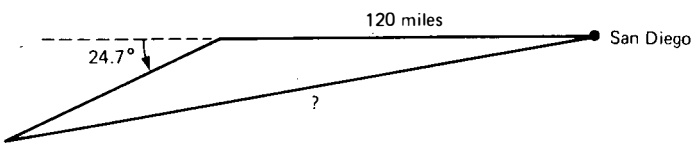
On the computer

```
8/TAN 18.5
```

```
23.9094797
```

The garden should start at least 23.9 feet from the wall.

2.



Angle X = 180 − 24.7 = 155.3°
Distance from San Diego =

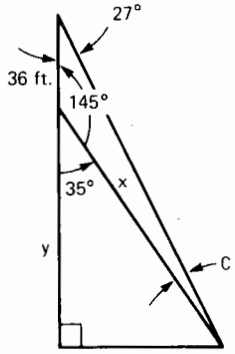
On the computer

```
√(75∧2+12Ø∧2−2*75*12Ø*COS 155.3)
```

```
                    19Ø.73Ø5616
```

The ship is approximately 191 miles from San Diego

3.



$C = 180 - (145 + 27) = 8°$

$$\frac{36}{\sin 8} = \frac{x}{\sin 27}$$

$$\frac{y}{x} = \cos 35°$$

On the computer

```
36*SIN27/SIN8
```

```
.   .          117.4340291
```

x

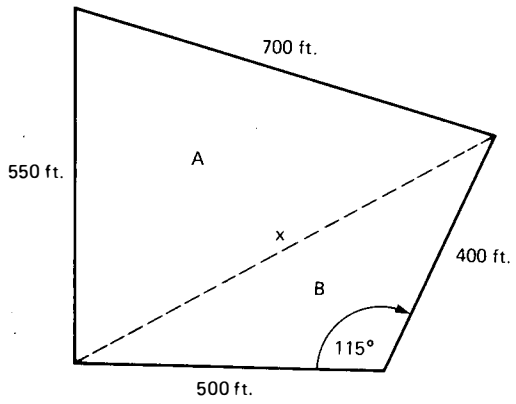
Type *COS35

```
                    96.19632501
```

y

The height of the building is 36 + 96 = 132 feet

4. From the original farm:



side $x = \sqrt{400^2 + 500^2 - 2(400)(500)(\cos 115)}$

On the computer

```
√(400∧2+500∧2-2*400*500*COS115)
```

```
760.9515784
```

$X \approx 761$ feet

Solving for the semiperimeter of $\triangle A$, we have

$$s = \tfrac{1}{2}(700 + 550 + 761) = 1005.5 \text{ feet}$$

$$\text{Area of } \triangle A = \sqrt{s(s - 700)(s - 550)(s - 761)}$$

On the computer

```
√(1005.5*(1005.5-700)*(1005.5-550)*(1005.5-761))
```

```
184961.0436
```

Area of $\triangle A \approx 184961$ sq. ft.
The area of triangle B is found next.
Area of $\triangle B = \tfrac{1}{2}(475)(375)(\sin 115)$

On the computer

```
.5*475*375*SIN115
```

Area of $\triangle B \approx 80718$ sq. ft.

```
80718.03728
```

CHAPTER SIX

Operation Time, Logic Functions, and Binary Bins

To get the most efficient use of your Pocket Computer, you should investigate the time it takes to perform its many operations. The early part of this chapter is devoted to timing events that take place in the computer.

Quite often, a programmer would like his or her program to deviate from its normal sequence of execution when the outcome of two or more separate events take on certain conditions. By using the outcome of certain logic functions, this can be done. We will show you how to program some logic functions that are not built into the Pocket Computer.

Finally, we will discuss the binary number system and some of its characteristics. There is a close connection between logic functions and binary numbers.

In this chapter you will learn:

- to time Pocket Computer BASIC statements as they are performed;
- to shorten problem solution time by careful choice of methods and variables used;
- to use the Pocket Computer logic statements, and some others that you can program;
- how to use the binary number system with its relationship to logic.

Operation Time

How fast is the TRS-80 Pocket Computer? The only way to find out is to have it do a task and time how long it takes. But short tasks are completed so fast that you can't distinguish between the start and the end of the task. To help out, you can put a BEEP at the beginning of the program that performs the task and put another BEEP at the end of the program. Try timing this program which adds two numbers and stores the result between BEEPS.

```
10 BEEP(1)
20 X = 2+3
30 BEEP(1)
```

Did it go by too fast to time? The BEEPs take longer than the addition.

You can slow the computer down by making it do the same task 500 times. Try the following program. Use a clock with a second hand to measure how long it takes the computer to run through the empty FOR-NEXT loop 500 times. The FOR-NEXT loop is called "empty" because the computer doesn't do anything between the FOR statement and the NEXT statement. (We are going to use this fact later to get time statements that we insert in the loop.) The FOR-NEXT loop will then no longer be empty. Time the following program from the first BEEP to the second BEEP.

Notice: we left out line 30 →

```
10 BEEP(1)
20 FOR X = 1 TO 500
40 NEXT X
50 BEEP(1)
```

We purposely skipped line 30. After we establish the running time for the empty loop, we'll insert some other statements that we want to time.

How many seconds does it take the computer to run the FOR-NEXT loop 500 times? _____

We measured 90 seconds between BEEPs.

Each second is equal to 1000 milliseconds. If it takes 90 seconds to do 500 empty FOR-NEXT loops, how many milliseconds does it take to do a single loop?

_____

Ours took: $\frac{90 \times 1000}{500} = 180$ milliseconds/loop

Now, replace X in the previous program with the variable, A.

Change →

```
10 BEEP(1)
20 FOR A = 1 TO 500
40 NEXT A
50 BEEP(1)
```

Run the revised program and time it again. What happened? It took longer this time. Why should the computer slow down when A is used instead of X? How long between BEEPs this time? _____

Ours took 116 seconds this time, or 232 milliseconds per loop.

Remember, variables A through Z are assigned to the 26 fixed memories in this way: (See Chapter 2 for a refresher if needed.)

A to memory 1
B to memory 2
C to memory 3
. .
. .
. .
. .
X to memory 24
Y to memory 25
Z to memory 26

The TRS-80 Pocket Computer searches for the variables from the last (Z) to the first (A). Therefore, it takes less time to access the variable X than the variable A. You should remember this if you wish to speed up programs that access a variable many times.

> Use letters at the end of the alphabet for variables that are often accessed in a program.

Go back to X as the variable in our timing program and find out how long it takes to load a number into the memory assigned to the variable A.

```
1Ø BEEP(1)
2Ø FOR X = 1 TO 5ØØ
3Ø A = 1
4Ø NEXT X
5Ø BEEP(1)
```

← store 1 into A 500 times.

Remember, it took approximately 180 milliseconds to execute the FOR-NEXT loop when there was no other operation taking place. Now we have one operation (A = 1) inside the loop. It took 90 seconds to perform the empty FOR-NEXT loop 500 times. Now you'll see how long it takes when one operation is added. Run the program and time it.

a. How long did it take between BEEPs? _____
b. 500 empty FOR-NEXT loops took _____.
c. Therefore, placing 1 into memory A 500 times took _____.

d. To place 1 in memory A once took _____.

a. 129 seconds
b. 90 seconds

c. 39 seconds for 500 stores

d. $\frac{39 \times 1000}{500}$ = 78 milliseconds to assign 1 to A

Now let's see how long it would take to do the same thing in memory location Z. Change line 30 from:

A = 1 to 30 Z = 1

Run the program again and time it.

a. What is the total time between BEEPs?_____
b. What is the total time for 500 operations of Z = 1? _____
c. Time for one operation of Z = 1? _____

a. 121 seconds between BEEPs
b. 31 seconds for 500 operations
c. 62 milliseconds to assign Z = 1

Here again, you see that the computer can work with variables at the end of the alphabet faster than it can with variables at the beginning of the alphabet.

Which arithmetic operation can be performed fastest (addition, subtraction, multiplication, or division)? To find out, change line 30 of the program, time the FOR-NEXT loop, and compute the time for each operation, as before.

Line 30	500 loops	empty loop	500 operations	one operation
30 Z = 3 + 2		90		
30 Z = 3 − 2		90		
30 Z = 3 *2		90		
30 Z = 3/2		90		

Using the previous technique, time the following operations. Fill in the table with the time, in milliseconds, for each operation. You know the answers to the first five operations from the previous experiment.

TIME TABLE

Operation	Time (milliseconds)
Z = 1	
Z = 3 + 2	
Z = 3 − 2	
Z = 3 * 2	
Z = 3/2	
Z = (2<3)	
GOSUB 100 . . . RETURN	
GOTO 40	
Z = SIN 1 (rad.)	
Z = COS 1 (rad.)	
Z = TAN 1 (rad.)	
Z = $\sqrt{1}$	
Z = 2∧3	
Z = LOG 1	
Z = EXP 1	
IF 0 THEN 1	

TIME TABLE

Operation	Time (milliseconds)
Z = 1	62
Z = 3 + 2	92
Z = 3 − 2	92
Z = 3*2	102
Z = 3/2	124
Z = (2<3)	120
GOSUB 100 . . . RETURN	212
GOTO 40	72
Z = SIN 1	622
Z = COS 1	620
Z = TAN 1	466
Z = $\sqrt{1}$	116
Z = 2∧3	326
Z = LOG 1	170
Z = EXP 1	250
IF 0 THEN 1	60

Problem # 2

All of the questions below relate to the speed of your computer's operation.

a. Which computation would run faster: 2*2*2*2*2 or 2∧5?

_____

b. Does the size of a number affect how fast addition or multiplication operations are performed? _____
If so, how? _____

c. Which takes longer, GOTO 1 or GOTO 100? _____

d. How long does it take to fill memories 31 through 80 with the number 1?

_____

e. Which works faster, $\sqrt{3}$ or 3∧.5? _____

f. Which works faster, 1/3 or 3∧− 1? _____

a. 2*2*2*2*2 is faster
b. Yes
The larger the number, the slower the operation is performed.
c. GOTO 100 takes longer
d. 46 to 47 seconds, using a FOR-NEXT loop
e. $\sqrt{3}$ is faster
f. 1/3 is faster

Logical Comparisons

There will be times when you will want to check whether a number is big enough, or small enough, to perform a specific operation. There will also be times when you will want to check whether two strings are the same. The Pocket Computer has several logical comparison functions for this purpose. These functions compare numbers, or strings, for size or equality. If the condition is met, then the value of the function is equal to 1. If the condition is not met, the value of the function is equal to 0.

Here are a few logical comparisons. Write down what you think the value of each expression will be (0 or 1). Write 0 if the statement is false. Write 1 if the statement is true.

1. $3 = 2$ _____
2. $2 = 2$ _____
3. $4 < 5$ (4 is less than 5) _____
4. $2 < 1$ _____
5. $5 > 1$ (5 is greater than 1) _____
6. $7 > 9$ _____
7. $3 \geq 2$ (3 is greater than or equal to 2) _____
8. $5 \leq 5$ _____
9. $4 <> 5$ (4 is not equal to 5) _____
10. $4 <> 4$ _____
11. $(2 < 5) < (5 > 2)$ _____
12. What would this do? IF $(3 > 2)$ PRINT "GREATER THAN"
    _____
13. How is A related to $(A = 0)$? _____
    _____
    _____
    _____
    _____
    _____
14. What would PRINT $(5 = 3)$ do? _____
    _____

1. 0 (since $3 = 2$ is false)
2. 1 (since $2 = 2$ is true)
3. 1 (since 4 is less than 5)
4. 0 (since $2 < 1$ is false)
5. 1 (since 5 is greater than 1)
6. 0 (since $7 > 9$ is false)
7. 1 (since 3 is greater than, or equal to, 2)
8. 1 (since 5 is greater than, or equal to, 5)
9. 1 (since 4 is not equal to 5)
10. 0 (since $4 <> 4$ is false)

11. 0 Let's take this one in pieces.
    (2 < 5) has a value of 1 (it is true)
    (5 > 2) has a value of 1 (it is true)
    (2 < 5) < (5 > 2) has the same value as 1 < 1 which is false. Therefore, the complete expression is false and has value 0.
12. This prints GREATER THAN (since it is true that 3 > 2)
13. If A = 0, then (A = 0) is true — takes the value of 1.
    IF A = 1, then (A = 0) is false — takes the value of 0.
    In either case, the value of (A = 0) is the opposite of the value of A.
14. 0 (since (5 = 3) has a value of 0)

The Truth and Nothing but the Truth

Here are two problems involving the logical comparison functions AND and OR. The AND function will be true only if the two conditions being ANDed are both true. Otherwise the value of the AND function is false. The OR function will be true if either one of the conditions being ORed is true (or both are true). It is false only when both conditions are false.

Problem # 1

Write a program which takes the values of A and B as input. In each case, the program is to print the corresponding output values (A AND B). Don't use any BASIC statements except INPUT, PRINT, GOTO, and the logical operators <, >, and =.

	Inputs A	B	Output A AND B
Case 1	0	0	0
Case 2	0	1	0
Case 3	1	0	0
Case 4	1	1	1

10 _____
20 _____
30 _____

```
1Ø INPUT A,B
2Ø PRINT (A = Ø)<B
3Ø GOTO 1Ø
```

Problem # 2

Write a program which takes the values of A and B as inputs and prints out the value of the logical function A OR B. Use the same BASIC statements and operators as in Problem # 1.

	Inputs		Output
	A	B	A OR B
Case 1	0	0	0
Case 2	0	1	1
Case 3	1	0	1
Case 4	1	1	1

10 _____
20 _____
30 _____

```
1Ø INPUT A,B
2Ø PRINT (A>B) <>B
3Ø GOTO 1Ø
```

Binary Bins

One of the things that computers are reputed to be good at is counting in the binary number system. This is because they are constructed to work with binary numbers. In fact, that's the only kind of numbers that the computer *can* work with. Someone else had to write an interpreter to interpret BASIC commands and decimal numbers into binary language that it can understand.

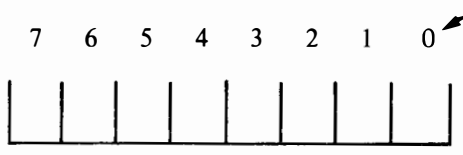
Here are the first few positive numbers in binary notation.

0, 1, 10, 11, 100, 101, 110, 111, 1,000, . . . etc.

If your car had a binary odometer, it would work just like the binary counter that we will construct a little later by a computer program. First, though, let's talk a little more about binary numbers.

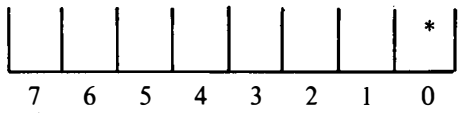
You could make a binary counter using stones and boxes.

Line up eight boxes in a row.



each box numbered (computer people like to start numbering with zero)

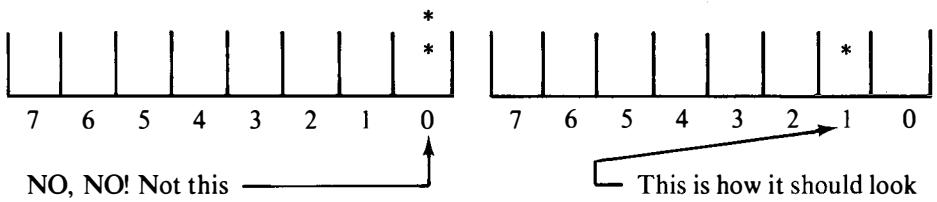
Put a stone in the first box.



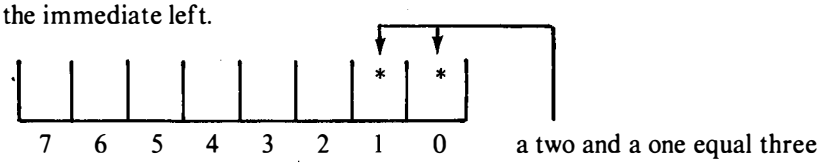
That's one stone counted. Now binary boxes hold, at most, one stone. Suppose you wanted to count another stone: There's no room in the first box. That's OK. The next box to the left counts twice as much as the first box. The rule is this:

> If any box already contains a stone, and you are trying to put another stone into the box, DON'T. Instead, empty that box and carry *one* stone over to the next box on the immediate left.

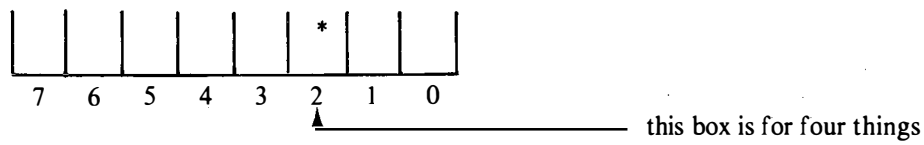
Sounds like fun, doesn't it? Here's how the boxes look as two stones are counted.



The second box counts for two. To count three things is easy. Just add another store. There is room for it in the first box. Always work from right to left. If a box is empty, you can put in a stone. If it has a stone in it, empty it and take a stone to the box on the immediate left.



What happens if you try to count a fourth thing? You try to add another stone to the first box. No room — there's already a stone there. Remember the rule, empty that box and carry *one* stone to the next box on the left. But the next box is full. Empty it and carry one stone to the next box on the left. There is room this time. Put the stone in that box. Here is how the final result for four things looks.



If we let 00000 stand for 5 empty boxes and 11111 stand for 5 full boxes, then what number does 11111 represent? If you said 31, then you know how to count in the binary number system.

Binary Counter Problem

Write a program which prints out the binary numbers in order. The output should look like this:

```
00000
00001
00010
00011
  .
and so on until
  .
11111
```

Hint 1 to Binary Counter

The brute force method is to write out a series of 32 PRINT statements, one for each number. You can be cleverer than that. The first thing that you need to do is decide where to store the 5 binary digits. Let's store them in memories A, B, C, D, and E.

When you start, all the memories will contain 0.

```
0 0 0 0 0
A B C D E
```

To set the next number, you can add 1. Since E = 0, you can set E = 1 and print the result. You might write this in BASIC like this:

```
IF E = 0 LET E = 1: GOTO "PRINT
```

The line labeled "PRINT" will do the obvious thing and print the result. The memories A, B, C, D, and E would now look like this:

```
0 0 0 0 1
A B C D E
```

Next, repeat the process and add 1 to get the next number. To do this, check E. It is already equal to 1. Remember the binary counter rule: Change the 1 to 0 and carry a 1 to the box to the left. Here is how the process might look in BASIC:

```
IF E = 0 LET E = 1: GOTO "PRINT"
IF E = 1 LET E = 0
IF D = 0 LET D = 1 : GOTO "PRINT"
IF D = 1 LET D = 0
and so on
```

Using this procedure, write a program to count and print from 00000 to 11111.

a. 10 _____
20 _____
30 _____
40 _____
50 _____
60 _____

70 _____
80 _____
90 _____
100 _____
110 _____
120 _____
130 _____

a.
```
10 A = 0: B = 0: C = 0: D = 0: E = 0
20 ''PRINT''
30 PRINT A;B;C;D;E
40 IF E = 0 LET E = 1: GOTO ''PRINT''
50 IF E = 1 LET E = 0
60 IF D = 0 LET D = 1: GOTO ''PRINT''
70 IF D = 1 LET D = 0
80 IF C = 0 LET C = 1: GOTO ''PRINT''
90 IF C = 1 LET C = 0
100 IF B = 0 LET B = 1: GOTO ''PRINT''
110 IF B = 1 LET B = 0
120 IF A = 0 LET A = 1: GOTO ''PRINT''
130 IF A = 1 END
```
← You might use PAUSE here

Notice pattern in the program. Lines 40 and 50 look much like lines 60 and 70, lines 80 and 90, and lines 100 and 110. You can make the pattern more explicit by using subscripted variables. Let's change the names of the variables from A, B, C, D, and E to A(5), A(4), A(3), A(2), and A(1).

With these changes you can use a FOR-NEXT loop to simplify the program. Lines 40 to 110 can be replaced to shorten the program to:

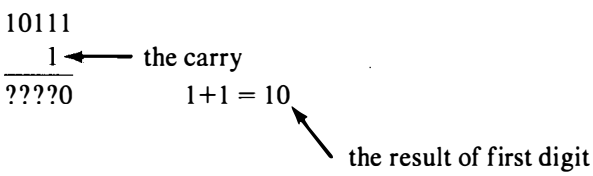
```
10 A(1) = 0: A(2) = 0: A(3) = 0: A(4) = 0: A(5) = 0
20 ''PRINT''
30 PRINT A(5);A(4);A(3);A(2);A(1)
40 FOR Z = 1 TO 4
50 IF A(Z) = 0 LET A(Z) = 1: GOTO ''PRINT''
60 IF A(Z) = 1 LET A(Z) = 0
70 NEXT Z
```

Hint 2 to Binary Counter

Here is another method that you might use. Notice that adding 1 at the right-most digit starts a process of adding and carrying that procedes to the left. For example, suppose you wish to add a number X = 1 to the binary number 10111.

```
 10111
     1 ← X = 1
-------
 ?????
```

You get a digit of the result and possibly a carry to the left. That means you perform a similar addition of X = 1 to the next digit to the left.



This add and carry process procedes to the left until you reach some position for which there is no carry. If there is no carry, then you are done.

At each stage in this process, you compute two numbers: the digit of the result and the digit of the carry. Look at the process using the notation of the first solution.

Suppose A(5), A(4), A(3), A(2), and A(1) represent the digits of a counter. Let X represent the carry. Initially, as you begin the process of incrementing the counter, X = 1.

				A(5) A(4) A(3) A(2) A(1)	
					X
A(15)	A(14)	A(13)	A(12)	A(11)	
					?

You want to compute A(11) and a value for X to carry to the left. If A(1) and X have the same value, then A(11) = 0. If A(1) and X have different values, then A(11) = 1. We can say it this way in BASIC:

```
IF A(1) = X LET A(11) = Ø
IF A(1) <> X LET A(11) = 1
```

Write a program for a binary counter based on these hints. Use a FOR-NEXT loop to keep things moving.

a.
```
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
```

```
 90 _____________________________
100 _____________________________
110 _____________________________
120 _____________________________
130 _____________________________
140 _____________________________
150 _____________________________
160 _____________________________
170 _____________________________
```

You may not need this many lines, but we like lots of remarks

---

a.
```
 1Ø REM SET COUNTER TO Ø
 2Ø FOR Z=1 TO 6: A(Z)=Ø: A(Z+1Ø)=Ø: NEXT Z
 3Ø ''PRINT''
 4Ø X = 1
 5Ø PAUSE A(5);A(4);A(3);A(2);A(1)
 6Ø FOR Z = 1 TO 6
 7Ø   IF A(Z) = X LET A(Z+1Ø) = Ø
 8Ø   IF A(Z)<>X LET A(Z+1Ø) = 1
 9Ø   X = X*A(Z)
1ØØ   IF X = Ø LET Z = 6: REM EXIT THE LOOP
11Ø NEXT Z
12Ø REM TRANSFER RESULT A(Z+1Ø) TO A(Z)
13Ø FOR Z=1 TO 6: A(Z)=A(Z+1Ø): NEXT Z
14Ø REM CHECK IF DONE
15Ø IF A(6) = 1 END
16Ø REM GO DO IT AGAIN
17Ø GOTO ''PRINT''
```

Really cramming it in

Your program may be quite different from this. Try yours and see if it counts correctly. If it does, your program is OK.

Here is yet another solution. By a judicious use of the logical comparison functions, you can simplify the program a bit more. This last solution is written in the form of a subroutine called "COUNTER". The subroutine is initialized in line 20. It returns the binary digits in memories A(6), A(5), A(4), A(3), A(2), and A(1). When all the memories A(1) through A(5) are full, A(6) will equal 1.

```
1Ø REM INITIALIZE ''COUNTER''
2Ø FOR Z = 1 TO 6: A(Z) = Ø: NEXT Z
3Ø ''PRINT''
4Ø PAUSE A(5);A(4);A(3);A(2);A(1)
5Ø GOSUB ''COUNTER''
6Ø IF A(6) = 1 END
7Ø GOTO ''PRINT''

2ØØ ''COUNTER''
21Ø X = 1
22Ø FOR Z = 1 TO 6
23Ø W =(A(Z)<> X): X = X*A(Z)
24Ø A(Z) = W
25Ø IF X = Ø LET Z = 6
26Ø NEXT Z
27Ø RETURN
```

Why Binary Numbers Are Useful

What are binary numbers good for? For one thing, computers use them, and an understanding of binary numbers help you understand how computers operate. But there is another reason why binary numbers are useful. For example, suppose you have four flavors: salt, sweet, sour, and bitter. You want to experiment by adding various combinations of the flavors to bread dough to see how it tastes. You might try salt and sour, or you might try sweet, sour, and bitter. You might even leave them all out, or put them all in.

How many different ways are there to mix the four flavors? The answer is not obvious. Did you say 16? One way to see that there are 16 different combinations of the four flavors is to notice that each binary number from 0000 to 1111 could represent a selection of the four flavors. The number 0000 represents the case where all the flavors are left out. The number 1111 represents the case where all the flavors are added. The number 1100 could represent the case where salt and sweet are added. For each possible combination, there is a binary number, and for each binary number there is a unique combination. There are 16 binary numbers between 0000 to 1111 inclusive, and there are 16 combinations of flavors.

Binary Bread Problem

Write a program to print all the 16 possible combinations of the flavors salt, sweet, sour, and bitter.

Hint: You can count in binary and use the binary number as printing directions. For example, the number 1010 would tell you to print SALT,BITTER. You will need memory space for the four binary digits. Store them in A(4), A(3), A(2), and A(1). You will also need some memories to store the four flavor names. Use A$(14), A$(13), A$(12), and A$(11) to store the flavor names.

a.
```
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____
100 _____
110 _____
120 _____
130 _____
140 _____
200 "COUNTER"
```

(Insert the "COUNTER" subroutine
from the previous program here.)

---

a.
```
10 REM INITIALIZE ''COUNTER''
20 FOR Z = 1 TO 6: A(Z) = 0: NEXT Z
30 X = 1

40 A$(14)=''SALT'': A$(13)=''SWEET'': A$(12)=''SOUR'': A$(11)=''BITTER''
50 GOSUB ''COUNTER''

60 REM CHECK IF DONE
70 IF A(5) = 1 END

80 REM PRINT CHOSEN FLAVORS
90 FOR Z = 1 TO 4
100 IF A(Z) = 1 PAUSE A$(Z+10)
110 NEXT Z

120 BEEP(1)

130 REM GO DO IT AGAIN
140 GOTO 50

200 ''COUNTER''
210 X = 1
220 FOR Z = 1 TO 6
230   W =(A(Z)<>X): X = X*A(Z)
240   A(Z) = W
250   IF X = 0 LET Z = 6
260 NEXT Z
270 RETURN
```

Summing up Chapter Six

The TRS-80 Pocket Computer performs some operations faster than others.

- Variables near the end of the alphabet are accessed quicker than those at the beginning of the alphabet.
- Some operations are faster than others that produce the same results. For example:

  2*2*2*2*2 is faster than 2^5

  However, 2∧5 is quicker to enter in a program.
- Programs and problems in this chapter have given you a better understanding of the speed with which various operations are performed.

- Some logic functions were introduced, as shown in this table.

Condition	Use	Result
Equal	(A)=(B)	1 if A=B; 0 if A ≠ B
Greater than	(A)>(B)	1 if A>B; 0 IF A≤B
Less than	(A)<(B)	1 if A<B; 0 if A≥B
Greater than or equal to	(A)>=(B)	1 if A≥B; 0 if A<B
Less than or equal to	(A)<=(B)	1 if A≤B; 0 if A>B
Not equal to	(A)<>(B)	1 if A ≠ B; 0 if A=B
Not A	(A=0)	1 if (A=0) is false; 0 if (A=0) is true
OR	(A>B)<>B	1 if A or B (or both) is true; 0 if A and B are both false
AND	(A=0)<B	1 if A and B are both true; 0 otherwise
Exclusive OR	A<>B	1 if A or B (but not both) is true; 0 otherwise
NOR	A<(A=B)	1 if neither A nor B is true; 0 otherwise

- Binary numbers are formed from the symbols 1 and 0.
- Binary place values are powers of 2.

CHAPTER SEVEN

Feedback and Systems

This chapter centers around problems that grow on themselves. We all know that money doesn't grow on trees, but it does grow in banks. We are most concerned here with how fast it grows. Other things grow too, either larger or smaller. Our weight changes, the temperature changes — life is full of changes. In this chapter, you will learn:

- to compute the growth of money in banking;
- to predict school class growth;
- to compute weight changes from energy expended and calorie intake;
- to compute the result of several variables that affect a single event;
- to compute the result of feedback in closed systems; and
- how changing variables may lead to constant results.

Multiplier Effect Problem

The banker in our town has a good thing going. If I deposit $100 in his bank, the friendly banker puts $20 in the vault and loans out the remaining $80 to someone else, meanwhile collecting a bit of interest for his trouble. Actually, things get even better for the banker. When that $80 is loaned out, it goes to someone in town. What is there to do with money in a small town? Take it to the bank, of course — giving the joyful banker a new deposit of $80. You know what the banker does now; deposits 20% ($16) in the vault and loans out 80% ($64) to someone else. Where does the $64 go? You guessed it; right back to the bank. The process is repeated again and again. The question is this: After sufficient time has gone by, how much will be in the vault, and how much will have been loaned out?

Hint to Multiplier Effect Problem

There are three numbers that will need to be stored; the amount in the vault, the total amount loaned, and, of course, the $100 that is the new deposit.

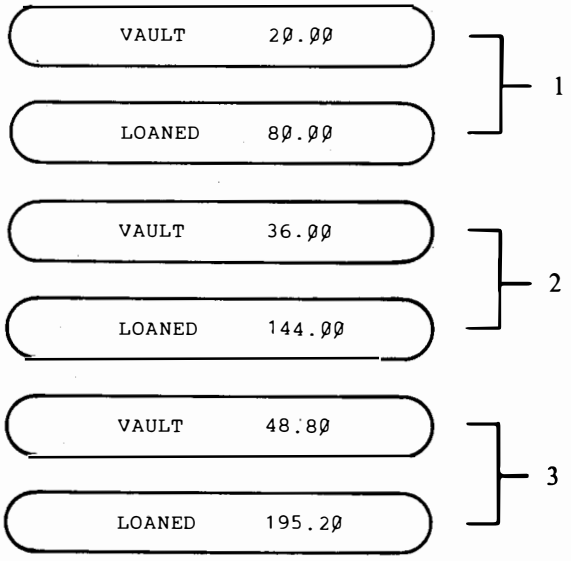
Store the amount in the vault at memory Z, the total amount loaned out at memory Y, and the new deposit at memory X. At each step in the process take 20% of the deposit and add this to the vault. Take 80% of the deposit and add it to the total loaned out, and also store it as the new deposit.

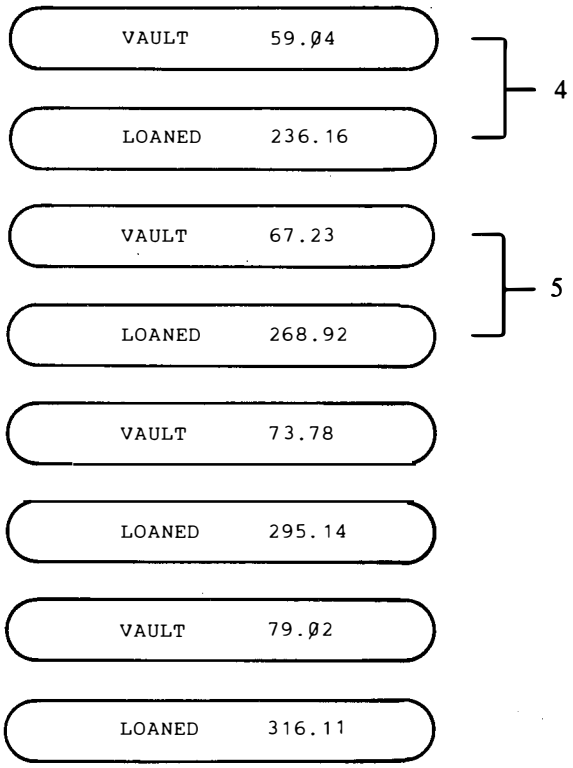
Solution to Multiplier Effect

```
15 REM THE INITIAL AMOUNT IN VAULT
20 Z=0

25 REM THE INITIAL LOAN TOTAL
30 Y=0

35 REM THE INITIAL DEPOSIT
40 X=100

45 REM 20% TO VAULT, 80% TO LOANS
50 Z=Z+.2*X: Y=Y+.8*X

55 REM THE LOAN IS NEW DEPOSIT
60 X=.8*X

65 REM DISPLAY RESULTS
70 USING ''######.##''
80 PRINT ''VAULT'':Z
90 PRINT ''LOANED'':Y

95 REM REPEAT
100 GOTO 50
```

format the data for $ and ¢

We ran the program on our Pocket Computer and this is what we saw:





At this point, we decided to change the PRINT statements to PAUSE statements and to let the program run awhile.

The changes:

```
8Ø PAUSE ''VAULT'':Z
9Ø PAUSE ''LOANED'':Y
```

We let it run for several minutes to see what happened. Since the banker only loans out 80% of the new deposit, this amount gets smaller and smaller with time. Do you think the amount in the vault and the amount loaned out will ever stabilize? Try the version with the PAUSE statements and see. We found that the amount in the vault approached $100 and the amount loaned out reached $400.

English System Problem

Every year ComputerTown College accepts 1,000 eager, bright-eyed freshmen. Every freshman takes English 1. Of those taking English 1, 80% pass and 20% do not. Those who do not pass will retake it the next year. Notice that the size of the class tends to grow because of those students who retake the course. Write a program that will print out the number of students in English 1 in each future year.

Hint to English System Problem

The initial class size is 1,000 students. You should store this number somewhere, say in Z. The next year there will be 1,000 new students, plus 20% of the initial year's class. Print this total and store the result in Z. The next class size will be 1,000, plus 20% of the class size in Z. Print this total and store it in Z. Repeat this process for each future year.

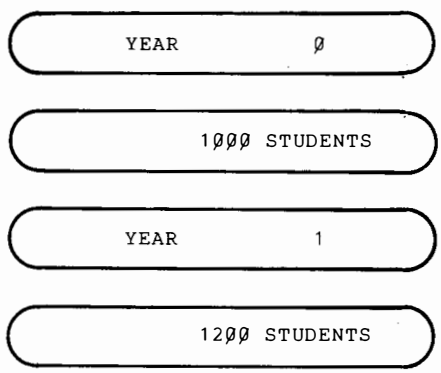
a. Will the class size stabilize at a given number? _____
b. If so, what is the number? _____

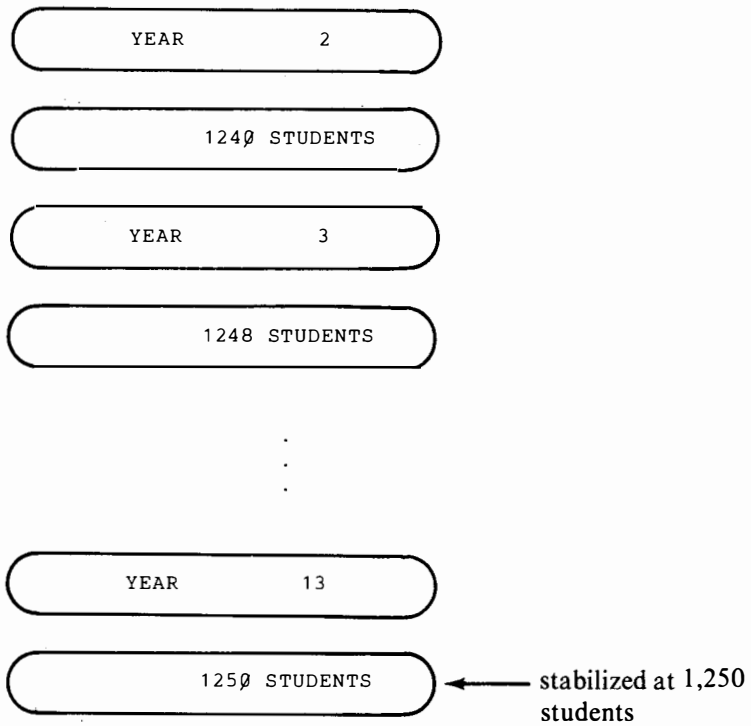
Solution to English System Problem

```
15 REM N WILL COUNT THE YEARS
20 N=0

25 REM Z IS THE NUMBER IN THE CLASS
30 Z=1000

35 PRINT CURRENT NUMBERS
40 USING ''########''
50 PRINT ''YEAR '';N
60 PRINT Z;'' STUDENTS''

65 REM COMPUTE A NEW CLASS NUMBER
70 Z=1000+.2*Z

75 REM COMPUTE NEW YEAR NUMBER
80 N=N+1
90 GOTO 50
```

a. Yes, the class size will stabilize.
b. To find out where, RUN the program (or use a little algebra.)

Our run produced these results:





Using a little algebra,

$$Z = 1000 + .2Z$$

$$.8Z = 1000$$

$$Z = 1250$$

Weight Wait Problem

You have no doubt observed that animals store fat for the winter. Wishing to be ecologically sound, organically whole, and in tune with nature, we should certainly adopt nature's own strategy. There is one difficulty. Carrying fat around takes energy; in fact, it takes 16 calories to carry around one pound of fat for one day. One must get the 16 calories by eating food or by using up fat. One pound of fat furnishes 3,500 calories. This also means that to add one pound of body fat, one must accumulate 3,500 more calories than one uses.

If a person weighs 150 pounds now and eats 2,500 calories each day, then to what weight will the person eventually grow? (Please ignore the weight of the food eaten.)

Hint to Weight Wait

First, some decisions. What do you need to store and where shall you store it? You will need to store the initial weight, W. You will need to store the new weight at the end of each day. Call this X. The new weight (X) will equal the old weight (W) minus the weight of the fat burned up to carry the old weight around, plus the weight of the fat accumulated due to the calories eaten.

Solution to Weight Wait Problem

The person's weight is initially 150 pounds. To carry around W pounds for one day requires $16 \times W/3500$ parts of a pound of fat. The person will burn up this amount of fat during the day to carry W pounds around.

The person has also eaten 2,500 calories during the day. This is equivalent to 2500/3500 parts of a pound of fat. The person will add this amount.

```
15 REM INITIALIZE WEIGHT W
2Ø W=15Ø

25 REM COMPUTE NEW WEIGHT X
3Ø X=W-16*W/35ØØ+25ØØ/35ØØ

35 REM PRINT NEW WEIGHT
4Ø PAUSE ''NEW WEIGHT IS '':X

45 REM REINITIALIZE WEIGHT W
5Ø W=X

55 REM GO COMPUTE NEW WEIGHT
6Ø GOTO 3Ø
```

The weight will grow to 156.2. Put on a pot of coffee and take a break while you run this program. It took our run about 22 minutes to reach 156.0. The closer you get to the final weight, the smaller the increases are. Here are the first few printouts that flashed by.

```
NEW WEIGHT IS 15Ø.Ø28575

NEW WEIGHT IS 15Ø.Ø57Ø2Ø

NEW WEIGHT IS 15Ø.Ø85335

NEW WEIGHT IS 15Ø.113521

NEW WEIGHT IS 15Ø.141578
```

You could speed up this routine. Notice that you are doing some needless divisions in line 30. Since 16/3500=.004571429 and 2500/3500=.7142857, you can rewrite line 30 as `3Ø X=W-.ØØ4571429*W+.7142857`. In fact, you could factor out W to get `3Ø X=.9954386*W+.7142857`. This saves one subtraction and two divisions.

The print statements take much time too. Change line 60 to

```
6Ø IF X-W<.ØØØ5 THEN 11Ø.
```

Add line 110:

```
11Ø BEEP(3):PRINT X:END
```

You don't have to keep an eye on the display, but keep your ears open for the beep. With these two changes our run took about 10½ minutes. At that point X − W was .005, and the program automatically ended. We didn't quite reach 156.2, but we did get close. The final reading was:

```
156.1426Ø59
```

By changing line 60 to `6Ø IF X-W<.ØØØØ5 THEN 11Ø` you could get much closer. However, the program would run much longer.

Paying for the Potlatch Problem

The native Indians of the Northwest have had an interesting tradition. Rather than compete and show off their possessions and wealth, they try to outdo one another in the giving of gifts. My neighbors Little Nut and Paying Deer are trying to establish this great tradition locally. Little Nut gives $10 per day in gifts to Paying Deer, and also adds $.90 extra for each $1 of gifts that he received from Paying Deer on the previous day. Paying Deer gives $12 each day to Little Nut and adds $.80 for each $1 in gifts that he received on the previous day from Little Nut.

You should check to see what happens as the days pass. Will their gifts grow without bound? Who is the most generous?

Hint for Paying for the Potlatch Problem

The new amount given by each neighbor is completely determined by the previous amounts both gave. Initially, Little Nut gave $10 and Paying Deer gave $12. It is useful to compute a few values by hand to get a feel for the computations that the computer must do. Here are the first few values.

Little Nut	Paying Deer
10	12
20.8	20
35.78	28.64

Carry on the computations until you decide what must be done in the computer program. At each new computation cycle, the new values become the old values and newer values are computed.

Solution to Paying for the Potlatch Problem

Let Y denote the current amount given by Little Nut. Let Z denote the current amount given by Paying Deer. These are given initially by Y = 10 and Z = 12 (line 20). Let L denote the new amount given by Little Nut. This new amount is $10, plus 90% of the amount given by Paying Deer (line 30). Similarly, the new amount given by Paying Deer is $12, plus 80% of the amount given by Little Nut (line 40). The first stage of the computation cycle is now finished, and you can print the new computed values (line 50). To continue the next cycle, set the current amounts equal to the new amounts and redo the calculations (lines 60 and 70).

When you make a run, you should notice the difference between Little Nut's gifts and Paying Deer's gifts. Is the difference getting larger or smaller?

```
15 REM INITIALIZE VARIABLES
2Ø Y=1Ø:Z=12

3Ø L=1Ø+.9*Z: REM LITTLE NUT'S GIFT

4Ø P=12+.8*Y: REM PAYING DEER'S GIFT

5Ø PRINT L,P

55 REM REDEFINE INITIAL VALUES
6Ø Y=L: Z=P

65 REM GO COMPUTE NEW GIFTS
7Ø GOTO 3Ø
```

Here are the first few displays of our run.

```
          2Ø.8              2Ø.
            28.            28.64
         35.776             34.4
           4Ø.96          4Ø.62Ø8
        46.55872            44.768
          5Ø.2912         49.246976
       54.3222784          52.23296
        57.ØØ9664       55.45782272
      59.912Ø4Ø45        57.6Ø77312
      61.84695Ø8         59.92963236
      63.93666912        61.47756646
      65.32980981         63.1493353
      66.8344Ø177        64.26384785
```

You might want to insert a line to print out the differences between L and P. Put it in at line 52.

```
52 PRINT L-P
```

We observed these differences:

L−P
.8
−.64
1.376
.3392
1.79Ø72
1.Ø44224
2.Ø893184
1.55184128
2.3Ø43Ø925
1.91732572
2.4591Ø266
2.18Ø47451
2.57Ø55392
.
.

Do you think the differences are approaching some constant value? Go through enough cycles to find out for yourself. You might also want to study the differences of the differences. Those of you with some higher mathematics can tie this into rates and accelerations.

Triple Threat Problem

The triplets, Arnold, Bertrand, and Clem, each got an electric blanket for their birthday this year. The blankets are of the very latest design; they cool in the summer, heat in the winter, and have a remarkable range of temperature.

The triplets do have their individual preferences. Arnold likes to be cool and is comfortable at 71 degrees. Bertrand is happy as a clam at 72 degrees. Clem likes it a bit warmer at 73 degrees.

It looked like the perfect gift, but unfortunately, there was a catastrophy. When the triplets went to bed, the temperature was 72 degrees. They did not suspect the awful truth — their controls had become switched. Arnold had the control for Bertrand's blanket, Bertrand had the control for Clem's blanket, and Clem had the control for Arnold's blanket. You can hardly imagine the trouble that followed.

Arnold was too warm at 72 degrees, so he lowered the temperature (Bertrand's temperature) by 1 degree. Meanwhile, Bertrand was perfectly comfortable at 72 degrees and left the temperature (Clem's temperature) unchanged. Clem, of course, was too cold at 72 degrees and raised the temperature (Arnold's temperature) by 1 degree. Things got worse and worse.

After one minute had gone by, all the changes had taken effect. Arnold was now feeling warm at 73 degrees, Bertrand was feeling cold at 71 degrees, and Clem was feeling cold at 72 degrees. They all turned over at the same time and changed their controls by 1 degree. Well, this made it even worse. After this minute had gone by, all the changes had taken effect, and you can clearly see that it will take a computer to describe how the three temperatures changed, minute by minute, degree by degree, throughout the tragic night.

Write a program to compute the fate of the triplets.

Hint for Triple Threat Problem

In order to see what needs to be done in a computer program, it is often useful to carry out the computations by hand until the inner logic of the program begins to become clear. Here are the first few values of the temperatures for the blankets.

ARNOLD	BERTRAND	CLEM
72	72	72
73	71	72
74	70	73
74	69	74

Carry the computation on by hand until you are clear about the calculations involved. Notice that the new values become old values for the next stage of calculation. The new temperature at each stage depends on whether some old temperature was above, at, or below the comfort level. For example, Arnold's new temperature is Arnold's old temperature, plus −1, 0, or 1, depending on whether Clem's old temperature was above, at, or below 73.

The first thing you need to do is decide what you need to store and where to store it. You need to store the initial temperatures for Arnold, Bertrand, and Clem. Use A(1), A(2), and A(3). You will be computing the next values for each of the triplets. Store the new values in A(4), A(5), and A(6).

The next thing you will need to do is figure out how to simulate the decisions to raise or lower the temperature of the adjoining blanket. Arnold, for example, will compare his present temperature A(1) with his desired temperature which is 71. Arnold will change Bertrand's old temperature A(2) by −1, 0, or 1, to get the new temperature A(5), depending on the sign of the difference, 71-A(1).

We're sure you will be able to transfer this information to Bertrand and Clem.

Solution to Triple Threat Problem

Arnold will change Bertrand's temperature by −1, 0, or 1, depending on the sign of the difference 71-A(1). Bertrand's new temperature A(5) will be Bertrand's old temperature A(2), plus the change SGN(71-A(1)). We can say this in BASIC. The SGN function returns the sign of any number.

For example: SGN(−3) = −1
SGN(3) = 1
SGN(0) = 0

The SGN function returns only the values −1, 0, or 1. Thus, in BASIC, A(5)=A(2)+SGN(71-A(1)).

Similarly, Clem's new temperature A(6) will be Clem's old temperature A(3), plus the change SGN(72-A(2)) caused by Bertrand's temperature difference. In BASIC, this is A(6)=A(3)+SGN(72−A(2)).

Similarly, Arnold's new temperature A(4) is given by A(4)=A(1)+SGN(73−A(3)). Here is a program using these suggestions.

```
15 REM INITIALIZE TEMPERATURES
20 A(1)=72: A(2)=72: A(3)=72

25 REM COMPUTE NEW VALUES
30 A(5)=A(2)+SGN(71-A(1))
40 A(6)=A(3)+SGN(72-A(2))
50 A(4)=A(1)+SGN(73-A(3))

55 REM PRINT NEW TEMPERATURES
60 PRINT A(4);A(5);A(6)

65 REM REINITIALIZE TEMPERATURES
70 A(1)=A(4): A(2)=A(5): A(3)=A(6)

75 REM REPEAT THE PROCESS
80 GOTO 30
```

It is sad to relate that Clem froze solid as a snow ball early in the evening. Then, as Clem began to thaw out, Arnold's temperature rose to feverish heights. We didn't run our program long enough to determine the final outcome, but their survival is in doubt. Here are the results of our watch. Perhaps you were more patient and saw further results.

A	B	C									
73	71	72	53	83	74	73	103	51	96	80	28
74	70	73	52	84	73	74	102	50	97	79	27
74	69	74	52	85	72	75	101	49	98	78	26
73	68	75	53	86	71	76	100	48	99	77	25
72	67	76	54	87	70	77	99	47	100	76	24
71	66	77	55	88	69	78	98	46	101	75	23
70	66	78	56	89	68	79	97	45	102	74	22
69	67	79	57	90	67	80	96	44	103	73	21
68	68	80	58	91	66	81	95	43	104	72	20
67	69	81	59	92	65	82	94	42	105	71	20
66	70	82	60	93	64	83	93	41	106	70	21
65	71	83	61	94	63	84	92	40	107	69	22
64	72	84	62	95	62	85	91	39	108	68	23
63	73	84	63	96	61	86	90	38	109	67	24
62	74	83	64	97	60	87	89	37	110	66	25
61	75	82	65	98	59	88	88	36	111	65	26
60	76	81	66	99	58	89	87	35	112	64	27
59	77	80	67	100	57	90	86	34	113	63	28
58	78	79	68	101	56	91	85	33	114	62	29
57	79	78	69	102	55	92	84	32		.	
56	80	77	70	103	54	93	83	31		.	
55	81	76	71	104	53	94	82	30		.	
54	82	75	72	104	52	95	81	29		.	

War Game Problem

The kingdom of Pandab and the republic of Quat are always on the verge of war. Neither country trusts the other, and they spend most of their time and energy preparing for the possibility of war. They both follow what seems to them to be a reasonable and rational policy. Each spends a basic amount of money on defense, but adds an extra amount to take into account the armaments of the other side. Pandab spends 10 billion dollars each year, plus half of whatever the Quats spent the year before. The Quats, similarly, spend 11 billion dollars each year, plus one-third of whatever the Pandabs spent the year before.

The foreign ministers of both countries have pointed out that this shows their peaceful intentions, since any decrease in the expenditures by the other side will automatically result in a decrease in their own expenditures.

What will be the result of this common policy? Write a program to simulate the conditions.

Hint for War Game Problem

The first year Pandab spends 10 billion dollars and Quat spends 11 billion. How much will each spend the second year? Pandab will add half of 11 billion to its budget. The total spent by Pandab will be $10 + .5 \times 11 = 15.5$ billion.

The Quats will spend 11 billion dollars, plus one-third of 10 billion, for a total of 14.33 billion dollars.

The following year Pandab will spend $10 + .5 \times 14.33$, while Quat will spend $11 + .33 \times 15.5$. The computation will continue in this way in each succeeding year.

Solution to War Game Problem

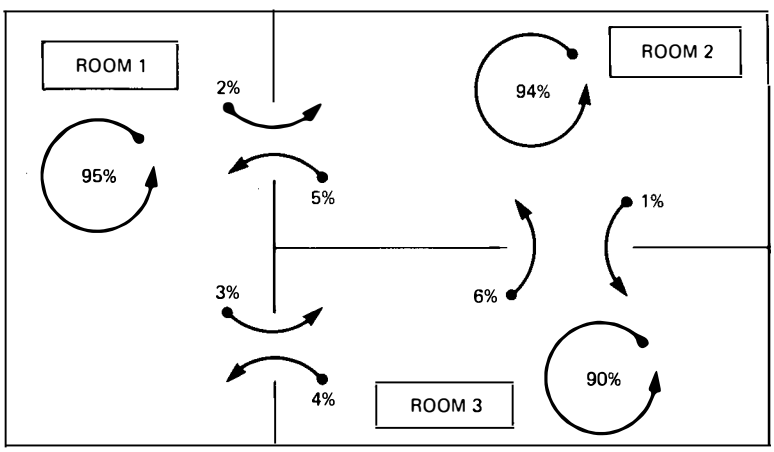
You will need to store the initial budgets. Let A(1) and A(2) contain the budgets for Pandab and Quat, respectively. You will need to compute and store the new budget for the next year. Let A(3) and A(4) contain the new budgets.

```
15 REM STORE INITIAL BUDGETS
2Ø A(1)=1Ø: A(2)=11

25 REM PRINT THE BUDGETS
3Ø USING''#####.#'':PRINT A(1),A(2)

35 REM COMPUTE PANDAB'S NEXT BUDGET
4Ø A(3) = 1Ø+A(2)/2

45 REM COMPUTE QUAT'S NEXT BUDGET
5Ø A(4) = 11+A(1)/3

55 REM REINITIALIZE
6Ø A(1)=A(3): A(2)=A(4)

65 REM DO IT AGAIN
7Ø GOTO 3Ø
```

Notice that we have formatted the print statement in line 30 to show tenths of billions of dollars. It seems ridiculous to try to print greater accuracy than that. However, more accurate figures are used to compute the results. The budgets will stabilize at 18.6 and 17.2.

Show Flow Problem

The recent ComputerTown Computer Show was a great success. Everyone came and stayed all day. There was a lot to see. There were computers that talked and computers that listened. There were computers that walked and computers that thought. There were musical computers and typing computers and computers that played games in all the colors of the rainbow. The show was held in three large rooms. It was noticed early in the morning that the people moved from room to room in a very orderly way. The diagram below shows how the people moved. The numbers show the percentage of persons in the room who flowed through the doorway in each minute. This shows, for example, that during a one-minute period, 95% of the people in Room 1 stayed there, while 2% moved to room 2, and 3% moved to room 3.



Initially, there were 1,000 people in each room. How many people will be in each room at the end of 1 hour? Do you think they might all be in the same room? Might the numbers fluctuate? Will the number in each room reach some equilibrium level? Is the final state dependent on the initial amounts in each room? A cleverly constructed program would yield the answer to these questions.

Hint to Show Flow Problem

How are you going to use all those numbers? Do a few cases by hand and see just what this is all about. How many people will be in room 1 after 1 minute? There are 1,000 people there originally, and the diagram shows that 95% stayed. This means that .95 × 1000 = 950 stayed in room 1. However, more people came in from the other two rooms. Of the 1,000 people in room 2, 5%, or 50 persons, came into room 1. Of the 1,000 people in room 3, 4%, or 40 persons, came into room 1. The total in room 1 after 1 minute is .95 × (# in room 1) + .05 × (# in room2) + .04 × (# in room 3) = 1,040.

You can see how the calculations will go. A little planning and a few decisions are now in order. You will need to store the initial numbers for each of the rooms. These will be the basis for the numbers in each room at the end of 1 minute. Store the initial numbers in rooms 1, 2, and 3 in memories A(1), A(2), and A(3). Store the computed numbers in A(4), A(5), and A(6).

Solution to Show Flow Problem

```
15 REM SET INITIAL ROOM NUMBERS
2Ø A(1)=1ØØØ: A(2)=1ØØØ: A(3)=1ØØØ

25 REM COMPUTE NEW NUMBERS
3Ø A(4)=.95*A(1)+.05*A(2)+.04*A(3)
4Ø A(5)=.02*A(1)+.94*A(2)+.06*A(3)
5Ø A(6)=.Ø3*A(1)+.Ø1*A(2)+.9Ø*A(3)

55 REM REINITIALIZE
6Ø A(1)=A(4):A(2)=A(5):A(3)=A(6)

65 REM GO COMPUTE AGAIN
7Ø GOTO 3Ø
```

Oops! We forgot something. I hope you didn't. Can you see an important line missing from this program? It would be useful to add a print statement at some appropriate line. Since we know the original values, we won't print them. Instead, we put our print statement at line 62. We also formatted the PRINT statement with a USING statement in line 22.

```
22 USING ''#####''
62 PRINT A(1);A(2);A(3)
```

This forces the print statement to ignore fractional parts of people.

Here's how the format worked out when the program was run.

```
Room 1    Room 2    Room 3

1Ø4Ø      1Ø2Ø      94Ø
1Ø76      1Ø36      887
111Ø      1Ø48      841
114Ø      1Ø58      8ØØ
1168      1Ø65      765
1194      1Ø71      734
1217      1Ø74      7Ø7
1238      1Ø77      684
1257      1Ø78      663
1275      1Ø78      645
1291      1Ø78      63Ø
13Ø5      1Ø77      616
```

If you add the three values, you can see that the total doesn't always equal 3,000 (because fractional parts are ignored). No doubt, one or two may have sneaked off to the snack bar, or the rest room. They may have even left the show. The official counter may have missed one or two as they were passing from room to room.

Well, are you ready to make any predictions from the print-outs so far? Do you think the rooms will stabilize? Try changing the PRINT statement to PAUSE. Then you can do something else for awhile and come back to check the display later.

```
62 PAUSE A(1);A(2);A(3)
```

After several minutes, we came back to the Pocket Computer. It seems to be stabilized at:

```
1446 1Ø18 535
```

We let it run awhile and found a slight change to:

```
1446 1Ø17 535
```

Presumably, someone went home. That's not bad though — 2,998 people still at the show. If you were planning a computer show, it would be nice to know such things as people flow before the show began. Then you could plan for three different-size rooms, and there would be equal space for all.

In case you are wondering, after one hour there were 1,445 people in room 1, 1,019 people in room 2, and 535 people in room 3. So, the flow had just about stabilized the count in each room.

Remember, the Pocket Computer has been using fractions of people in its calculations even though we formatted the print statement for integers. If someone were to

actually count the people in each room, they couldn't count the fractions. Therefore, our program should calculate with integers.

We could change lines 30, 40, and 50, using the integer function.

```
3Ø A(4)= INT(.95*A(1))+INT(.Ø5*A(2))+INT(.Ø4*A(3))
    etc.
```

However, this method would cut off the decimal part of our calculations completely. We would find that many people were disappearing from our count. It would be better if we rounded off the calculations to the nearest integer. Then we would get a more accurate count. We will therefore change lines 30, 40, and 50 as follows:

```
3Ø A(4)=INT(.95*A(1)+.5)+INT(.05*A(2)+.5)+INT(.04*A(3)+.5)
4Ø A(5)=INT(.02*A(1)+.5)+INT(.94*A(2)+.5)+INT(.06*A(3)+.5)
5Ø A(6)=INT(.Ø3*A(1)+.5)+INT(.Ø1*A(2)+.5)+INT(.9Ø*A(3)+.5)
```

Now, when we run the program, we see:

```
1Ø4Ø 1Ø2Ø 94Ø
1Ø77 1Ø36 887
1119 1Ø49 84Ø
1141 1Ø58 799
1169 1Ø66 764
1195 1Ø71 734
1218 1Ø75 7Ø8
```

You can see that these figures are similar to the first approach, but not quite the same.

If we let the program run until the room counts stabilize, we see the display:

```
1442 1Ø25 535
```

So, you see, final results of a computer program depend quite a bit on methods that the programmer uses.

Summing Up Chapter Seven

All the programs in this chapter have been feedback systems that perform calculations and feed the results back to the beginning of the program. They are recalculated and fed back over and over again. Many real life situations are similar. We make decisions based on the result of previous situations. A computer is very good at solving problems that can be organized in this way.

Problem solving becomes much simpler when you can separate the problem into small steps, decide how each step can be accomplished, and see how each step fits into the complete solution.

You used the same, or similar, techniques to solve each of the problems in this chapter. Although the problems appeared quite different on the surface, the same methods were used to reach the varied solutions.

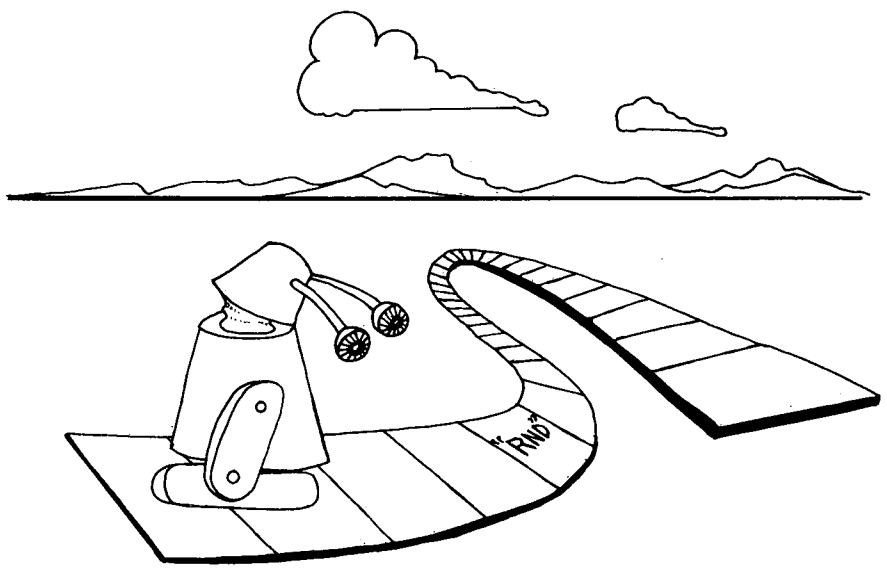
CHAPTER EIGHT

Random Walk

Numbers occur in this world in numerous quantities and styles. Quite often they occur at random. Sometimes numbers appear to be *equally* random. Other times they seem to be random within certain areas or shapes, such as a "normal" distribution.

This chapter looks at the randomness of numbers in many ways. In this chapter, you will learn how to:

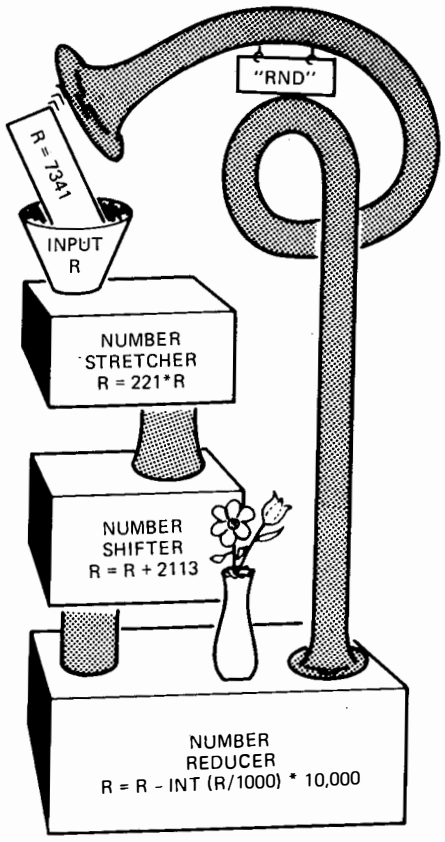
- generate random numbers to simulate coins and dice;
- describe different kinds of randomness;
- sort numbers into a histogram; and
- use random numbers to simulate real events.



Random Number Generator Problem

How do you change a 4-digit number, R, into another 4-digit number in a "random" manner? Here is one recipe:

1. Stretch it by 221. (221 times the original number)
2. Shift it by 2,113. (add 2,113 to the result of 1)
3. Reduce it by multiples of 10,000. (subtract INT(R/10000) × 10000)



In BASIC, this may be written as:

```
R=221*R+2113
R=R-INT(R/10000)*10000
```

The new value of R is remarkably independent of the old value of R.

Write a program that allows the user to input an initial value for R, then computes and prints out new values for R in a "random" manner using the recipe above.

Hint to a Random Number Generator

This will be an often used program. You should write it in such a way that it can be easily used as a subroutine in other programs. Write the subroutine starting at line 500 with a label, "RND". Here is how the top of the program might look.

```
15 REM INPUT AN INITIAL VALUE FOR R
2Ø INPUT R

25 REM GET A RANDOM NUMBER R
3Ø GOSUB ''RND''

4Ø PRINT R
```

Complete the program so that an indefinite number of random numbers will be printed. The initial value for R at line 20 only has to be executed once. The subroutine will produce a series of 4-digit random numbers.

Solution to a Random Number Generator Problem

```
15 REM INPUT AN INITIAL VALUE FOR R
2Ø INPUT R

25 REM GET A RANDOM NUMBER R
3Ø GOSUB ''RND''

4Ø PRINT R

45 REM DO IT AGAIN
5Ø GOTO 3Ø
6Ø END

5ØØ ''RND'':REM RANDOM GENERATOR
51Ø R=221*R+2113
52Ø R=R-INT(R/10000)*10000
53Ø RETURN
```

We input the initial value of 1,234 for R and saw the following random numbers appear.

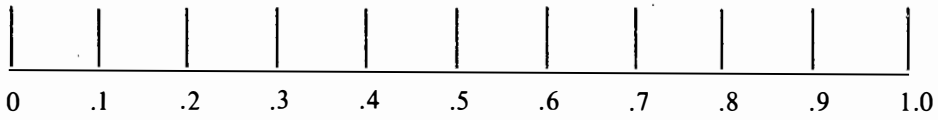
```
4827      3692      4Ø57
888Ø      8Ø45      871Ø
4593      ØØ58      7Ø23
7166      4931      4196
5799      1864      9429
```

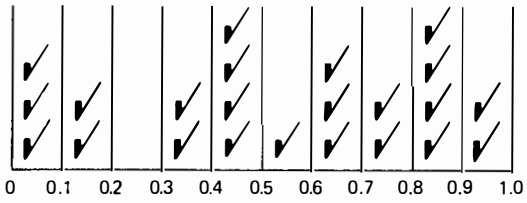
If we stop the program and run it again using the same value (1,234) as the initial value of R, we will see the same random numbers displayed in the same order. But, if we input a different initial value for R, we will get a new and different list of random numbers.

Try this program with several different inputs.

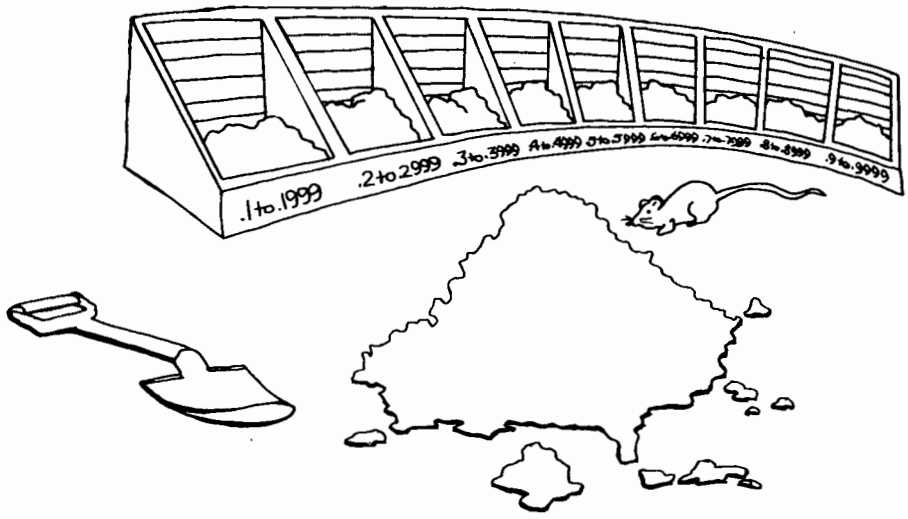
Random Tests Problem

In the last problem, you saw the devious ways of the random number generator. You ought now to be responsible and craftsmanlike, and test the random generator to see if it does really generate randomly. Wouldn't you like to generate a few numbers using the "RND" subroutine and sort them out by size? Even if you said no, you *should* try it anyway. Here is a chart to fill in. Just make a check mark in the appropriate box for each random number that you generate. Notice the boxes are for sorting decimals between 0 and 1. Write your program so that it prints random numbers between 0 and 1.





Hint to Random Tests



Here's what we got when we sorted out the first 23 random numbers that were generated. The first random number that our generator gave us was R/10000 = .4678. This number is between .4 and .5 so a check was put in the appropriate box.

Solution to Random Tests Problem

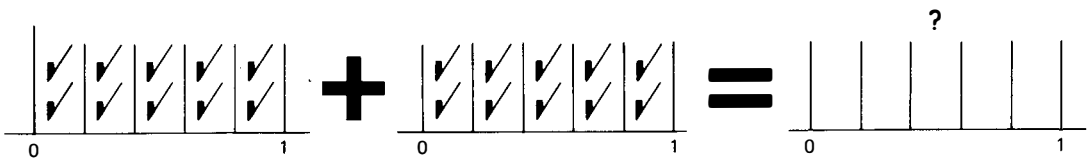
The check marks should tend to fall into all boxes with the same probability. This shows that the numbers being generated are really random in the sense that any number is as likely as any other number to occur. The random numbers which are generated should be uniformly distributed between 0 and 1. With such a small sample of numbers (23), our distribution is not truly uniform. A much larger sample would be needed to provide a good test. Each four place decimal number between 0 and 1 should be as likely as any other. This routine furnishes us with a standard, uniformly distributed random variable.

```
15 REM INITIALIZE ''RND''
2Ø INPUT R
3Ø GOSUB ''RND''
4Ø PRINT R/1ØØØØ

45 REM GET ANOTHER NUMBER
5Ø GOTO 3Ø

5ØØ ''RND''
51Ø R=221*R+2113
52Ø R=R-INT(R/1ØØØØ)*1ØØØØ
53Ø RETURN
```

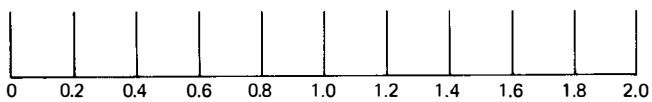
After you have thoroughly tested the random number generator, you'll undoubtedly use it in future programs.

Some Triangle Problem

What do you suppose happens when two uniform variables are added together?



Write a program to add two "standard," random numbers. Place the results in the following boxes.



Hint to Some Triangle Problem

Be sure you input an initial value for R, then some lines that look something like this:

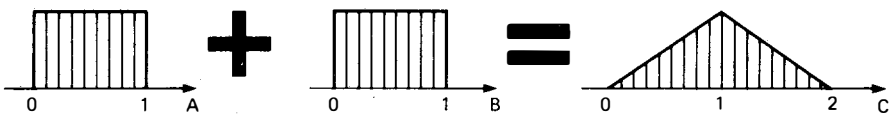
```
GOSUB ''RND'': A=R/1ØØØØ
GOSUB ''RND'': B=R/1ØØØØ
C=A+B
PRINT C
```

Don't forget to add line numbers and insert the random number generator.

Solution to Some Triangle Problem

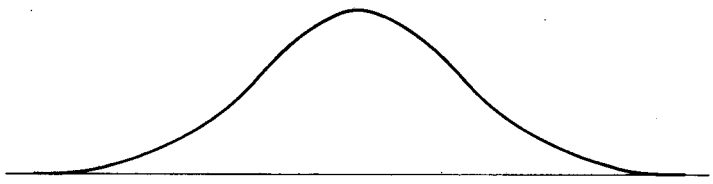
```
15 REM INITIALIZE ''RND'' WITH 4 DIGIT NUMBER
2Ø INPUT R

3Ø ''BEGIN''

35 REM GET TWO RND NUMBERS
4Ø GOSUB ''RND'': A=R/1ØØØØ
5Ø GOSUB'''RND'': B=R/1ØØØØ

55 REM COMPUTE AND PRINT SUM
6Ø C=A+B
7Ø PRINT C

75 REM DO IT AGAIN
8Ø GOTO ''BEGIN''
9Ø END

5ØØ ''RND''
51Ø R=221*R+2113
52Ø R=R-INT(R/1ØØØØ)*1ØØØØ
53Ø RETURN
```

The random numbers A and B, and their sum C, tend to look like this:



Yours might not look exactly like this, but the random generator will be trying.

What do you think happens when three uniform variables are added? Maybe this?

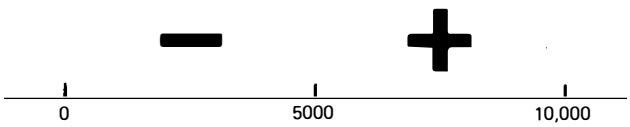


Plus or Minus Problem

Back and forth, up and down, in and out, randomly coming and going. What we need now is a short subroutine, called "SIGN", that returns a random variable S, which is equal to +1 or −1 with equal probability. Number the "SIGN" subroutine starting with line 600. Include a GOSUB for "RND".

Hint for Plus or Minus

Get a 4-digit random number R from "RND". R will be a number between 0 and 10,000, such as 5,713. Divide the numbers as shown here.



Solution to Plus or Minus Problem

```
600 ''SIGN''
610 GOSUB ''RND'': REM R IS NOW DEFINED
620 IF R>= 5000 LET S=1
630 IF R<5000 LET S=-1
640 RETURN
```

Don't forget "RND" at lines 500 through 530.

Lines 620 and 630 can be simplified a bit. Here is another way to get S.

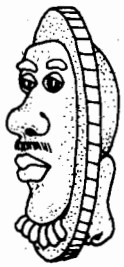
```
620 S=1
630 IF R<5000 LET S=-1

     or

620 S=SGN(R-5000)
630 IF S=0 LET S=1

     or

620 IF R>=5000 LET S=1:GOTO 640
630 S=-1
```

Coin Flop Problem

A coin comes in one of two states, heads or tails. This has been useful, entertaining, and instructive for humankind throughout recorded history. A coin can tell a fortune or win a fortune. A coin can send a message or make a decision.

```
0 0 1 0 1 1 1 0 1 0 0 1 0 1 0 0 0 1 1 1 0
```

(Coin messages tend to be all noise and no message.) There is much to be learned from coins.



Write a program which prints out a random sequence of 0's and 1's using the "RND" subroutine. Make it a subroutine called "COIN" beginning at line 550. It should return a variable, C, which is randomly equal to 1 or 0.

Hint to Coin Flop Problem

It looks like a random number generator is needed. How can we use our 4-digit generator to set random 0's or 1's?

Generate random numbers with the "RND" subroutine. Call "RND" with the GOSUB command GOSUB "RND". Remember, "RND" has been used from lines 500 through 530 so it will be out of the way of "COIN".

"RND" will return a 4-digit number R. Now you need to do something with R to get 0's and 1's. You could set C = 1, when R $\geq$ 5000.

Solution to Coin Flop Problem

Get a random 4-digit number from "RND". Check the size of R to get C. There are several ways you could do this.

```
IF R>= 5ØØØ LET C=1
IF R<5ØØØ LET C=Ø
  or
C=1
IF R<5ØØØ LET C=Ø
  or
C=(5ØØØ<R)
```

We used the last one. Notice that this subroutine will need to call "RND". Append "RND" to the program. The complete program with documentation removed looks like this:

```
15 REM INITIALIZE R
20 INPUT R

30 GOSUB ''COIN''
40 PRINT C
50 GOTO 30

500 ''RND''
510 R=221*R+2113
520 R=R-INT(R/100000)*100000
530 RETURN

550 ''COIN''
560 GOSUB ''RND''
570 C=(5000<R)
580 RETURN
```

As a result of 20 coin flops, we saw this sequence of zeros and ones on the display.

```
0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 1
```

10 zeros and 10 ones — we were lucky.

Gambler's Ruin Problem

Gambling was invented right after the coin. (It was probably a good thing: It helps to redistribute the coins.) Of one thing you may be sure; if you play long enough, you'll run out of money. The length of time it takes to run out of money in a coin tossing game illustrates the nature of the gambler's ruin problem.

Write a program that simulates a gambler who starts with five pennies. On each toss of the coin, the gambler loses a penny if it comes up tails (−1) and wins a penny if it comes up heads (+1). How long does it take the gambler to go broke?

Hint to Gambler's Ruin

Use the "SIGN" subroutine to generate +1 and −1 at random. A running total will keep track of the number of coins that the gambler has left.

Solution to Gambler's Ruin Problem

```
10 INPUT R: REM INITIALIZE ''RND''
20 T=5: REM GAMBLER STARTS WITH 5

25 REM GET RANDOM S= +1,-1
30 GOSUB ''SIGN''

35 REM KEEP TRACK OF TOTAL PENNIES
40 T=T+S

45 REM PRINT TOTAL AND CHECK IF BUSTED
50 PRINT T
60 IF T>0 THEN 30
70 PRINT ''BUSTED'':END

500 ''RND''
510 R=221*R+2113
520 R=R-INT(R/10000)*10000
530 RETURN

600 ''SIGN''
610 GOSUB ''RND''
620 S=SGN(R-5000)
630 IF S=0 LET S=1
640 RETURN
```

To see how quick the gambler's ruin is reached, we ran the program with these results.

First run: 4 3 2 1 0 BUSTED

Second run: 6 5 4 5 4 3 2 1 BUST ED

Third run: 6 5 6 7 6 5 6 7 8 9 8 9 8 9 10 11 10 11 10 11 10
11 12 13 12 13 12 11 10 11 12 11 12 13 12 13 12
11 12 13 14 15 16 17 18 17 18 19 20 21 22 21 20
19 20 19 20 19 20 20 20 21 22 21 22 23 .....

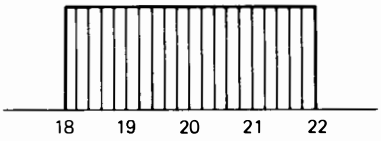
We gave up! If we had kept going, would the gambler have ever busted?

How to Add When You Don't Have the Numbers

You may have noticed that good numbers are hard to find. Error, ignorance, and misinformation are everywhere. What is one to do? Decisions are necessary and plans must be made, even though we don't have all the facts. There must be some way to take our ignorance into account.

Here is a common situation. Your car gets *around* 20 miles per gallon. The vacation trip that you want to make will be *about* 500 miles. *About* how much gas will it take?

You don't know all the facts exactly; but you do know something. What you know is hard to describe. Let's try to describe one of your fuzzy facts. What is it that you know when you say your car gets *around* 20 miles per gallon? Sometimes it gets 18, sometimes it gets 22, sometimes it gets something in between. A picture describes the situation.



Any of the numbers between 18 and 22 could be your actual mileage. The number that actually occurs is largely a matter of chance. The mileage is a random number between 18 and 22. You can simulate mileage numbers using the "RND" subroutine listed earlier in the book.

Random Mileage Problem

Can you write a program which will generate random numbers between 18 and 22 so that each decimal number is as likely as any other?

Hint to Random Mileage Problem

The subroutine "RND" generates numbers between 0 and 9,999. You need decimal numbers between 18 and 22. You can generate numbers between 0 and 4 by dividing each number from "RND" by 2,500. Now, if you add 18 to these numbers, you will have what you want.

a.
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____

---

a.

```
10 REM START THE RANDOM GENERATOR
20 INPUT R

30 REM GO GET A RANDOM NUMBER
40 GOSUB ''RND''

50 REM ADJUST NUMBER FROM ''RND''
60 M = R/2500+18

70 PRINT M

80 REM GO DO IT AGAIN
90 GOTO 40
```

You could use the previous method to generate random numbers that simulate the distance D that you will cover on your trip. Suppose the trip will be between 450 and 550 miles. Line 60 would look like this:

```
6Ø D = R/1ØØ+45Ø
```

Now that you are so good at generating random numbers, perhaps you would like to finish the problem that we started.

Random Gas Problem

Your car mileage is a random number M that varies uniformly between 18 and 22. The distance to be traveled is a random number D that varies uniformly between 450 and 550. Write a program to generate random numbers G which correspond to the total gallons of gasoline used on the trip.

Hint to Random Gas

Miles per gallon times gallons equals miles. Put another way, $M \times G = D$. What you need is the random number for G.

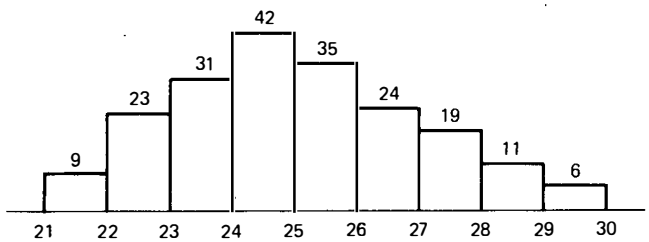
a.
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____
100 _____
110 _____
120 _____
130 _____

500 "RND"

(Random subroutine goes here)

a.
```
10 REM START THE RANDOM GENERATOR
20 INPUT R

30 REM COMPUTE RANDOM DISTANCE D
40 GOSUB ''RND''
50 D = R/100+450

60 REM COMPUTE RANDOM MILEAGE M
70 GOSUB ''RND''
80 M = R/2500+18

90 REM GENERATE G
100 G = D/M

110 PRINT G

120 REM GO DO IT AGAIN
130 GOTO 40
```

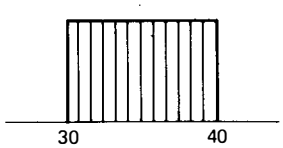
We ran this program, and generated 200 numbers. The numbers we got ranged from 21 up to 30. When we sorted them out, we got the following distribution.



This distribution tells us a great deal. Although you can expect to use about 25 gallons, the actual amount may deviate from 21 gallons or, on rare occasions, to nearly 30 gallons.

Open House Problem

Parties are hard to plan, and an open house is nearly impossible. The main trouble is that you never know just how many people will show up. Even if you know how many people are coming, you won't know how much they will eat or drink. Let's suppose that the number of people coming is a random number between 30 and 40, whose distribution looks like this:



Let's also suppose that the number of crackers eaten by each person is a random number between 10 and 15, whose distribution looks like this:

Write a program to investigate the number of crackers required for the party. Do you think 500 crackers will be enough?

Hint to Open House

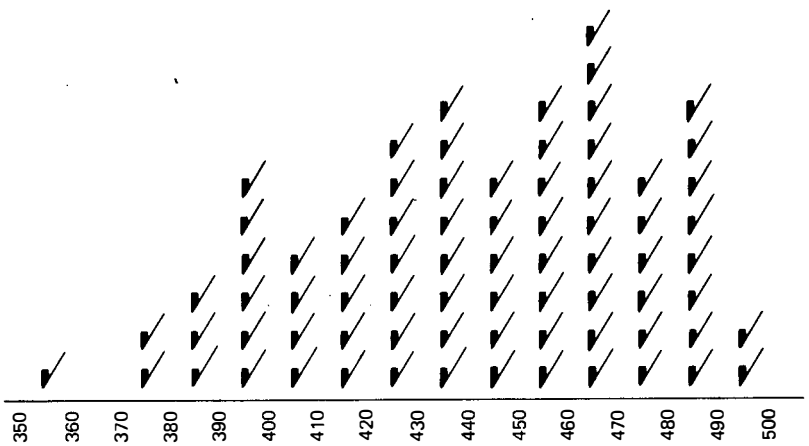
You will need to generate a random number P to simulate the number of people who come to the party, and you will need a random number C to simulate the number of crackers eaten by each person who comes. For example, if P = 35, then you will need to generate 35 random numbers C to simulate the crackers eaten by each of the guests. The sum of these cracker numbers is what we need to know.

a.
```
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170

500 "RND"
        (Add the "RND" subroutine here)
```

a.
```
10 REM INITIALIZE ''RND''
20 INPUT R

30 REM GET RANDOM P
40 GOSUB ''RND''
50 P = R/1000+30

60 REM SET ACCUMULATOR TO ZERO
70 S = 0

80 REM GET P RANDOM C's
90 FOR Z = 1 TO P
100 GOSUB ''RND''
110 C = R/2000+10

120 REM ADD UP THE C's
130 S = S+C
140 NEXT Z

150 BEEP(2): PRINT S

160 REM GO HAVE ANOTHER PARTY
170 GOTO 40

500 ''RND''
```

(Add the "RND" subroutine here)

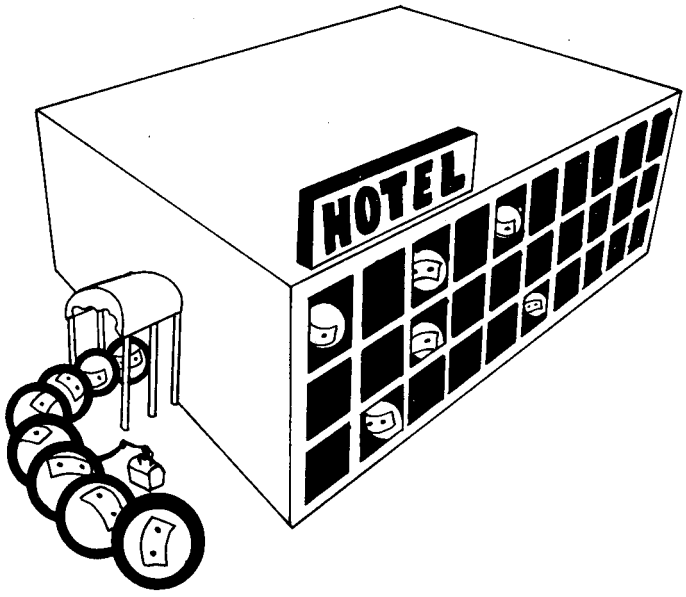
We ran this program to get 75 numbers (P). Here is how the numbers stacked up in a histogram. It looks like 500 crackers will be plenty.



Electron Hotel Problem

Nearly everything has a battery in it these days; wristwatches and golf carts, toothbrushes and TV sets, not to mention the occasional horseless carriage.

A battery, as any child knows, is a hotel for electrons. The electrons line up patiently, waiting to enter.



The first electron in line enters and checks a room at random. If the room is empty, then the electron occupies the room. If the room is not empty, then the electron must go back to the front of the line and try again. (We recently looked at a battery with a large magnifying glass and can assure you of the veracity of this report.)

Write a program that will simulate the electron hotel problem. You are, no doubt, wondering just how many tries it will take to fill 9 rooms in a 10-room hotel.

Hint for Electron Hotel Problem

There are 10 numbered rooms. A random room number is chosen. The room is checked to see if it is empty or not. If the room is empty, then it is filled. If the room is not empty, then the process is repeated; a room is chosen at random and checked.

Let the rooms correspond to memories. Let the state of the room correspond to the state of the memory.

Solution to Electron Hotel Problem

```
 15 REM INITIALIZE ''RND''
 2Ø INPUT R

 25 REM INITIALIZE COUNTERS
 3Ø T=Ø: W=Ø

 35 INITIALIZE MEMORIES
 4Ø FOR Z=1 TO 1Ø:A$(Z)=''E'':NEXT Z

 5Ø ''NEWTRY''
 6Ø T=T+1

 65 REM GET A NUMBER, 1 TO 1Ø
 7Ø GOSUB ''NUMBER''

 75 REM CHECK IF FULL OR EMPTY ROOM
 8Ø IF A$(N)=''F'' GOTO ''NEWTRY''

 85 REM ROOM IS EMPTY, FILL IT
 9Ø A$(N)=''F''

 95 REM INCREMENT FILL COUNTER
1ØØ W=W+1

1Ø5 REM CHECK IF 9 ROOMS ARE FULL
11Ø IF W>8 GOTO ''PRINT''

115 REM IF 9 ARE NOT FULL TRY AGAIN
12Ø GOTO ''NEWTRY''

13Ø ''PRINT''
14Ø PRINT ''TRIES '';T
15Ø GOTO 30

2ØØ ''NUMBER''
21Ø GOSUB ''RND''
22Ø N = INT(R/1ØØØ)+1)
23Ø RETURN
```

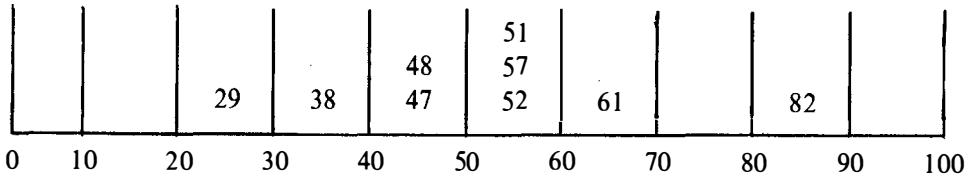
(Append "RND" subroutine)

Histogram Problem

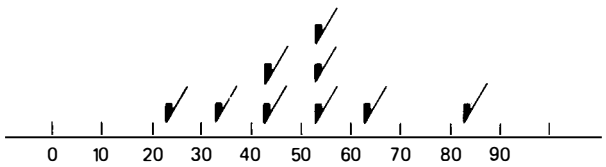
Numbers pour in upon us. Heaps of numbers, swelling oceans of numbers. It's the plague of the age. We at last have more information than we can deal with. The simplest and most useful thing you can do with a string of numbers is to sort it out by size. Here is a string of numbers:

47, 61, 52, 48, 57, 29, 51, 82, 38

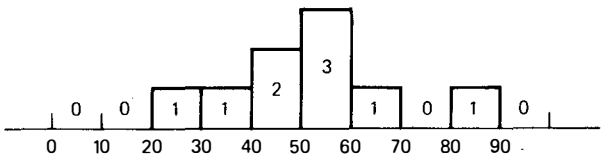
Here is the string sorted into boxes by size. The sorting boxes are 10 numbers wide. For example, any number between 40 and 49 goes into the same box. Any number between 50 and 59 goes into the next box, and so on.



It is quicker and easier to use check marks instead of writing all the numbers down. This is sometimes called a *histogram.*



We are most interested in the number of checks in each box. If we count the numbers in the boxes we get:



Write a subroutine called "HISTOGRAM" that starts at line 800 and allows the user to input a sequence of numbers between 0 and 100 (but not including 100). The program should keep count of the number of occurrences in each interval. When a negative number is input, the program should print out the number of occurrences in each interval and end. The output for the previous example would be:

```
Ø, Ø, 1, 1, 2, 3, 1, Ø, 1, Ø
```

Hint to Histogram Problem

It is always a good idea to make a program as foolproof as possible. The program takes numbers between 0 and 100 as input. Someone will certainly input 397,521 or some other very large number. What happens then? You had better put in a test to make sure the input is in the proper range.

It will also be necessary to make some decisions. You will need to keep track of the numbers in each box. Where shall you store the 10 numbers? Let's agree on A(1) through A(10). These will be counters. What shall you call the input? Use the variable Z. Now, all that's left for you to do is figure out how to test Z to get its box number and add 1 to the number in that box.

Solution to Histogram Problem

Consider the heart of the program first. How are you going to test the number Z and determine what box it should go into? You can make this a subroutine and give it some appropriate name, like "FINDBOX". Here is one possible approach:

```
200 ''FINDBOX''
205 REM FINDS BOX NUMBER Y FOR NUMBER Z
210 IF 90<=Z LET Y=10:GOTO 310
220 IF 80<=Z LET Y=9:GOTO 310
230 IF 70<=Z LET Y=8:GOTO 310
240 IF 60<=Z LET Y=7:GOTO 310
250 IF 50<=Z LET Y=6:GOTO 310
260 IF 40<=Z LET Y=5:GOTO 310
270 IF 30<=Z LET Y=4:GOTO 310
280 IF 20<=Z LET Y=3:GOTO 310
290 IF 10<=Z LET Y=2:GOTO 310
300 Y=1
310 RETURN
```

Here is another method using a FOR-NEXT loop. It takes fewer lines, but must search through each 10 conditions for each number. The first version simply searches until the proper box is found and then goes to the end of the routine. Which method is faster?

```
200 ''FINDBOX''
205 REM FINDS NUMBER Y FOR NUMBER Z
210 FOR X=1 TO 10
220   W=10*(X-1)
230   V=(W<=Z)*(Z<W+9)
240   IF V=1 LET Y=X
250 NEXT X
260 RETURN
```

logic AND function

Here is another possibility:

```
200 ''FINDBOX''
205 REM FINDS BOX NUMBER Y FOR NUMBER Z
210 FOR X=1 TO 10
220 IF Z<10*X LET Y=X:X=10
230 NEXT X
240 RETURN
```

To break out of the FOR-NEXT loop

One last possibility:

```
200 ''FINDBOX'': Y=INT(Z/10)+1: RETURN
```

Using the last method, here is how a finished version of the whole program looks.

```
15 REM INITIALIZE MEMORY
20 FOR Z=1 TO 10: A(Z)=0: NEXT Z

25 REM INPUT A NUMBER BETWEEN 0 AND 100
30 INPUT Z

35 REM TEST Z
40 IF Z<0 GOTO ''PRINT''
50 IF Z>=100 BEEP!3):PRINT''TOO BIG, PRESS ENTER'':GOTO 30

55 REM GET BOX NUMBER Y
60 GOSUB ''FINDBOX''

65 REM INCREMENT THE BOX
70 A(Y)=A(Y)+1

75 REM INPUT ANOTHER NUMBER
80 GOTO 30

200 ''FINDBOX'':Y=INT(Z/10)+1:RETURN

300 ''PRINT''
305 REM PRINT RESULT AND END
310 FOR Z=1 TO 10: PRINT A(Z):NEXT Z
320 END
```

Fox and Rabbits Program

On a certain island lives 20 rabbits and a single fox. The rabbits each have their own territory, and there is one rabbit per territory. The fox hunts in exactly one territory per day, at random, in the 20 territories.

Write a program to simulate this situation and print out the total number of rabbits caught after any number of days. Assume that there are very atypical rabbits that do not reproduce. The fox also doesn't realize that if it has already caught a rabbit in a given territory, he will not catch another rabbit there. So, some days he will not make a catch. About how long will it take the fox to catch 18 of the rabbits?

Hint to Fox and Rabbits Problem

The first thing to do is make a decision as to how the rabbit territories are to be represented by the computer's memories. You need 20 memory registers. Each will record whether a rabbit is there (1) or not (0). Use A(31) through A(50) to record the state of the 20 rabbit territories.

The second thing to do is devise a random number subroutine that returns numbers from 31 through 50. The random number will simulate the foxes random choice of a territory.

You will also need counters for the day and for the number of rabbits caught. And don't forget the printout.

Solution to Fox and Rabbits Problem

```
15 REM INITIALIZE ''RND'' Ø<R<1ØØ ØØ
2Ø INPUT R

25 REM INITIALIZE TERRITORIES
3Ø FOR Z=31 TO 5Ø:A(Z)=1:NEXT Z

35 REM INITIALIZE DAY COUNTER
4Ø D=Ø:C=Ø
45 REM GET A RANDOM NUMBER R
5Ø GOSUB ''RND''

55 REM ADJUST R TO T THE TERRITORY
6Ø T=INT(R*2Ø/1ØØØØ+31)

65 REM INCREMENT DAY COUNTER
7Ø D=D+1

75 REM IF IT'S THERE CATCH THE RABBIT
8Ø IF A(T)=1 LET A(T)=Ø: C=C+1

85 REM PRINT THE DAY'S RESULTS
9Ø PRINT D,C

95 REM GO TO NEXT DAY
1ØØ GOTO 5Ø

2ØØ ''RND''
21Ø R=221*R+2113
22Ø R=R-INT(R/1ØØØØ)*1ØØØØ
23Ø RETURN
```

Short Story Problem

Have you ever wanted to be a writer? All you need to do is put words together to get sentences. Once you've gotten enough sentences, you've got a book. Soon thereafter, fame and fortune will knock on your door. Since all that is needed is a goodly supply of sentences, perhaps you ought to see if you can mechanize the process.

Write a program which picks one word from each of the lines that follow to make a complete sentence.

0. A, THE, THIS, THAT
1. TALL, SHORT, HAPPY, SAD
2. BLONDE, CHINESE, FRENCH, MEXICAN
3. MAN, WOMAN, BOY, GIRL
4. RAN, WALKED, CRAWLED, DANCED
5. SLOWLY, QUICKLY, WILDLY, NAKED
6. ON, OVER, THROUGH, AMONG
7. THESE, DRY, THE, WET
8. GRASSES, LEAVES, ROCKS, FLOWERS

Hint for Short Story Problem

The first thing you need is to decide where to put all those words. Store them in memories A$(30) through A$(65). A$(30)="A"; A$(31)="THE"; A$(32)="THIS"; and so on until A$(65)="FLOWERS".

To choose a word from line 0, you need to choose a random number (I) from the set 0,1,2,3. The word you choose is at A$(30+I). To choose a word from line 1, you need another random number (I). The word you choose is at A$(30+4+I). In a similar way, you choose a word from line 6 at A$(30+4*6+I).

Solution to Short Story Problem

```
15 REM INITIALIZE ''RND''
2Ø INPUT R

25 REM INPUT WORD LIST
3Ø GOSUB ''WORDLIST''

35 REM CHOOSE A WORD FROM EACH LINE
4Ø FOR L=Ø TO 8

45   REM GET I (Ø-3)
5Ø   GOSUB ''INTEGER''

55   REM PRINT SELECTED WORD
6Ø   PAUSE A$(3Ø+4*L+I)
7Ø NEXT L

75 REM LEAVE BLANK AFTER SENTENCE
8Ø PAUSE '' '' :BEEP(1)

85 REM GET ANOTHER SENTENCE
9Ø GOTO 4Ø

2ØØ ''WORDLIST''
21Ø FOR Z=3Ø TO 65
22Ø   PRINT ''WORD'';Z
23Ø   INPUT A$(Z)
24Ø NEXT Z
25Ø RETURN

5ØØ ''RND''
51Ø R=R*221+2113
52Ø R=R-INT(R/1ØØØØ)*1ØØØØ
53Ø RETURN

8ØØ ''INTEGER''
81Ø GOSUB ''RND''
82Ø I=INT(R*4/1ØØØØ)
83Ø RETURN
```

Summing up Chapter Eight

This chapter contained a wide variety of problems demonstrating the use of random numbers. So many things in our everyday lives seem to occur randomly that we have only touched on the TRS-80 Pocket Computer's use in this type of application.

A variety of programming techniques were presented, but, once again, we have only touched on the possibilities in our solutions to the problems. No doubt, your solutions were different from ours. That's the beauty and excitement of problem solving. There are many ways to reach a solution. Who is to say which method is best?

CHAPTER NINE

Computing Interest

Money was invented just a moment before borrowing and lending. It never seems to be in exactly the right place at the right time. What to do? Borrow until you strike it rich. Lend until you need to spend.

And, of course, the borrowers always have to make some promises to the lenders. The chief promise is to give the money back. Another promise is to give back a lot more money than was borrowed. That's the way it is. There is not a lot of charity among lenders.

Computing interest can become a complex problem. You'll definitely find the problems challenging. In this chapter you will learn:

- how to compute compound interest;
- how to compute the value of an annuity;
- how to compute time payment size;
- how to compute repayment time;
- how to compute continuous interest;
- how to compute time payment schedules; and
- how to compute the present value of a cash flow.

Compounding Your Interest

Suppose you invest $1,000 in an account that pays 2% each month. At the end of the month, you will have $1,000, plus 2% of $1,000. That's $1,020. Next month, you will collect 2% interest on the $1,020. Therefore, at the end of the second month, you have $1,040.40 in the bank. Why not write a program to see how fast your money grows. How long would it take for your money to double its original size at this rate? Remember, your interest is compounding each month at 2% per month.

When you write a program to answer the previous question on money doubling, let the variable A represent the amount that you deposit in the bank account. To compute a new value for A, increase the old value by 2%.

New amount = Old amount, plus 2% of the old amount

Or in BASIC, $A = A + .02 \times A$

In BASIC, the right side of an equality is calculated. The result is then assigned to the variable on the left side. Therefore, the old amount, A, is multiplied by .02 and added to the old amount. This calculated value is then placed in the memory assigned to A.

Your program:

a.
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____

---

a.
```
10 REM A IS AMOUNT IN ACCOUNT
20 A = 1000

30 REM ADD INTEREST TO GET NEW A
40 A = A + .02*A

50 PRINT A

60 REM GO DO IT AGAIN
70 GOTO 40
```

When we ran this program we found that after 35 months the amount in the account was a fraction of a cent under $2,000.

Yacht Account

Catherine owns the ComputerTown Coffee Shop. She is saving her money to buy a racing yacht. At the end of each month, she takes $200 to the bank and deposits it in her yacht account. The money she deposits earns interest, which is added to her account. The amount of interest that the bank adds to her account is 2% of the amount in her account during the preceding month.

How long will it take Catherine to accumulate the $100,000 necessary to start her racing career?

Hint to Yacht Account Problem

Let's compute a few cases to see how Catherine's money grows month by month.

Month	Amount	+	2%×Amount	+	Deposit	=	New Amount
0	0	+	.02 × 0	+	200	=	200
1	200	+	.02×200	+	200	=	404
2	404	+	.02×404	+	200	=	612.08

To write a program to solve Catherine's problem, you will need to store some numbers. In particular, you will need to store the amount A, which is the present balance in her account. At the end of each month, you will compute a new A. It can be stored in the same place as the previous amount. You will also need to keep track of the months. Use the variable I to count the months.

a.
```
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____
100 _____
```

---

a.
```
10 REM I=MONTHS, A=ACCOUNT AMOUNT
20 I = 0: A = 0

30 REM COMPUTE NEW A
40 A = A + .02*A + 200

50 PAUSE I,A

60 IF A >= 100000 END

70 REM INCREMENT MONTH
80 I = I+1

90 REM DO IT AGAIN
100 GOTO 30
```

Time Payment Problem

If your credit is good, the Friendly Car Dealer will let you take home that sleek beauty with the chrome hubcaps. Of course, there will be modest monthly payments and a bit of interest on the unpaid balance.

Can you write a program to keep track of your debt? Let's say the car costs $10,000. The interest rate is 2% per month on the unpaid balance. Your monthly payments are to be $250. The program should print the amount still owed each month after interest has been added and the payment subtracted.

Hint to Time Payment

Let's do a few month's worth of calculations to see how it goes.

Month	Amount Owed	Amount Plus Interest	Amount Plus Interest Minus Payment
1	10,000	10,200	9,950
2	9,950	10,149	9,899
3	9,899	10,097	9,847

We started off with the principle amount (A=$10,000). Next, we increased A by 2% to get $A + .02 \times A$. Then we subtracted the payment of $250 to get $A + .02 \times A - 250$. This is the amount that is owed after the interest has been added and the payment deducted. This amount is owed during the next month. We call this amount A for the next round of calculations.

Write a program to calculate the amount owed at the end of each month. Print the result.

a.
```
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____
```

- - - - - - - - - - - - - - - -

a.
```
10 REM PRINCIPLE AMOUNT A
20 A = 10000

30 REM ADD INTEREST
40 A = A + .02*A

50 REM SUBTRACT PAYMENT
60 A = A-250

70 PRINT A

80 REM DO IT AGAIN
90 GOTO 40
```

It is easy to make this program more general. You can modify the program so that the user can input the principle amount, the interest, and the monthly payment.

```
10 INPUT ''AMOUNT?'',A
20 INPUT ''RATE?'',R
30 INPUT ''PAYMENT?'',P

40 REM COMPUTE NEW AMOUNT
50 A = A+R*A-P

60 PRINT A

70 GOTO 50
```

Input R as a decimal (.02 for 2%)

There is another approach to this problem. If you wrote out the arithmetic in the right way, you would notice a nice pattern. Here is what it looks like:

Month	Amount + Interest − Payment
1	A * (1 + R) − P
2	A * (1 + R)∧2 − P*(1 + (1 + R))
3	A * (1 + R)∧3 − P*(1+(1 + R)+(1 + R)∧2)
.	
.	
.	
N	A * (1 + R)∧N − P(1+(1 + R)+(1 + R)∧2+...+(1 + R)∧(N − 1))

A bit of algebraic cleverness suffices to show that the balance B owed after N months simplifies to:

$$B = ((A * R - P) * (1 + R)\wedge N + P)/R$$

Not simple, but simpler than performing each step. You can use this expression to write a program that allows you to avoid all the intermediate steps and to print the amount owed after any number of months, N.

Your program:

a. 10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____

---

a.
```
10 INPUT ''AMOUNT?'',A
20 INPUT ''RATE?'',R
30 INPUT ''PAYMENTS?'',P
40 INPUT ''HOW MANY MONTHS?'',N

50 REM AMOUNT OWED AFTER N MONTHS
60 B = ((A*R-P)*(1+R)^N+P)/R

70 PRINT ''AMOUNT OWED IS $'';B

80 GOTO 10
```

R is inserted as a decimal (.02 for 2%)

The Time Payment Equation

The time payment equation relates the balance B to the initial amount A, the interest rate R per time period, the payment P per time period, and the number T of time periods that have passed.

$$B = ((A \times R - P) \times (1 + R) \wedge T + P)/R$$

This equation guides a large part of all economic activity. Most consumer credit involves this equation.

The time payment equation can be rearranged to get other important formulas. If N is the number of payments necessary to pay off an initial debt of amount A, then it must be true that the balance B will be equal to zero after N periods. This observation leads eventually to the next equation.

The Payment Size Equation

$$P = A \times R/(1 - (1 + R) \wedge (-N))$$

This equation gives the periodic payment P in terms of the initial amount A, the interest rate R, and the number N of time periods to achieve repayment.

Payment Size Problem

Write a program called "P" that prompts the user to enter the initial amount A, the number N of time periods to achieve repayment, and the interest rate R per time period. The program should take these numbers A, N, and R and compute the payment size P. The output should be the payment size P.

Your program:

a. 100 "P"
110 _____
120 _____
130 _____
140 _____
150 _____
160 _____

---

a.
```
100 ''P''
110 INPUT ''INITIAL AMOUNT?'';A
120 INPUT ''INTEREST RATE?'';R
130 INPUT ''PERIODS TO REPAY?'',N
140 P = A*R/(1-(1+R)^(-N))
150 PRINT ''SIZE OF PAYMENT: '';P
160 END
```

The payment size equation can be rearranged when applying logarithms (a mathematical notation related to exponents) to get:

The Repayment Time Equation

$$N = LOG(P - A * R)/LOG(1 + R)$$

This equation gives the number N of time periods necessary to repay an amount A, when the interest rate is R per time period, and the payment is P per time period. The equation is written in logarithmic form to take advantage of the logarithm function of the Pocket Computer.

Repayment Time Problem

Write a program called "T" that prompts the user to input the initial amount A, the interest rate R, and the payment size P. The program should compute the number N of time periods to achieve repayment. The output should be the number N of time periods.

Your program:

a. 200 "T"
210 _____
220 _____
230 _____
240 _____
250 _____
260 _____

---

a.
```
200 ''T''
210 INPUT ''INITIAL AMOUNT?'',A
220 INPUT ''DECIMAL RATE?'',R
230 INPUT ''PAYMENT SIZE?'',P
240 N = LOG(P-A*R)/LOG(1+R)
250 PRINT ''PERIODS TO REPAY: '';N
260 END
```

Money Grows in More Than One Way

Some money grows in spurts and jumps. Some money grows steadily and continuously. The two ways of growing are related. Consider what happens when you put $100 in the local bank. This bank pays 15% interest at the end of each year. For each dollar invested, $1.15 is returned. At the end of one year, you have $100(1 + .15) = \$115$ in the bank. You made $15. Nice going. That's pretty good; but, down the street is a bank that pays half of the interest at the middle of the year. That is, it compounds your interest semi-annually.

Assume that you do move your money down the street. At the end of the year, you have $100(1 + .15/2) \times (1 + .15/2) = 100 \times (1 + .15/2) \wedge 2 = \$115.56$. That's better, but hold on. There is yet another bank further down the street that will pay you your interest every day (compounded daily). At this bank you make $100 \times (1 + .15/365) \wedge 365 = \$116.18$. That's a little better yet. But don't move your money yet.

All three of the banks discussed pay the interest in spurts and jumps. One bank calculated the interest annually, one calculated the interest semiannually, and the third calculated the interest daily.

There is one last bank that will pay you interest every moment. They will compound your investment continuously. This provides a steady and continuous growth.

Continuous Interest Problem

Write a program to compute how much every initial dollar will grow if the annual interest rate is 15%, but the compounding periods are made smaller and smaller.

Hint to Continuous Interest Problem

The quantity that needs to be computed is $(1 + .15/N) \wedge N$, where N is allowed to grow larger and larger. Does the quantity grow even larger? Does it approach some limiting number?

Your program:

a. 10 _____
20 _____
30 _____
40 _____
50 _____

---

a.
```
10 N = 0
20 N = N+1000
30 A = (1+.15/N)^N
40 PRINT N,A
50 GOTO 20
```

When we ran this program, we got these numbers.

N = 1000	A = 1.161821173
N = 10000	A = 1.161832936
N = 100000	A = 1.161834112

It looks like the numbers may be approaching a limit. Your TRS-80 Pocket Computer has a built-in function EXP(X). It is a remarkable and important fact that $(1 + .15/N) \wedge N$ gets closer and closer to EXP(.15) as N gets larger and larger. This same fact is true for interest rates other than 15%.

Continuous Interest Using EXP

Write a program to compare the values of:

$$(1 + R/10000) \wedge 10000 \text{ and EXPR}$$

for values of R = .1, .12, .14, .16, .18, and .2.

Your program:

a. 10 _____
20 _____
30 _____
40 _____

---

a.
```
10 FOR S = 10 TO 20 STEP 2
20 R = S/100
30 PRINT (1+R/10000)^10000, EXP(R)
40 NEXT S
```
← The Pocket Computer can only use integral steps in a FOR-NEXT loop

When you run the program, you will see that the two expressions give you very close results.

Catherine's Folly Problem

Catherine wants her racing yacht badly. She can't wait until she saves up enough money to pay cash. She has saved $10,000, but the ship costs $100,000. The yacht dealer is a kindly fellow and will let her have the boat with a down payment of $10,000 and a payment of $2,000 per month. The dealer will charge her 2% of the unpaid balance during each preceding month. How long will it take Catherine to pay off the boat, and how much will she pay to the dealer including interest?

Hint to Catherine's Folly Problem

This problem is similar to the time payment problem. The main difference is that we want to add a counter to count the months, and an accumulator to store all the payments. Write a program to do this.

Your program:

a. 10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____
100 _____

a. We have added a month counter (I) and an accumulator (Q) to the time payment program.

```
10 REM MONTH I, TOTAL PAID Q
20 I = 0: Q = 0

30 A = 90000 :REM AMOUNT OWED (S)
40 R = .02 :REM INTEREST RATE (R)
50 P = 2000 :REM PAYMENT (P)

60 PRINT I,Q

70 A = A+R*A-P :REM NEW AMOUNT

80 Q = Q+P :REM ADD PAYMENT TO TOTAL

90 I = I+1 :REM INCREMENT MONTH COUNTER

100 GOTO 60
```

Present Value of a Cash Flow

Would you rather have a bank account from which you can withdraw $100 each month or a business that pays you $100 per month? It really doesn't matter much, does it? From the accountant's point of view, one is just as good as the other. The fact that these situations have the same value is the basis for an ingenious method for comparing seemingly incomparable situations which is called present value analysis. Here is how it works.

Suppose that you are offered a bond that will pay you $1,000 in 10 years. What is a fair price for the bond? You certainly wouldn't pay $1,000. If you keep the $1,000, you can put it in the bank or some other investment and earn interest for 10 years. It would not benefit you to waste all that interest for 10 years. The fair price of the bond should take into account all the interest that you can earn during the length of the bond's life.

Suppose that the interest rate that you can earn by investing your $1,000 is 12% compounded yearly. What amount invested now at that rate would grow to be $1,000 after 10 years? Call this amount X for the moment. This amount X will be a fair price for the bond, since, by investing the amount X, you can withdraw $1,000 after 10 years.

Having the amount X right now is equivalent to getting $1,000 in 10 years. We know that:

$$X \times (1 + .12) \wedge 10 = 1000$$

A bit of algebra yields: $X = 1000/(1.12) \wedge 10 = 321.97$
Therefore, the present value of the bond is $321.97. This method works for more complex cash flows.

In a home, as well as in a business, you are interested in how much money is coming in and how much money is going out. Money may come in from more than one source, and it certainly flows out to many different destinations. The flow of money to and from a home or business is referred to as its cash flow.

How much should you pay now for the following package of two bonds? The first bond yields $2,000 in 5 years, and the second bond yields $3,000 in 10 years. Assume a 12% interest rate prevails.

The present amount that you need to deposit at 12% interest so that you can withdraw $2,000 in 5 years is given by the expression:

$$2000/1.12 \wedge 5 = 1134.85$$

The present amount that you need to deposit to get $3,000 in 10 years is given by:

$$3000/1.12 \wedge 10 = 965.92$$

The present value of the two bonds is the sum of these amounts, or $2,100.77. This is the amount you need to invest now at 12% to furnish the same cash flow as the two bonds. The present value of a cash or income flow (money coming in as opposed to money going out) depends heavily on the assumed interest rate.

Present Value of a Cash Flow Problem

Write a program that takes as input the expected interest rate and the amounts and time to maturity of any number of bonds. The program should print the present value of the bond package.

Hint to Present Value of a Cash Flow Problem

The central computation in this problem takes the future value F of the bond, the time T to maturity, and the expected interest rate R. It computes the present value P of the bond. In BASIC it is:

$$P = F/(1 + R) \wedge T$$

The present values P of the individual bonds in the package must be added to get the present value of the package. It will also be necessary to tell the program when we have finished entering all the input variables. Since bonds of future value 0 will not occur, you can use this as a signal that you are through entering data.

Your program:

a.
```
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____
90 _____
100 _____
```

- - - - - - - - - - - - - - - -

a.
```
10 REM P ACCUMULATES INDIVIDUAL PRESENT VALUES
20 P = 0

30 INPUT ''INTEREST RATE?'',R
40 INPUT ''FUTURE VALUE OF BOND?'',F

50 REM CHECK IF DONE
60 IF F = 0 THEN 100

70 INPUT ''TIME TO MATURITY?'',T

80 P = P+F/(1+R)^T

90 GOTO 40

100 PRINT ''THE PRESENT VALUE IS '',P
```

Stock Dividend Values

Here are two more deals for you. The first deal is:

You can buy stock in Acme Technology Company, a high growth computer manufacturer. There is good reason to believe that each dollar's worth of Acme stock will pay the following dividends at the end of each of the next 5 years.

Year	1	2	3	4	5
Dividend	.10	.20	.30	.40	.60

The second deal is:

You can buy stock in Bolt Bubble Gum Company. There is good reason to believe that each dollar's worth of stock will pay the following dividends at the end of each of the next 5 years.

Year	1	2	3	4	5
Dividend	.60	.40	.20	.10	.10

Which of these cash flows has the highest present value? Assume a 15% compound annual interest rate.

Hint to Big Deal Problem

What we've got here is a fixed interest rate of 15% per year, and various amounts arriving at various times in the future. This is exactly what the previous program deals with. Use the previous program for the present value of a cash flow to do this problem. Run each deal separately and compare the results:

Present Value for Acme _____
Present Value for Bolt _____
_____ has the greater present value.

Solution to Big Deal Problem

When you run the program for present value of a cash flow, it asks "INTEREST RATE?" You enter .15 as the interest rate. Next it asks "FUTURE VALUE OF BOND?". You enter .1, which is Acme's first yearly dividend. Next the computer asks "TIME TO MATURITY?". You enter 1, which is the time until you receive the dividend. Enter each of the other dividends and times in a similar manner. When all of Acme's dividends are entered, type 0 to get the present value of the future dividends. We got .96 for the present value of Acme's expected dividends. Run the program a second time to find the present value of Bolt's dividends. We got 1.06 as the present value of Bolt's expected future dividends. Bolt Bubble Gum has the greatest present value. Although it receives less total money, it receives it sooner. That's good.

```
Present Value for Acme is .96
Present Value for Bolt is 1.06
Bolt Bubble Gum Company has the greater present value.
```

Present Value with Changing Interest Rates Problem

Suppose some economic forecasters project a declining interest rate for the next 5 years. Here are their latest figures.

Year	1	2	3	4	5
Rate	15%	14%	12%	10%	8%

These are the rates that you can expect to earn during each of the next 5 years on your investments. You should take this into account when computing present value.

We need a program which computes the present value of an amount to be received in 5 years, but which takes into account the changing interest rates.

Hint to Present Value with Changing Interest Rates Problem

Compute an example by hand to see what is required in the computer program. Suppose that you will receive $1,000 in 5 years. What amount needs to be invested now, assuming the five previously projected interest rates over the next 5 years, in order to withdraw $1,000 at the end of that time. Call the present amount P. Then P will grow to P + R × P, or (1 + R) × P each year, where R is the rate for that year. Here is how the calculations look:

Year	Amount at Start of Year	Amount at End of Year
1	P	(1 + .15) * P
2	(1 + .15) * P	(1 + .15) * (1 + .14) * P
3	(1 + .15) * (1 + .14) * P	(1 + .15) * (1 + .14) * (1 + .12) * P

After 5 years P will have grown to:

$$(1 + .15) * (1 + .14) * (1 + .12) * (1 + 10) * (1 + .08) * P = 1000$$

You can see that:

$$P = 1000/((1 + .15) * (1 + .14) * (1 + .12) * (1 + .10) * (1 + .08))$$

The general equation for P is:

$$P = F/((1 + R) * (1 + S) * (1 + T) * (1 + U) * (1 + V))$$

where P is the present value
F is the future value
R, S, T, U, and V are the interest rates for each of the 5 years

Your program:

a.
10 _____
20 _____
30 _____
40 _____
50 _____
60 _____
70 _____
80 _____

---

a.

```
10 REM R,S,T,U,V ARE INTEREST RATES
20 R = .15: S=.14: T=.12: U=.10: V=.08

30 REM F IS THE FUTURE AMOUNT
40 INPUT F

50 REM COMPUTE PRESENT P FOR F
60 P= F/((1+R)*(1+S)*(1+T)*(1+U)*(1+V))

70 PRINT ''PRESENT VALUE = '';P

80 END
```

Summing Up Chapter Nine

The TRS-80 Pocket Computer performs complex business computations easily. In this chapter we saw how to do the following calculations.

- compute the future value of an investment earning compound interest with:
  A = A × R × A where A is the amount and R is the rate;
- compute the future value of an annuity with A = A + R * A + D, where A and R are the same as above and D is the deposit;
- compute time payment size with: P = A * R/(1 − (1 + R) ∧ (−N);
- compute repayment time with: N = LOG(P − A * R)/LOG(1 + R);
- compute continuous interest with: A = (1 + R/N) ∧ N or EXP(R);
- compute time payment schedules with: A = A + R * A − P, where A and R are as above and P is the payment made;
- compute the present value of a cash flow with P = F/(1+ R) ∧ T, where P is the present value, F is the future value, R is the rate, and T is the time to maturity;
- compute the present value when the rate is changing with:
  P = F/((1 + R1) * (1 + R2) * (1 + R3) * . . . . . *(1 + Rn))

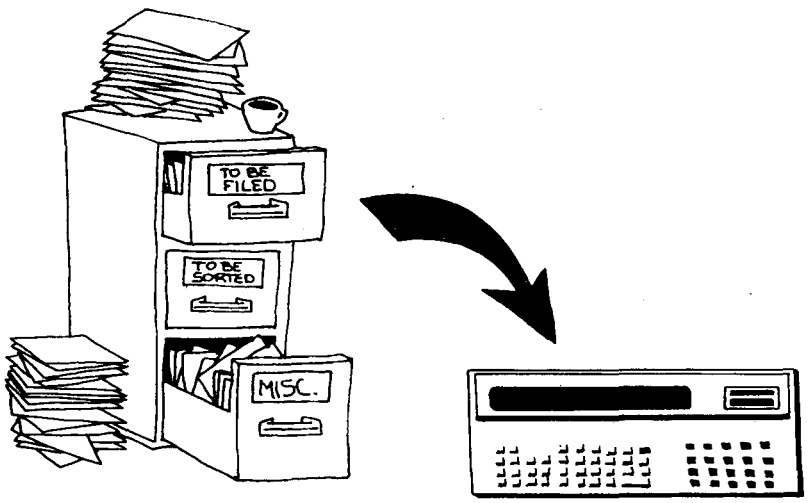
CHAPTER TEN

Storing, Sorting, and Searching (Or How to Make Sure You Know Where It's At)

The only way to keep your sanity in this increasingly complicated world is to get things into some kind of order. Rearrange the junk in the miscellaneous drawer. File all those clippings in the TO-BE-FILED folder. Add all the new names and telephone numbers in your address book (or in your Pocket Computer data file). Put all the bills into alphabetical order. Put the names of customers into alphabetical order. Well, maybe tomorrow, or the next day, when you have some spare time with nothing better to do.

Computers are renowned for their orderly and efficient ways. Can you use your friendly TRS-80 Pocket Computer to help get things in order at home and at work? Of course you can. In this chapter you will learn how to:

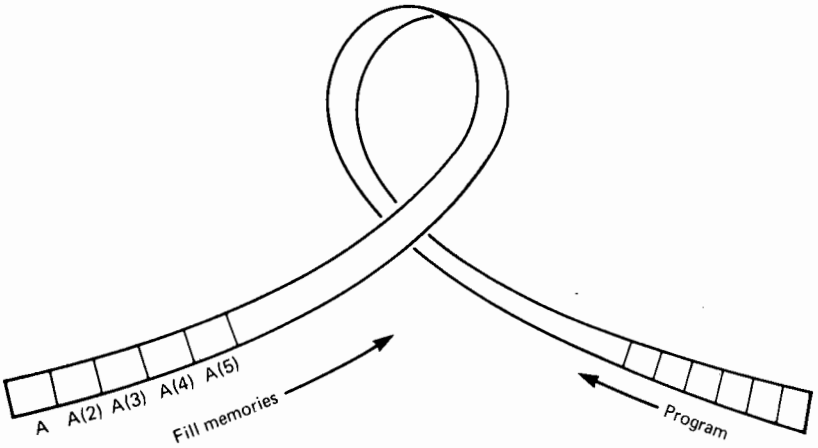
- build lists;
- add, delete, and move elements within those lists;
- put a list into proper numeric order; and
- search for elements within a list.



Full House Problem

Before we start storing, sorting, and searching lists of data, let's find out how much room we have in the Pocket Computer to store things. Write a quick program that stores the number 1 into every memory location. Start with memory A(2), then A(3), A(4), etc., until all the available memory locations are full. The program should print out the number of each memory location just before it is filled with the 1.

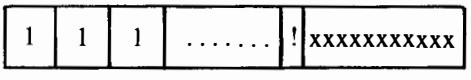
Remember, the program is filling memory locations from one end and the other is used to store data. If you write a short program, there will be more memories available for data.



- Which of the following will happen when all the available memories are filled with ones?
  - ☐ An error message will be displayed
  - ☐ 1's will be writen over the program memory
  - ☐ The RUN will stop automatically when the program memories are reached, but there will be no error message.
- How long will it take to fill all the available memories? _____ minutes.
- What will be the subscript of the last memory filled with a one? _____

Hint to Full House Problem

Be careful not to use any variables other than A. These memories will be filled with data when the program runs:



A(1) A(2) A(3) A(4) . . . . . Program

Use A=A(1) as the only variable in the program. Keep it short so that you can store lots of data.

Put your Full House Program here

a. 10 _____
20 _____
30 _____
40 _____
50 _____

---

a.
```
1Ø A=1
2Ø PAUSE A
3Ø A(A)=1
4Ø A=A+1
5Ø GOTO 2Ø
```

You could save a little space by writing this program on two lines.

```
1Ø A=1
2Ø PAUSE A: A(A)=1: A=A+1: GOTO 2Ø
```

- When all available memories were filled with ones:
  ☒ An error message was displayed

```
3Ø: 4. . . . . .
```

(ERROR CODE #4 - Insufficient memory)

- It took about 5 minutes to run the complete program.
- The last memory filled with a 1 was A(199).

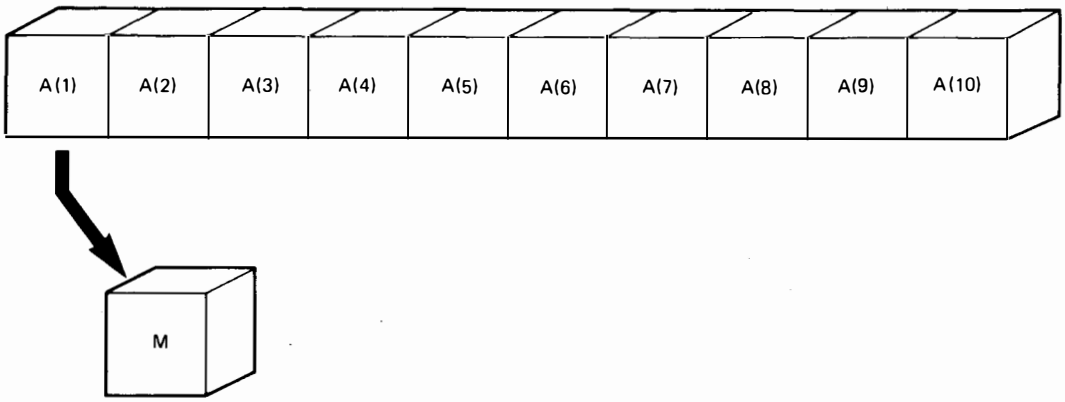
When you have a short program, you can use lots of memories to store data. If certain memory locations are filled with different numbers, can you write a program that will pick out the largest of the numbers?

Number Search Problem

Write a program that will let you input numbers into memories A(1) through A(10). The program must then search through the memories and find and print the largest of the ten numbers that you have input.

Hint To Number Search Problem

Let's take a look. Hmmm, A(1) is looked at first. At the beginning, it is the biggest number (since you haven't looked at any of the others). You should probably store A(1) somewhere. Let M = A(1). M will be the biggest number found at any stage of the program.



Next, check A(2). Is it bigger than M? If it is, then you should store the contents of A(2) in M. Otherwise, go right on to look at A(3).

At each step, compare the contents of a memory with M, which contains the biggest number found so far.

Solution to Number Search Problem

```
10 REM STORE NUMBERS IN MEMORY
20 FOR X = 1 TO 10
30   INPUT A(X)
40 NEXT X

50 REM A(1) IS INITIALLY THE BIGGEST
60 M=A(1)

70 REM COMPARE THE REST WITH M
80 FOR X=2 TO 10
90   IF A(X)>M LET M=A(X)
100 NEXT X

110 PRINT M
```

A different approach would be to start with M as a number so small that all 10 numbers stored in memory would be larger. Then the numbers would be compared with M, starting with A(1). For practical purposes, the number −999999 is probably small enough.

Second Solution to Number Search Problem

```
10 REM STORE NUMBERS IN MEMORY
20 FOR X=1 TO 10
30   INPUT A(X)
40 NEXT X

50 REM START WITH SMALL M
60 M=-999999

70 REM COMPARE TEN NUMBERS WITH M
80 FOR X=1 TO 10
90   IF   A(X)>M LET M=A(X)
100 NEXT X

110 PRINT M
```

The Pocket Computer finds the largest number very fast. Of course, it would take longer if there were more numbers. It would take it even longer if it had to rearrange the numbers in order from largest to smallest.

Out of Order Problem

Suppose that you have a list of numbers and would like to find out if they are in ascending order or not. You would need a program that checks whether each number examined is at least as big as the previous number. The program should allow you to input the numbers. It should then check whether the number input is at least as big as the previous number. If the input number is smaller than the previous one, the program should print "OUT OF ORDER" and stop.

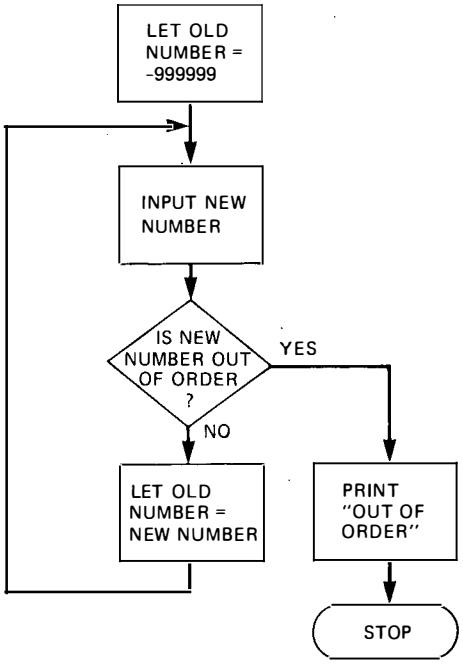
There is one minor constraint. Only three data memories can be used. You may choose any three memories but no more than three.

Hint to Out of Order Problem

It will be necessary to store the old number, the new number, and the message "OUT OF ORDER". After the new number has been checked, it then becomes the old number, and a newer number is input.

The first number input is always in order. What old number should you compare it with? How about a number less than any number that might be input? You can use −999999 again for this purpose.

This flow chart might help you with your program.



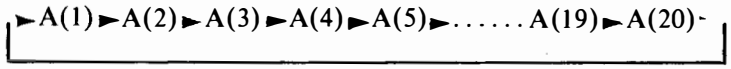
Solution to Out of Order Problem

We will store the old number in X, the new number in Y, and the message in Z$. We will start X off as the large negative number −999999. This ensures that the first number input will be greater than X.

The Program	From the Flowchart
`10 X=-999999`	Let old number = −999999
`20 INPUT Y`	Input new number
`30 IF Y<X THEN 60`	Is new number out of order? If yes, go to line 60
`40 X=Y`	Let old number = new number
`50 GOTO 20`	Go back for new number
`60 PRINT ''OUT OF ORDER''`	Print "out of order"
`70 END`	Stop

Cycle Up Problem

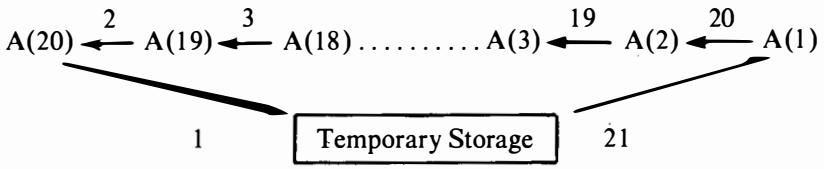
Have you ever noticed how often things get out of order? There are some days when the whole world seems off a notch. For times like that, you may need a little routine to put things right again. See if you can write a routine which shifts the contents of A(1) to A(2), the contents of A(2) to A(3), the contents of A(3) to A(4), and so on, until the contents of A(19) go into A(20) and the contents of A(20) go into A(1).



Around the World in 20 Cycles

Hint for Cycle Up Problem

If you can save a copy of A(20), then you can store A(19) in A(20). Once A(19) has been saved in A(20), you can store A(18) in A(19). Eventually, A(1) will be stored in A(2). This will then allow you to place the copy you made of A(20) into A(1). All the data have been moved without losing any.



It is done in reverse order to avoid losing data that have not yet been moved.

Solution to Cycle Up Problem

```
10 REM SAVE A COPY OF A(20)
20 Z=A(20)

30 REM Y WILL INDEX THE LOCATIONS
40 FOR Y=20 TO 2 STEP -1

50   REM SHIFT ALL CONTENTS UP
60   A(Y)=A(Y-1)
70 NEXT Y

80 REM NOW STORE THE CONTENTS OF A(20)
90 A(1)=Z
```

Of course you must have some data in memories A(1) through A(20) before you can move it. This can be done through another FOR-NEXT loop.

```
1 REM LOAD DATA
2 FOR Y=1 TO 20
3   INPUT A(Y)
4 NEXT Y
```

If you want to see the final results, you can print them out with a PAUSE statement in a FOR-NEXT loop.

```
100 FOR Y=1 TO 20
110   PAUSE ''A('';Y;'')='';A(Y)
120 NEXT Y
```

Reverse Play Problem

Thurston is a fine fellow and a good programmer. He has only one apparent fault. Thurston really never did learn to tell his left hand from his right hand. That causes some problems, as you can imagine. When Thurston writes programs, he tends to store the data in the wrong order. Since Thurston is such a great person, we will program around his backward data. We need a program that takes the numbers in locations A(1), A(2), A(3) . . . . , A(20) and puts the numbers in the opposite order. The contents of A(1) goes to A(20), the contents of A(2) goes to A(19), and so on, until the contents of A(20) goes to A(1).

Hint to Reverse Play Problem

The program will be similar to the Cycle Up Program. Watch out that you don't lose A(20) when you store A(1). Save a spare copy of A(20).

Solution to Reverse Play Problem

```
 10 REM LET Z INDEX THE LOCATIONS
 20 FOR Z=1 TO 10

 30 REM Y WILL RUN FROM 20 TO 11
 40 Y=21-Z

 50 REM STORE A COPY OF A(Y) IN X
 60 X=A(Y)

 70 REM TRANSFER A(Z) TO A(Y)
 80 A(Y)=A(Z)

 90 REM TRANSFER X TO A(Z)
100 A(Z)=X
110 NEXT Z
```

If you were to trace the action that takes place as the FOR-NEXT loop is executed, the results would be as shown in the following table. Of course, you will have to put some numbers in A(1) through A(20) before reversing the data.

Old Content	New Content	Z	Y	X
A(1) A(20)	A(20) A(1)	1	20	A(20)
A(2) A(19)	A(19) A(2)	2	19	A(19)
A(3) A(18)	A(18) A(3)	3	18	A(18)
A(4) A(17)	A(17) A(4)	4	17	A(17)
A(5) A(16)	A(16) A(5)	5	16	A(16)
A(6) A(15)	A(15) A(6)	6	15	A(15)
A(7) A(14)	A(14) A(7)	7	14	A(14)
A(8) A(13)	A(13) A(8)	8	13	A(13)
A(9) A(12)	A(12) A(9)	9	12	A(12)
A(10) A(11)	A(11) A(10)	10	11	A(11)

After ten times through the loop, the data have been completely reversed.

Sort of a Problem Problem

Sorting a set of numbers is one of the most tedious jobs a person can do. Try putting this set of numbers in order:

127, 33, 1, 201, 47, 127, 41, 6, 11, 117

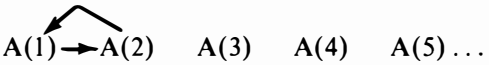
Even with only ten numbers it is a disagreeable job. One hundred numbers can be painful, and one thousand numbers can be a trauma. However, sorting is one of the most common and important jobs done by computers. Can you write a program that takes as input 20 numbers, then arranges them in order, and stores them in order in registers A(27) through A(46)?

Hint for Sort of a Problem Problem

Sorting by Insertion

Have you ever played bridge, or sorted papers alphabetically? What one usually does is to start at one end of the cards or papers, say the low end. The first card is left where it is. Now consider the second card. It is inserted before or after the first card depending on its value. Now consider the third card. It is inserted at the appropriate place among the first two cards. The process continues until all the cards have been inserted among the previously sorted cards.

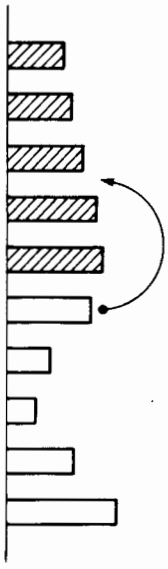
To make a computer version of the insertion sort, we must first decide where we will store the numbers to be sorted. Let's store them in memories A(1), A(2), . . . , A(20). Compare A(2) with A(1). If A(2) < A(1), then we wish to take A(2) out of its present position and put it in the first position.



To do this, we must move the contents of A(1) to A(2), and A(2) to A(1).

In the next step we will compare A(3) with A(1) and A(2). We want the contents of A(3) to replace the contents of the first location whose contents are greater than or equal to A(3).

Similarly, in each succeeding step we will compare an item with each previous item and insert it at the first location whose contents are greater than or equal to it.



Solution to Sort of a Problem Problem

Insertion Sort

The first thing that is needed is a routine which inputs and stores the items to be sorted. We used multiple statements on some lines to make the program shorter.

```
10 REM INPUT THE ITEMS TO BE SORTED
20 FOR Z=1 TO 20: INPUT A(Z): NEXT Z
```

The heart of the routine involves comparing each item with the set of previously sorted items and inserting it among them at the appropriate place.

Step 1

The item in A(1) may be considered to be a rather small set of sorted items.

Step 2

Compare the item in A(2) with the item in A(1). If A(1) <= A(2), then the items are in the proper order and we can deal with the item in A(3). But, if A(2) < A(1), then the item in A(2) must be inserted at A(1), and the item at A(1) must be moved up to A(2).

Step 3

Compare the item in A(3) with the items in A(1) and A(2). If it is smaller than one of these previous items, then it is inserted at that spot and those above that spot are moved up to fill the gap left by A(3).

Other steps

The program continues in this way until each item has been compared with all the previous positions and inserted into its proper place. Here is a crude first attempt at a program.

```
 10 REM INPUT THE ITEMS TO BE SORTED
 20 FOR Z=1 TO 20: INPUT A(Z): NEXT Z

 30 W=A(2): REM SAVE A COPY OF A(2)
 40 IF W<A(1) LET A(2)=A(1): A(1)=W

 50 REM NOW COMPARE AND INSERT A(3)
 60 W=A(3)
 70 IF W<A(1) LET A(3)=A(2): A(2)=A(1): A(1)=W
 80 IF W<A(2) LET A(3)=A(2): A(2)=W

 90 REM NOW COMPARE AND INSERT A(4)
100 W=A(4)
110 IF W<A(1) LET A(4)=A(3): A(3)=A(2):A(2)=A(1):
    A(1)=W
120 IF W<A(2) LET A(4)=A(3): A(3)=A(2):A(2)=W
130 IF W<A(3) LET A(4)=A(3): A(3)=W

140 REM NOW COMPARE AND INSERT A(5)
```

. . . and so on.

A program like this would work in principle, but not in practice. The final lines of this program would be many and long. This first attempt gives us something to work on and improve.

We notice that the program falls into blocks, each having very similar structure. Lines 90 to 130 form a typical block. A FOR-NEXT loop could be used for the LET statement and the statements to the right of the LET statement to simplify each line. We will put the block that moves the data away in a subroutine.

Here is how our new program looks.

```
 5 REM INSERTION SORT

10 REM INPUT 20 ITEMS TO BE SORTED
20 FOR Z=1 TO 20: INPUT A(Z): NEXT Z

25 REM NOW SORT THEM
30 FOR X=2 TO 20
40  W=A(X)

45   REM COMPARE W WITH A(Y) AND INSERT
50   FOR Y=1 TO X-1
60     IF W<A(Y) GOSUB 100
70   NEXT Y
80 NEXT X
90 END

95 REM MOVE SUBROUTINE
110 FOR Z=X TO Y+1 STEP-1:A(Z)=A(Z-1):NEXT Z
120 A(Y)=W:Y=X-1: RETURN
```

Let's make the program a little more general now. Instead of providing for 20 data items, it would be desirable to let you put in the number of data items that you desire.

```
 5 REM INPUT DATA
10 INPUT ''HOW MANY NUMBERS?'';N
20 FOR Z=1 TO N:INPUT A(Z):NEXT Z

25 REM NOW SORT THEM
30 FOR X=2 TO N
40   W=A(X): REM SAVE A COPY OF A(X)
50   Y=X-1: REM INDEX Y WILL COUNT DOWN
60   IF W>=A(Y) THEN 90: REM INSERT WHEN W
     CAN SINK NO FARTHER

65   REM IF W<A(Y) THEN CONTINUE
70   A(Y+1)=A(Y): REM SHIFT UP ONE
80   IF Y>0 THEN 60: REM REPEAT UNTIL Y=0
90   A(Y+1)=W: REM INSERT W
100 NEXT X
```

Hint for Sort of a Problem Problem

Exchange Sort

The choir director used this method. He wants to line up the choir boys in a row and then would move down the row exchanging adjacent boys so that the smaller stood on the left. After a few passes up the row, all the boys would be ordered from smallest to tallest.

To make a computer version, the numbers to be sorted must be stored, say in memory locations A(1), A(2) . . . A(20). Begin with the first pair. If A(1) < = A(2), then move on to the next pair. But, if A(1) > A(2), then exchange the contents of A(1) and A(2). Then move on and check the next pair. If A(2) < = A(3), then move on to the next pair. But, if A(2) > A(30), exchange their contents. Continue in this way until no further changes occur. The numbers are then in order.

Solution to Sort of a Problem Problem

Exchange Sort

Enter the numbers that you want to sort in memories A(1) through A(20). You want to examine each pair and exchange the pair, if necessary, so that the smaller is in the lower numbered location. Do this until no further exchanges occur. When this happens, the items are in proper order.

```
 5 REM INPUT 2Ø ITEMS
1Ø FOR Z=1 TO 2Ø: INPUT A(Z):NEXT Z
2Ø Y=Ø: REM Y WILL COUNT EXCHANGES

25 REM RUN THROUGH ALL PAIRS
3Ø FOR X=1 TO 19
4Ø   W=A(X): REM SAVE A COPY OF A(X)

45    REM IF A(X)<=A(X+1) GO TO NEXT PAIR
5Ø    IF W<=A(X+1) THEN 9Ø

55    REM IF A(X)>A(X+1) EXCHANGE
6Ø    A(X)=A(X+1)
7Ø    A(X+1)=W
8Ø    Y=Y+1: REM COUNT EXCHANGES

9Ø NEXT X

95 REM IF NO EXCHANGES, ITEMS ARE IN ORDER
1ØØ IF Y=Ø END

1Ø5 REM OTHERWISE RESET COUNTER AND DO AGAIN
11Ø Y=Ø
12Ø GOTO 3Ø
```

This program can be made more efficient. Consider what happens when the biggest item is encountered. It is exchanged with each succeeding item until it rises to the top of the list. On the next pass the next largest item rises as far as it can in the list. No later item will rise further than the position of the last exchange. The last exchange made on any pass marks the top of the unsorted items. Those items above this point are

in their final position and need not be considered in later passes through the list. You can use this observation by recording the last exchange position in T and ending the pass at that point. Here are the changes and additions to improve the previous program:

```
 22 T = 19
 3Ø FOR X=1 TO T
 82 S = X+1
115 T = S
```

Since the items rise like bubbles in the list, this type of sort is often called a "bubble sort."

Hint to Sort of a Problem Problem

Selection Sort

This method is the one that the coach used to find the biggest football team. He looked at all the players until he found the biggest. The biggest was called aside. Next, the coach looked at the remaining players until he found the biggest of the remaining players. That player was called to the side of the previously chosen player. This continued until all the players were in the new line, ordered from the smallest to the biggest. The coach could have called out the smallest players first just as easily.

Solution to Sort of a Problem Problem

Selection Sort

Once again, input 20 numbers and place them in memories A(1) through A(20). Then run through the list and find the smallest item. Store it in A(31). Next run through the list and find the next smallest. Store it in A(32). There are two difficulties to be considered. First, how does one find the smallest item in the list? Second, how does one avoid finding the first item again on the second run through the list?

To find the smallest item in a list you can hold a competition. The winner of any round competes at the next round. Store the current lowest number in W. To start, let A(1) be lowest. W = A(1). Next, compare W with A(2), A(3), and so on until we find a lower number, which is then stored in W. This new W is compared to the items remaining. At the end of the run through the list, W will contain the lowest number in the list.

Now you want to take this number out of the list so that it does not compete in the search for the second lowest number. The easiest thing to do is replace that item with a number so large that it is effectively infinite. Then it will not be considered before the other numbers in the list have been sorted out. Here is a routine which uses these observations.

```
 5 REM INPUT THE ITEMS
1Ø FOR X=1 TO 2Ø:INPUT A(X):NEXT X
2Ø Z=3Ø:REM Z COUNTS POSITION IN LIST
```

```
 3Ø FOR V=1 TO 2Ø:REM EACH PASS FINDS ONE ITEM
 4Ø   W=A(1): Y=1:REM A(1) STARTS AS SMALLEST
 5Ø   FOR X=2 TO 2Ø: REM SEARCH LIST
 6Ø     IF W<=A(X) THEN 9Ø: REM W STILL SMALLEST
 7Ø     W=A(X): REM A NEW SMALLEST
 8Ø     Y=X: REM STORE POSITION OF SMALLEST IN Y
 9Ø   NEXT X

 95   REM SET A(Y) TO INFINITY
1ØØ   A(Y)=999999

1Ø5   REM STORE NEW NUMBER W
11Ø   Z=Z+1
12Ø   A(Z)=W

13Ø NEXT V
```

Hint to Sort of a Problem Problem

Enumeration Sort

Here is a method you might use if the photographer tells your group to line up by height. Just count how many persons are shorter than you. They must all stand to your left. If five people are shorter than you, then you must be the sixth in line.

Solution to Sort of a Problem Problem

Enumeration Sort

For each item, you wish to count the number of items smaller than the given item. The final position of the item will be the number of smaller items, plus 1. Store the position in a separate list. Store the final position number of A(1) at A(31), the position number of A(2) at A(32), and so on. For example, if the final position of A(1) is fifth, then A(31) = 5. Here is a program based on this approach.

```
 5 REM INPUT 2Ø NUMBERS
1Ø FOR Z=1 TO 2Ø: INPUT A(Z): NEXT Z

15 REM A(31), . . . ,A(5Ø) ARE COUNTERS
2Ø FOR Z=31 TO 5Ø: A(Z)=1:NEXT Z

25 REM DETERMINE HOW MANY ITEMS ARE SMALLER THAN A(Z)
3Ø FOR Z=1 TO 2Ø

35   REM COMPARE A(Z) WITH EACH A(Y)
4Ø   FOR Y=1 TO 2Ø
5Ø     IF A(Y)<A(Z) LET A(Z+3Ø)=A)Z+3Ø)+1
6Ø   NEXT Y
7Ø NEXT Z

8Ø REM PRINT POSITION NUMBERS
9Ø FOR Z=31 TO 5Ø: PRINT A(Z):NEXT Z
```

Big Merger Problem

Lornalam and Smalley are cooperating on the checkbook this month. Lornalam took half the pile of checks and arranged them by the day of the year on which they were

cashed. Smalley did the same with his half. Now they need to merge the two ordered piles into a single ordered pile.

It would be interesting to make a computer version of this situation. Suppose that the dates of Lornalam's 10 checks have been ordered and stored in A(1) through A(10), and that Smalley's dates have been ordered and stored in A(11) through A(20). The memory locations each contain a number between 1 and 356 which represents the day of the year on which the check was cashed. Write a program which merges these two sequences and stores the result in A(31) through A(50).

Hint to Big Merger Problem

Check the first element in each list. Which is smaller, A(1) or A(11)? If, for example, A(1) is the smallest, then the contents of A(1) becomes the first element A(31) of the merged list. Next, consider A(2) and A(11). Which is the smaller? The smaller goes into A(32). If, for example, A(11) is the smallest, then you would next compare A(2) and A(12).

If one of the lists runs out, then all the elements in the remaining list go to the end of the merged list.

Solution to Big Merger Problem

The program has some small subtasks. First, the front items in the two lists are compared. Next, the smaller item is transferred to the merged list. Next, the index for the list from which the smallest item came is raised by 1 to point to the next item in that list. Also, the index in the merged list must be raised by 1 to point to the next vacant position. It is also necessary to check whether you are at the end of the list. If you are at the end of the list, then all the remaining items in the other list must be transferred to the end of the merged list.

```
 5 REM INPUT THE TWO LISTS
1Ø FOR Z=1 TO 1Ø:INPUT A(Z):NEXT Z
2Ø FOR Y=11 TO 2Ø INPUT A(Y):NEXT Y

25 REM SET LIST POINTERS
3Ø Z=1:Y=11:X=31
35 REM Z INDEXES THE FIRST LIST, Y THE SECOND LIST,
  X THE MERGED LIST

45 REM CHECK IF EITHER LIST IS DONE
5Ø IF  Z>1Ø THEN 115
6Ø IF Y>2Ø THEN 1Ø5

65 REM FIND AND STORE SMALLEST, THEN RESET POINTERS
7Ø IF A(Z)<A(Y) LET A(X)=A(Z): Z=Z+1:GO TO 9Ø
8Ø IF A(Y)<=A(Z) LET A(X)=A(Y):Y=Y+1
9Ø X=X+1
1ØØ GOTO 5Ø

1Ø5 REM ADD REMAINDER OF FIRST LIST TO MERGED LIST
11Ø FOR W=Z TO 1Ø:A(X)=A(W):X=X+1:NEXT W:END
115 REM ADD REMAINDER OF SECOND LIST TO MERGED LIST
12Ø FOR W=Y  TO 2Ø:A(X)=A(W):X=X+1:NEXT W:END

125 REM DISPLAY THE RESULTS
13Ø FOR X=31 TO 5Ø:PRINT A(X):NEXT X
```

We input the following lists:

A(1)	1
A(2)	20
A(3)	30
A(4)	40
A(5)	50
A(6)	60
A(7)	70
A(8)	80
A(9)	90
A(10)	100

A(11)	24
A(12)	27
A(13)	45
A(14)	51
A(15)	55
A(16)	65
A(17)	72
A(18)	95
A(19)	190
A(20)	300

After the run is complete, type RUN130 to see the merged list. We found:

A(31)	1
A(32)	20
A(33)	24
A(34)	27
A(35)	30
A(36)	40
A(37)	45
A(38)	50
A(39)	51
A(40)	55
A(41)	60
A(42)	65
A(43)	70
A(44)	72
A(45)	80
A(46)	90
A(47)	95
A(48)	100
A(49)	190
A(50)	300

A perfectly ordered, merged list

Summing Up Chapter Ten

You learned in this chapter that the TRS-80 Pocket Computer is adept at creating and manipulating data. It can search through lists of numbers, pick out the largest or smallest number, rearrange the numbers in ascending or descending order, reverse their order, or merge numbers from lists into a single ordered list. We haven't exhausted the Pocket Computer's manipulating capabilities, but we've given you an inkling of what can be done.

You, no doubt, will think of other uses of the Pocket Computer's manipulating abilities.

CHAPTER ELEVEN

Chaining Programs from Cassette

The Pocket Computer has enough internal memory to handle most of the tasks that you will find for it. But memory storage is like money; no amount ever seems like enough. You will certainly find, some sunny morning, that your program is too big and has just too much data to store in the internal memory. When that day comes, you will be delighted to find that the folks at Radio Shack have thoughtfully furnished the Pocket Computer with the ability to store information outside the machine.

In Chapters 3 and 4 you learned how to SAVE and LOAD programs and data on and from cassette tape. This chapter tells how to CHAIN (or connect) a program stored on cassette tape to a program currently in memory. However, there are several precautions and restrictions that must be observed.

You will learn:

- how to use the CHAIN statement;
- how the original section of a program is replaced by the chained section; and
- how variable values are passed from an original section of a long program to the chained section.

Chain Gang

A chain is useful for pulling up an anchor, or for holding up a swing, or for holding a large dog. Chains are certainly useful. Your TRS-80 Pocket Computer comes with a handy CHAIN statement. Like other chains it allows you to pull in a needed item.

In the computer case, the CHAIN statement pulls in programs from the cassette recorder and runs them. It allows you to break very large programs that won't entirely fit in internal memory into smaller ones, then call and run the pieces.

To keep things simple, we will not use large programs in our example. However, you can use your imagination as we demonstrate how the CHAIN statement works. First we need to have a program on the cassette. We will later use the CHAIN statement to call and run this program.

The Program to be CHAINed

```
10 PRINT ''CHAIN WORKS''
20 PRINT ''AND WORKS''
30 ''PUNT'':PRINT ''AND WORKS''
```

Load the program to be chained into the computer's memory. Save this microprogram on cassette under the name "MICRO", using the command:

```
CSAVE ''MICRO''
```

Clear the computer's internal memory by typing:

```
NEW
```

"MICRO" is now on the cassette but not in the computer.

Now use your imagination. Think of the next program as being a very long one that just fits in the computer's internal memory. It will use the CHAIN statement to call in the "MICRO" program that you just recorded and run it.

The BIG Program Using CHAIN

```
10 PRINT ''DOES CHAIN WORK?''
20 CHAIN ''MICRO''
```

Enter this program in the computer's memory. Since it will chain the program on cassette, you must prepare the recorder so that "MICRO" will be ready to run.

1. Rewind the recorder to the beginning of "MICRO".
2. Press the PLAY button on the recorder.

Now all is ready. Run the program in the computer. It will print the question:

```
DOES CHAIN WORK?
```

Press the ENTER key and the second program will be chained and run. You will see:

```
CHAIN WORKS
```

Press ENTER

```
AND WORKS
```

Press ENTER

```
AND WORKS
```

Notice line 30 of the program to be chained. We used a label "PUNT" that seems unnecessary. Sometimes you may not want to run the complete chained program. The CHAIN statement has some options that may be used. The statement:

```
CHAIN ''MICRO'',''PUNT''
```

will chain the complete program, "MICRO", but will execute it starting from the label, "PUNT". In this case, only line 30 would be executed and the display would show only:

```
AND WORKS
```

You can also use a line number instead of a label. The statement:

```
CHAIN ''MICRO'',20
```

would chain the program, "MICRO", and begin execution at line 20. In this case the display would show:

```
AND WORKS
```

from line 20

Press ENTER

```
AND WORKS
```

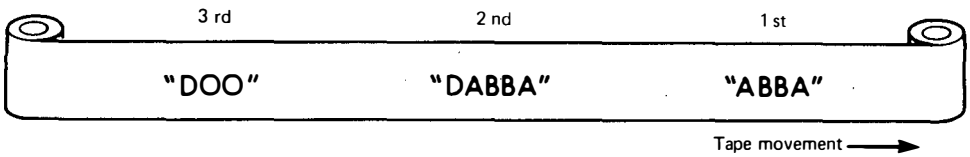
from line 30

Three Links Problem

Here are three short programs that have been saved on cassette with the names "ABBA", "DABBA", and "DOO".

```
10 ''ABBA''
20 PAUSE ''ABBA''
30 CHAIN ''DABBA''

10 ''DABBA''
20 PAUSE ''DABBA''
30 CHAIN ''DOO''

10 ''DOO''
20 PAUSE ''DOO''
30 CHAIN ''ABBA''
```

The programs are recorded on the cassette in this order:



The tape has been rewound to the beginning and is ready to play.

a. Describe what happens if you load and run the program "ABBA".

The tape is rewound and ready to play again.

b. Describe what happens if you load and run the program "DABBA".

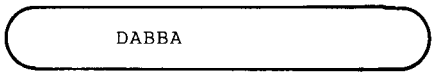
The tape is once again rewound and ready to play.

c. Describe what happens if you load the program "DOO".

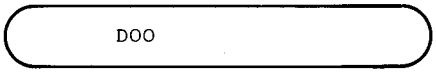
Solution to Three Links Problem

a. When you load and run the program "ABBA", it prints:

"DABBA" is then chained. It prints:

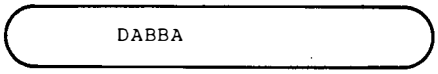


"DOO" is then chained. It prints:

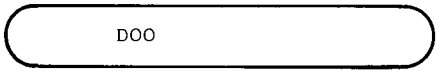


The cassette advances searching for "ABBA". Alas, "ABBA" will never be found since it is back in the other direction at the beginning of the tape.

b. When you run the program "DABBA", the cassette searches past the program "ABBA". It finds "DABBA" which runs and prints:

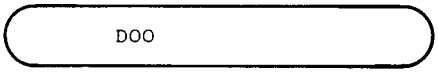


"DOO" is then chained. It prints:



The cassette then moves, searching for "ABBA". Of course, "ABBA" has already gone by and can't be found.

c. When you run the program "DOO", the cassette searches past "ABBA" and "DABBA" and finally finds "DOO". It prints:



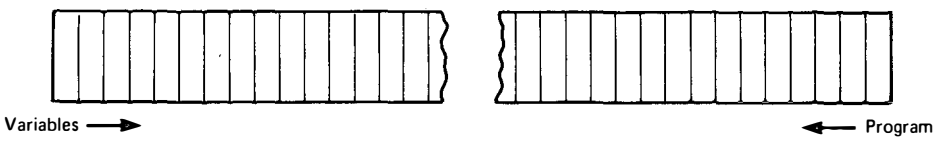
and then attempts to chain "ABBA". "ABBA" cannot be found since it is back in the other direction.

Precautions When Using CHAIN

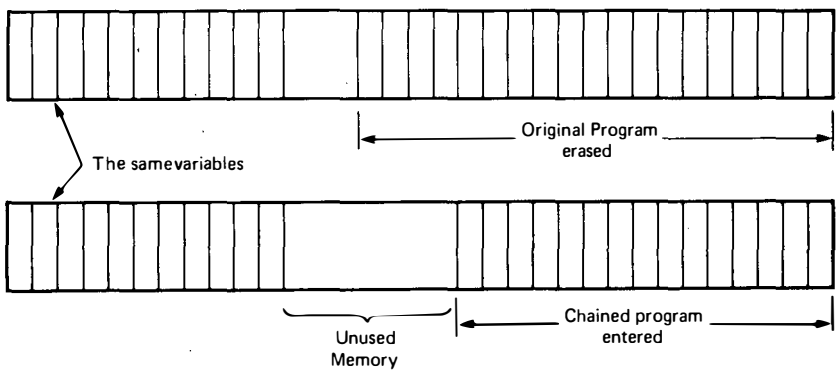
In Chapter 3 you found out that when a program was loaded from tape with the command CLOAD, any programs in the computer's internal memory were erased. The same thing happens when the CHAIN statement is used. The program, originally in memory, that uses the CHAIN statement is erased when the new program is read in from the cassette. You *cannot* return from the chained program to the original program. The purpose of the CHAIN command is to allow you to use programs too big to fit in the internal memory. When the second part is CHAINed, the first part must be erased to make room for the second part.

Using Variables in Chained Programs

Even though the original program is erased when a second program is chained in, the values for the variables used in the original program are still in memory. Remember how memory is used? The program is stored from one end while values for variables are stored from the other



As long as the chained program is not so big as to overlap the area used for the variables, the values for the variables will still be there.



The CHAIN statement is similar to the CLOAD statement, with these *important differences:*

1. The chained program is not only loaded but is *also executed.*
2. Any old program is erased, but the *values assigned by the old program remain in memory.*

Sample Use of CHAIN Statement

Here are two programs that demonstrate the use of the CHAIN statement with data transferred from the original program to the Chained Program.

Original Program

```
10 FOR X = 1 TO 5
20 A(X) = X+10
30 NEXT X
40 CHAIN ''NEXT''
```

Chained Program

```
10 ''NEXT''
20 FOR X = 1 TO 5
30 PRINT A(X)
40 NEXT X
```

Enter the Chained Program into the Pocket Computer's memory. Connect the cassette recorder interface and the recorder. Rewind the tape and press the RECORD and PLAY buttons on the recorder. Then type:

```
CSAVE ''NEXT''
```

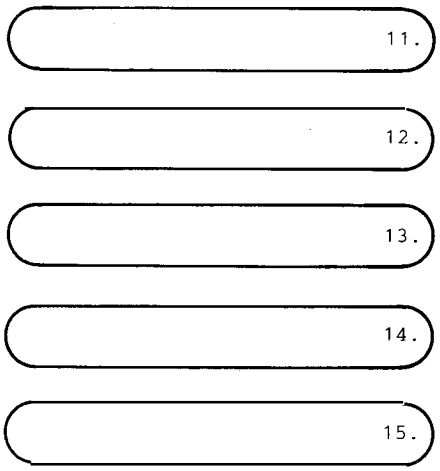
on the Pocket Computer and press the enter button.

After the Chained Program has been recorded, rewind the tape to the beginning of the tape. Now type:

```
NEW
```

and enter the original program in the Pocket Computer's memory. If you have rewound the tape (if not do so now), press the PLAY button so that the recorder is ready to read in the Chained Program. Set the Pocket Computer to the RUN mode and run the original program.

The original program is executed and the CHAIN statement at line 40 causes the Chained Program to be read in from the cassette recorder and executed. After this happens, the value for A(1) is seen on the display (caused by line 30 of the Chained Program). Press ENTER to see each of the five values A(1) through A(5).



A(1)

A(2)

A(3)

A(4)

A(5)

As you can see, the values for the variables were passed on from the original program to the Chained Program. To see that the original program has been erased, access the PRO mode and list the program in memory. The Chained Program will be there, but the original program will be gone.

Summing Up Chapter Eleven

When programs are so long that you must use the CHAIN statement, be sure to plan the division of the long program carefully. If very much data are to be passed between the sections of the program, be sure there is room to include them in the first section and room for them to be saved for the second section.

More than two sections may be chained, and the same precautions apply to each use of the CHAIN statement. With careful use, the CHAIN statement extends the power of the Pocket Computer far beyond the internal memory, making the execution of very long programs possible.

CHAPTER TWELVE

The TRS-80 Pocket Computer Printer

The Pocket Computer can pump numbers and letters out faster than you can write them down. In this chapter you'll learn how to use the TRS-80 printer to keep up with the flood of information and make a lasting impression of your work. You will learn;

- how to set up the printer;
- how to list your programs on the printer;
- how the printer places numbers and letters on the page;
- how to format numbers and letters on the page;
- how to use the printer to graph functions; and
- how to use the computer to store and print messages in the program area.

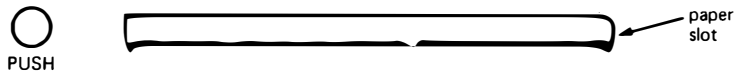
Getting Started

Unpack the printer. Is it blinking at you? The printer may have been packed with the switches on and the blinking light means that the battery is low. Turn the printer power switch off, and the blinking will stop. The printer has a nickel cadmium battery that can be recharged. (See Appendix E.) Plug the AC adaptor that was furnished with the printer into the receptacle on the back of the computer. Then plug the other end of the adaptor into a convenient wall outlet. The battery is now being fed. You may operate the printer while the battery adaptor (also used to charge the battery) is plugged into the wall socket.

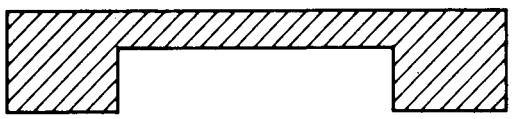
Next, turn the Power switch and the Print switch ON. Now press the button on the face of the printer that's marked □. That's the paper advance button. If you have paper in the printer, it comes out. If there is no paper in it, read on. If you already have installed the paper, skip over this section. You may want to come back to it when it's time to install a new roll of paper later on.

Installing Paper in Printer

Look closely at the left edge of the front of the printer. Next to the paper slot you will see a round dot and the word PUSH.



Give the dot a push and off comes the cover. Inside you can see the ribbon cartridge. It's shaped like this:



Sometime you may need to pop that out. Right now though, focus on the clear plastic cover at the top of the printer above the paper cover that you just removed. Lift the front edge of this cover and it will rotate on hinges toward the back of the printer. Inside the space that is revealed, the paper roll fits. Open one of those small paper rolls. The end of the roll may be a bit ragged by the time that you get the roll open. Cut the ragged end of the paper off with scissors. This will make the paper go in easily without any snags.

The paper enters the printer through the slot of the compartment used to hold the paper roll. Slide the paper into the slot and, at the same time, push the paper advance button □. Out comes the paper, toward the front of the printer. You should hold the button down until about one-half inch of paper feeds through. Next rotate the paper compartment cover back over the roll of paper. Slip the ribbon cover over the protruding one-half inch of paper. The paper goes through the slot with the jagged edge. Now snap the ribbon cover back into place and everything is ready.

Attaching the Computer

Directions for attaching the computer to the printer can be found on the printer itself.

1. Turn the printer power and the Pocket Computer off.
2. The bump at the upper right-hand corner of the printer interface fits neatly into a corresponding slot on the bottom of the computer. First, fit the bump into the slot; then slide the computer to the left, onto the connector prongs.

It's as simple as that. No need to force anything. Try putting the computer in and out a few times to be sure that you have the knack of it.

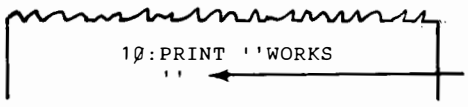
Now that the interface is connected, you can turn the printer Power switch ON. Also, turn on the Print switch. Last of all, turn on the Pocket Computer. In fact, push the computer ON button twice. The first push turns the computer on. The second push tells the computer that the printer is attached.

Listing a Program

Use the MODE key (of the computer) to put the computer into the PRO mode. Then type in this line.

```
1Ø PRINT ''WORKS''
```

Press the ENTER key and then type LIST. (Press the ENTER key again.) The printer leaps into action and lists the one-line program:

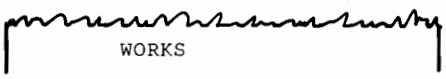


The ending quote appears on the second line.

In other words, when the computer is in the PRO mode, the LIST command will send the listing of the program to the printer.

Printing from a Program

Now press the MODE key several times to put the computer into the RUN mode. Type RUN and press the ENTER key. The printer will print:

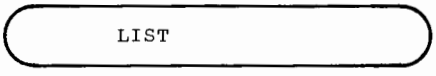


The program ends and the computer stops. Notice that the characters in the word WORKS were printed at the left side of the paper. Now let's go back and add another line to the program. Put the computer back into the PRO mode and add this line to the program.

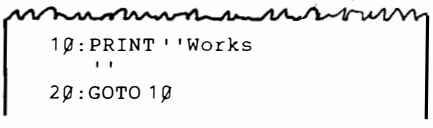
```
2Ø GOTO 1Ø
```

When you have entered line 20, type LIST to print the two-line program on the printer.

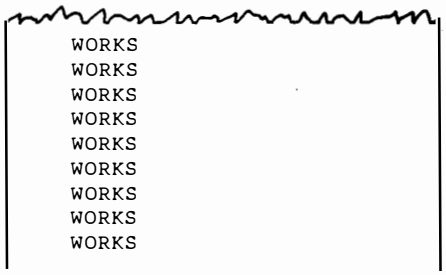
Display



Printer



Now run the program by putting the computer into the RUN mode and typing RUN.



When you've seen enough, press the computer's ON button to break the program.

The PRINT command works differently now that the printer is attached and turned on. The computer does not stop after each PRINT statement as it did without the printer. You don't need to press ENTER to tell the computer that you are ready to go on. The computer keeps on working when the printer is functioning in place of the display.

You saw how the printer printed the string "WORKS" on the left side of the paper. Let's try a string of numbers now. Put the computer in the PRO mode and change line 10 to:

```
10 PRINT''12345678901234567'' ←
```

It's still a string because of the quotes.

Use the MODE key to get into the RUN mode and run the program.

a. How many characters will the printer put on one line? _____

---

a. The printer prints:

```
1234567890123456
7
1234567890123456
7
1234567890123456
7
```

Press the ON button of the computer to break the program.

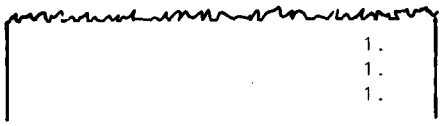
A printer line holds exactly 16 characters. The 17th character wraps around to the next line.

Now, let's try a shorter number that is not enclosed in quotes to see whether the printer treats strings and numerics alike. Change line 10 to:

```
10 PRINT 1
```

a. If you run the program now, on what part of a line will the printer print the number one? _____

The printer prints numbers on the right side of the page like this:

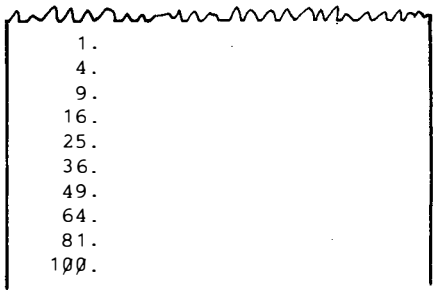


You can easily print tables for special functions using your printer. Here is a program that prints out the squares of numbers:

The Program as Listed on the Printer

```
10:FOR X=1TO 10
20:Y=X*X
30:PRINT Y
40:NEXT X
```

The RUN Printout



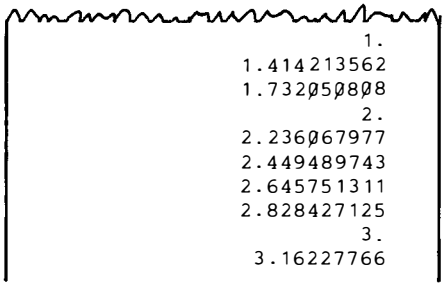
a. If you use the same program with the exception of line 10, which is changed to:

```
10 Y = √X
```

what will the printout of the run look like: ?

---

a. The numbers are printed at the right side of the page, but now there is an occasional decimal point. The printout looks like this:



Formatting Data for Printing

The numbers are a bit difficult to read in the previous jagged format. Let's see if we can tidy up the line of numbers by using the USING statement discussed in Chapters 2 and 7. Add:

```
5 USING ''##.####''
```

By listing the modified program we see:

```
 5:USING ''##.##
   ##''
1Ø:FOR X=1TO 1Ø
2Ø:Y=√X
3Ø:PRINT Y
4Ø:NEXT X
```

a. Now what happens when you RUN this program? Look at the printout of the previous program and show how line 5 changes the format of the printout.

---

a. The USING statement formats the number. The printout now looks like this:

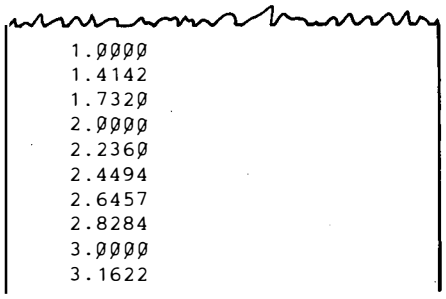
```
1.ØØØØ
1.4142
1.732Ø
2.ØØØØ
2.236Ø
2.4494
2.6457
2.8284
3.ØØØØ
3.1622
```

There is a way to place the numbers closer to the left side of the page. Change line 30 to:

```
3Ø PRINT '''';Y
```

no space between quotes

The printer puts strings at the left side of the paper. Even the string '' '' that has nothing in it starts on the left. The printer prints nothing, and then packs the number right up against it at the left edge. The printout looks like this:

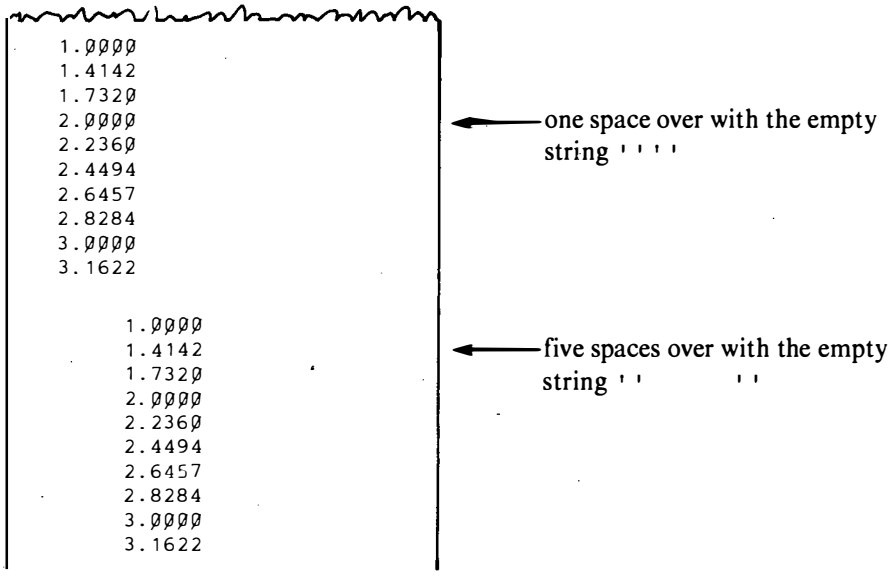


To space the numbers five positions to the right of the left side of the paper change line 30 to:

```
30 PRINT ''  '';
```

four spaces between quotes

Here is the difference in spacing:



You can also put two values on the same line. Change line 30 to:

```
30 PRINT X Y
```

Now X and Y will be printed on the same line when the program is run, like this:

Printer

```
 1.0000  1.0000
 2.0000  1.4142
 3.0000  1.7320
 4.0000  2.0000
 5.0000  2.2360
 6.0000  2.4494
 7.0000  2.6457
 8.0000  2.8284
 9.0000  3.0000
```

Display

```
30: 6 . . . . . .
```

Only nine pairs of values were printed. What happened to the tenth pair? Notice the computer display as the run stopped. It shows an error at line 30. It is error code number 6, showing an error in format. Although the `USING ''##.####''` statement on line 5 reserves two positions to the left of the decimal point, the left-most position is saved for the sign of the number (plus or minus). If the sign of a value is positive, it is not printed. However, the USING statement saves a place for it just the same. Therefore, when X = 10, there is no room for the sign and two digits. The computer stops and gives the error message. To correct this, change line 5 to:

```
5 USING ''###.####''
```

Now, RUN the program and all ten pairs of X,Y values will be printed.

With `USING ''##.####''`

```
 1.0000  1.0000
 2.0000  1.4142
 3.0000  1.7320
 4.0000  2.0000
 5.0000  2.2360
 6.0000  2.4494
 7.0000  2.6457
 8.0000  2.8284
 9.0000  3.0000
```

With `USING ''###.####''`

```
 1.0000  1.0000
 2.0000  1.4142
 3.0000  1.7320
 4.0000  2.0000
 5.0000  2.2360
 6.0000  2.4494
 7.0000  2.6457
 8.0000  2.8284
 9.0000  3.0000
10.0000  3.1622
```

If you remove the USING statement at line 5, you get a very hard-to-read output. The numbers are packed together like this:

```
 1.1.
 2.1.414213562
 3.1.732Ø5Ø8Ø8
 4.2.
 5.2.236Ø67977
 6.2.449489743
 7.2.645751311
 8.2.828427125
 9.3.
 1Ø.3.16227766
```

To get better spacing, you can put in an empty string. Change line 30 to:

```
3Ø PRINT X;''   '';Y
```

three blanks

a. Show how this change would effect the output.

---

a.

```
 1.    1.
 2.    1.414213562
 3.    1.732Ø5Ø8Ø8
 4.    2.
 5.    2.236Ø67977
 6.    2.449489743
 7.    2.645751311
 8.    2.828427125
 9.    3.
 1Ø.   3.16227766
```

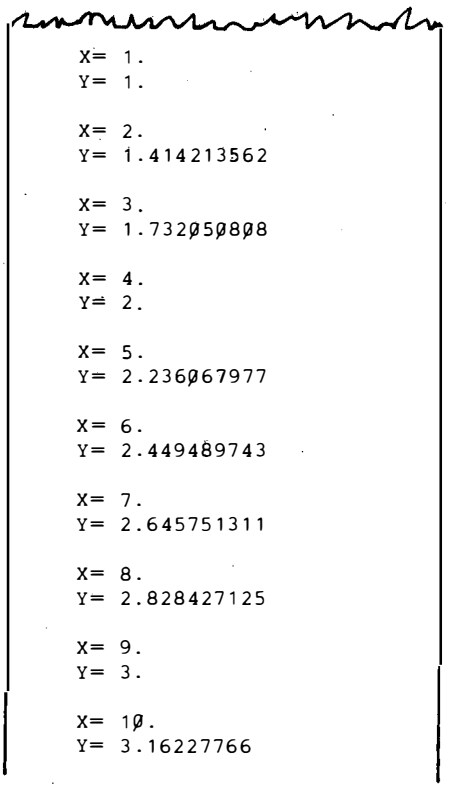
Here is another variation that provides a very neat and readable printed output.

change: `3Ø PRINT ''X= '';X`
add `32 PRINT ''Y= '';Y`
add: `34 PRINT '' ''` ← Prints a blank line

A listing of our program is now:

```
1Ø:FOR X=1TO 1Ø
2Ø:Y=√X
3Ø:PRINT ''X= '';
   X
32:PRINT ''Y= '';
   Y
34:PRINT '' ''
4Ø:NEXT X
```

The output looks like this:



```
X= 1.
Y= 1.

X= 2.
Y= 1.414213562

X= 3.
Y= 1.732050808

X= 4.
Y= 2.

X= 5.
Y= 2.236067977

X= 6.
Y= 2.449489743

X= 7.
Y= 2.645751311

X= 8.
Y= 2.828427125

X= 9.
Y= 3.

X= 10.
Y= 3.16227766
```

How to Store Messages

The Pocket Computer can be used to store long messages. You don't need to write an executable program to do it. It is REMarkably simple. Try this:

Put the computer in the PRO mode. Type NEW to clear the program memory. Now type this line and enter it.

```
1 ''SEND LISTS TO MEMORY
```

Now list the program. There are no surprises. You see the program line that you put in, with the addition of a colon following the line number.

```
1:''SEND LISTS TO MEMORY
```

The quote mark tells the computer not to interpret the sentence. It works just like a REM statement.

What is the maximum length for a message on a program line? Give this a try. Type:

```
1.''123456789Ø12345678.......
```

Keep going until the computer won't hold any more characters on the display.

a. How many characters can be stored on one program line? _____

a. You have probably noticed that the computer leaves a space at the left side of the page when you list a program. The computer saves the first three spaces for the line number. The colon comes next, then the program lines. One program line holds (or stores) 80 characters, if you count the line number, the colon, the quote mark, and all the characters typed in following the quote.

The length of a line displayed on the computer and the length of a printed line are quite different. Here are some examples of stored messages printed by the following method:

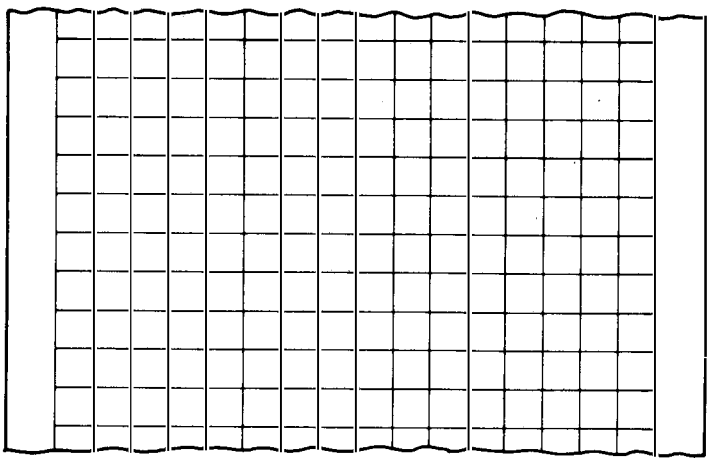
1. Type in the line numbers and messages.
2. When all are typed in, type the command LIST and press the ENTER key.

```
1:''12345678ØØ1
  234567ØØ123
  456789Ø12345
  6789Ø1234567
  89Ø123456789
  Ø12345678Ø01
  23456
2:''THE TRS-80
  POCKET COMPU
  TER IS DESIG
  NED TO HOLD
  MESSAGES IN
  THE PROGRAM
  MEMORY
3:''THE STANDAR
  D KEYBOARD M
  AKES TYPING
  FAST AND EAS
  Y.
4:''THE INSERT
  AND DELETE K
  EYS ALLOW YO
  U TO EDIT LI
  NES BEFOR ST
  ORING OR TYP
  ING.
5:''THE POCKET
  COMPUTER CAN
  EASILY STOR
  E A FULL, DO
  UBLE SPACED,
  PAGE.
```

The Pocket Computer can store long lines of text.

Graphing with the Printer

The TRS-80 printer can be used to do simple graphing. The graphic area is 16 units wide and as long as your paper tape.

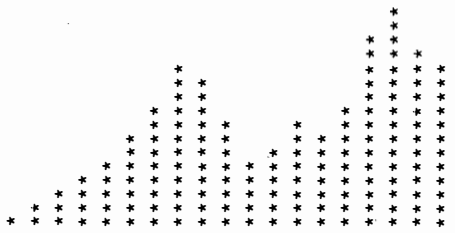


Here is a program that allows you to print a specified number of asterisks [*] on a line. The program takes the number of asterisks that you want printed as input and then prints them. We'll use this later in other graphing programs, so you may want to save the program with your cassette recorder.

```
10 INPUT Y:GOSUB ''GRAPH'':GOTO 10          <-- to input data

200 ''GRAPH''
210 REM CHECK SIZE OF INPUT
220 IF Y<0 LET Y=0
230 IF Y>16 LET Y=17
240 GOSUB 300+Y                             <-- pick the right number
250 RETURN                                      of asterisks

300 PRINT ''<'':RETURN
301 PRINT ''*'':RETURN
302 PRINT ''**'':RETURN
303 PRINT ''***'':RETURN
304 PRINT ''****'':RETURN
305 PRINT ''*****'':RETURN
306 PRINT ''******'':RETURN
307 PRINT ''*******'':RETURN
308 PRINT ''********'':RETURN
309 PRINT ''*********'':RETURN
310 PRINT ''**********'':RETURN
311 PRINT ''***********'':RETURN
312 PRINT ''************'':RETURN
313 PRINT ''*************'':RETURN
314 PRINT ''**************'':RETURN
315 PRINT ''***************'':RETURN
316 PRINT ''****************'':RETURN
317 PRINT ''. . . . . >'':RETURN
```

We used these inputs with the graph program to produce a graph of some stock market prices:

1, 2, 3, 4, 5, 7, 9, 12, 11, 8, 5, 6, 8, 7, 9, 14, 16, 13, 12



You probably have some numbers of your own that need graphing. Give them a try.

Graphs of Functions

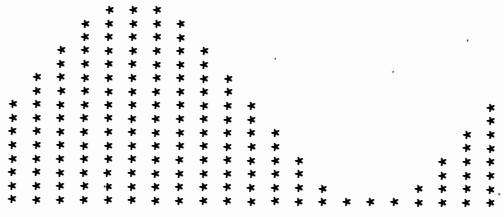
You can easily change the previous program to plot graphs of functions. Here is a program that uses the graph subroutine to graph the function Y = SIN(X).

```
10 DEG.
20 FOR X = 0 TO 360 STEP 18
30 Y = SIN(X)
40 Y = INT(7*Y+8.5)
50 GOSUB ''GRAPH''
60 NEXT X
70 END
200 ''GRAPH''
    .
    .
    .
```

This line scales the number to fit the printer.

Insert the previous "GRAPH" subroutine here.

The "GRAPH" subroutine (lines 200 through 317) is used just as it was before. Here's how the plot looks.

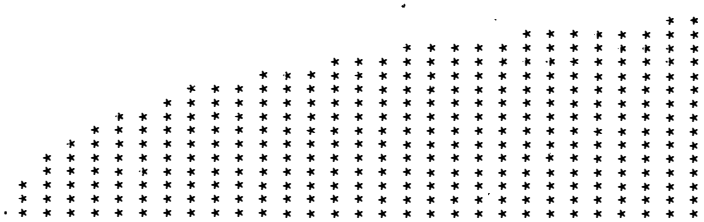


A graph of the function Y = LOG(X) can be made by making these changes to the graph of Y = SIN(X).

```
20 FOR X = 1 TO 30
30 Y = LOG(X)
40 Y = INT(10*Y+.5)
```

← This line scales the numbers to fit the printer paper.

Here is how the LOG plot looks:



Try some functions of your own. Here are a few to start with. Each function may need its own scaling values in line 40.

```
20 FOR X = -10 TO 10
30 Y = X*X
40 Y = INT(Y/100+.5)

20 FOR X = 0 TO 20
30 Y = EXP(-X)
40 Y = INT(Y*16+.5)

20 FOR X = -12 TO 12
30 Y = EXP(-X*X/4)
40 Y = INT(Y*16+.5)
```

Random Walk

By changing sections and adding sections to the previous program, a wide variety of simulations may be made. Let's examine a program that simulates a game where a gambler starts with five pennies. The gambler adds 1 to her total if the coin comes up heads, and subtracts 1 if the coin comes up tails. The coin is simulated by generating 1's and −1's randomly by the techniques of Chapter 8. The gambler's total on each play of the game is graphed on the printer. Make these changes and additions to the previous program.

```
1Ø PRINT ''INPUT A NUMBER''
2Ø INPUT R
3Ø Y = 5
4Ø GOSUB ''COIN''
5Ø Y = Y+C
6Ø IF Y<1 PRINT ''BUSTED'':END
7Ø IF Y>16 PRINT ''BANK BROKEN'':END
8Ø GOSUB ''GRAPH''
9Ø GOTO  4Ø

2ØØ ''GRAPH''
   .
   .
   .    (insert the subroutine "GRAPH" here)

5ØØ ''COIN''
51Ø REM GENERATE RANDOM 1, -1
52Ø R = 221*R+2113
53Ø R = R-INT(R/1ØØØØ)*1ØØØØ
54Ø C = SGN(R-5ØØØ)
55Ø RETURN
```

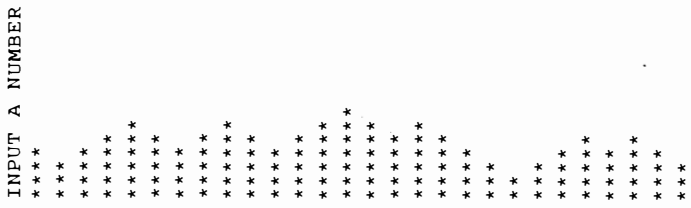
This starts the random number generator "COIN".

get a random C (1 or −1)

accumulate the total (Y)

add this section

If this program is RUN with the printer, the output will present a nice graph. Here is a sample printed output. Yours will look quite different, of course, since the numbers are chosen randomly by the program. We turned the paper 90 degrees to present the data in a more understandable format.



Because of the randomness of the numbers, it is difficult to predict the final result. Who knows if the gambler or the bank will go broke first?

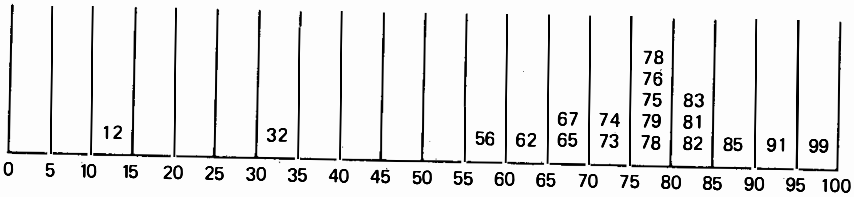
Frequency Distribution

One of the most useful tools used by true lovers of statistics is the frequency distribution, or histogram. Here is an example. Suppose you have a mountain of numbers. It might be the check amounts in your check book; it might be the running times for your favorite horses; it might be the grades in a class, or weights, or ages, or just about anything.

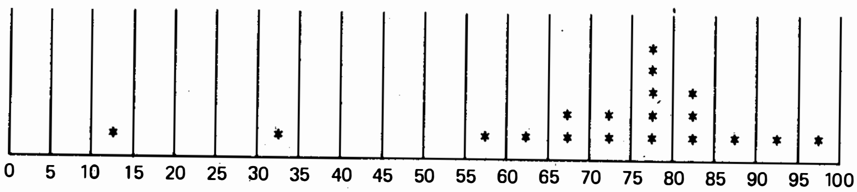
One of the most interesting and simple things that you can do is to sort the numbers by size. Let's sort some numbers. Here they are:

82, 73, 78, 65, 81, 79, 75, 62, 76, 85, 91, 78,
74, 67, 99, 56, 32, 83, 12

Sorted into columns, the numbers look like this:



If you replace the numbers with asterisks, the chart looks like this:



This chart is called a frequency distribution or histogram.

Histogram programs have many uses and are fairly easy to write. You can change the "GRAPH" subroutine program to get a histogram maker. The program takes numbers 0 through 100 as input. The number −1 (when input) signals the computer that you have typed in all your entries and are ready to graph. The computer then sorts the numbers by size, in intervals of length 5. Finally, it graphs the histogram on the printer.

```
10 CLEAR
20 ''HISTO''
30 INPUT Z
40 IF Z = -1 THEN ''PLOT''
50 IF Z < -1 PRINT ''WHAT?'':GOTO ''HISTO''
60 IF Z > 100 PRINT ''WHAT?'':GOTO ''HISTO''

70 REM FIND THE BOX (W)
80 W = 30+INT(Z/5)

90 REM ADD 1 TO THE BOX
100 A(W) = A(W)+1

110 REM GO GET ANOTHER Z
120 GOTO ''HISTO''

130 ''PLOT''
140 REM SEND NUMBERS TO PLOTTER
150 FOR Z = 30 TO 50
160 Y = A(Z)
170 GOSUB ''GRAPH''
180 NEXT Z
190 END

200 ''GRAPH''
  .
  .
```

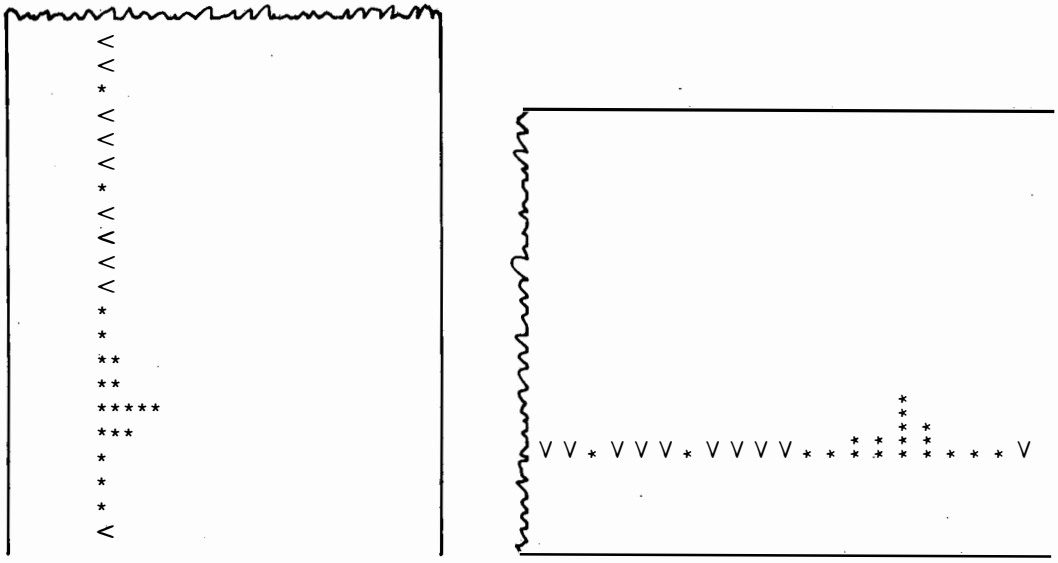
Once again, the "GRAPH" subroutine goes here.

You might find it instructive to use this program on the numbers in your check book. Here are the numbers that we put in, and the frequency distribution plotted by the program.

Inputs: 82, 73, 78, 65, 81, 79, 75,
62, 76, 85, 91, 78, 74, 67,
99, 56, 32, 83, 12

Frequency Distribution

Vertically Horizontally



Getting the Computer Display Back

You will occasionally want to display the results of your computations on the computer display rather than printing the results on the printer.

Put this short program in the computer to use as an example:

```
1 PRINT ''GOOD'':GOTO 1
```

When you run this program with the printer on, the word GOOD is repeatedly typed by the printer. To turn the printer off and get back to the display, do this:

1. Slide the PRINT switch to OFF.
2. Press the computer's ON button twice. This tells the computer that the printer's status has changed. Now the computer's output will go to the computer's display window.

a. RUN the program again and describe what happens.

_____

_____

---

a. The printer sits quietly. The word GOOD appears in the display window, and the computer stops.

When you are ready to print your results again, do this:

1. Slide the PRINT switch ON.
2. Press the computer's ON button twice.

a. RUN the program again and describe what happens.

_____

_____

---

a. The printer roars to life and repeatedly prints the word GOOD.

Using the Cassette Recorder

The TRS-80 Printer Interface can also transfer signals to the cassette recorder. The recorder is easy to hook up to the interface. The cable that comes with the printer has three plugs on each end. The directions are the same for each end of the cable (see Appendix E)

1. Black plug into the REM jack
2. Red plug into the MIC jack
3. Grey plug into the EAR jack

The easy way to remember this is, RED IS IN THE MIDDLE.

To use the cassette recorder with the interface:

1. Slide the Printer Power switch to ON.
2. Slide the Printer REMOTE switch to ON.

Now the interface is ready. Take a look at Chapters 3 and 4 for a detailed discussion of the use of the cassette recorder with the CSAVE, CLOAD, PRINT#, and INPUT# commands.

Summing Up Chapter Twelve

The printer is a powerful addition to the TRS-80 Pocket Computer. It is necessary for the computer to stop after each PRINT statement when no printer is attached so that you can see the results of a program. With the printer addition, your program can speed right along, printing each result as it goes. You can then keep a permanent record of results, data, and your programs. In this chapter you learned:

- how to set up the printer;
- how to LIST programs on the printer;
- how to use the PRINT command to print the output of programs on the printer;
- that the printer page holds 16 characters per line;
- that numbers print on the right side of the page and strings start on the left side;
- how to make numbers print where you want them to appear on the page;
- how to print numbers and characters together;
- how to store long messages in program memory;
- that the quote mark works like a REM statement;
- how to graph functions and make histograms;
- how to transfer from printer to display and back again; and
- how to hook up the cassette recorder.

Conclusion

The more you use your TRS-80 Pocket Computer, the more uses you will find for it. We hope this introduction will provide you with the background necessary for efficient use of the computer. If you have problems with it, or if you come up with some brilliant ideas for its use, write the authors at:

DYMAX
P.O. Box 310
Menlo Park, CA 94025

APPENDIX A

BASIC Statements and Commands

Note: Expressions enclosed in square brackets, [ ], are supplied by the user. [exp] stands for a mathematical expression, [char] stands for one or more alphanumeric characters, and [list] stands for a list of variables.

Statement	General Form	Function Performed
AREAD	AREAD [variable]	Assigns a value to a variable when used in the DEF mode. The value must be displayed before the execution of the defined program.
BEEP	BEEP ([exp])	Generates as many beeps as specified by the expression in parentheses.
CHAIN	CHAIN "[file name]"	Reads in and executes program recorded on tape.
CLEAR	CLEAR	Clears the data memories.
CLOAD	CLOAD "[file name]"	Loads a program from tape.
CLOAD?	CLOAD? "[file name]"	Compares the program in memory with the one just saved on tape.
CONT	CONT ENTER	Restarts an interrupted program at the place where the interruption occurred.
CSAVE	CSAVE "[file name]"	Records the program in memory onto tape.
DEBUG	DEBUG ENTER DEBUG [exp] ENTER DEBUG [char] ENTER	Starts instruction. Line numbers are listed as the program is executed.

Statement	General Form	Function Performed
DEGREE	DEGREE	Designates degrees as the unit for specifying angular measure.
END	END	Indicates the end of a program.
FOR	FOR [variable] = [exp] TO [exp] STEP [exp]	Begins a FOR-NEXT loop. The STEP is optional and defaults to one (1) if omitted.
GOSUB	GOSUB [exp] GOSUB [char]	Shifts execution to the specified line [exp] or label [char] where a subroutine begins.
GOTO	GOTO [exp] GOTO [char]	Shifts execution to the specified line [exp] or label [char].
GRAD	GRAD	Designates grads as the unit for specifying angular measure.
IF	(See *Radio Shack Software Manual* for numerous forms)	Based on the specified condition, a branch to a different part of a program is taken or not taken.
INPUT	INPUT [list]	Allows data to be input during program execution for one or more variables.
INPUT#	INPUT# "[filename]"	Records on tape data from data memories.
LET	LET [variable] = [exp] LET [variable]= [char]	Assigns a value to a variable.
LIST	LIST LIST [exp] LIST [char]	Lists the program currently in memory.
MEM	MEM	Displays the number of program steps and flexible memories that have not been used.
NEW	NEW	Clears all program and data memories.
NEXT	NEXT [variable]	Ends a FOR-NEXT loop. Increments the step.

Statement	General Form	Function Performed
PAUSE	PAUSE [exp] PAUSE "[char]" (For other forms see the *Radio Shack Software Manual.*)	Displays an output for approximately 0.85 seconds and then continues execution of the program.
PRINT	PRINT [exp] PRINT "[char]" (For other forms see the *Radio Shack Software Manual.*)	Displays an output and halts execution of the program until the ENTER key is pressed.
PRINT#	PRINT# "[filename]"	Records data memories on tape.
RADIAN	RADIAN	Designates radians as the unit for specifying angular measure.
REM	REM [char]	Designates a nonexecutable statement used to make remarks in a program.
RETURN	RETURN	Returns from execution of a subroutine.
RUN	RUN RUN [exp] RUN [char]	Starts the execution of a program.
STEP	(See FOR)	Increments a FOR-NEXT loop.
STOP	STOP	Stops the execution of a program. The execution may be resumed by a CONT statement.
THEN	(Used in an IF statement - see *Radio Shack Software Manual.*)	Shifts execution to a new line as GOTO. Used only in an IF statement.
USING	(Many forms - see *Radio Shack Software Manual.*)	Designates the format of the display of a PRINT or PAUSE statement.

APPENDIX B

Special BASIC Functions

BASIC Designation	Common Mathematical Designation or Meaning
ABS	absolute value
ACS	arc cos or $\cos^{-1}$ or inverse cosine
ASN	arc sin or $\sin^{-1}$ or inverse sine
ATN	arc tan or $\tan^{-1}$ or inverse tangent
COS	cos or cosine
DEG	degree/minute/second conversion to decimal
DMS	decimal conversion to degree/minute/second
EXP	e^x or exponential function or antilogarithm for LN
INT	integer
LN	$\log_e X$ or Natural logarithm
LOG	$\log_{10} X$ or Common logarithm
SGN	sign (positive or negative) or signum
SIN	sin or sine
TAN	tan or tangent
$\sqrt{}$	square root extraction

APPENDIX C

Acceptable Abbreviations for BASIC Statements, Commands, and Special Functions

Statement, Command, or Function	Acceptable Abbreviations				
ABS		AB.			
ACS		AC.			
AREAD	A.	AR.	ARE.	AREA.	
ASN		AS.			
ATN		AT.			
BEEP	B.	BE.	BEE.		
CHAIN		CH.	CHA.	CHAI.	
CLEAR		CL.	CLE.	CLEA.	
CLOAD			CLO.	CLOA.	
CLOAD?			CLO.?	CLOA.?	
CONT	C.		CO.	CON.	
COS					
CSAVE		CS.	CSA.	CSAV.	
DEBUG	D.	DE.	DEB.	DEBU.	
DEG					
DEGREE			DEG.	DEGR.	DEGRE.
DMS		DM.			
END	E.	EN.			
EXP		EX.			
FOR	F.	FO.			
GOSUB			GOS.	GOSU.	
GOTO	G.	GO.	GOT.		

GRAD		GR.	GRA.		
IF					
INPUT	I.	IN.	INP.	INPU.	
INPUT#	I.#	IN.#	INP.#	INPU.#	
INT					
LET		LE.			
LIST	L.	LI.	LIS.		
LN					
LOG		LO.			
MEM	M.	ME.			
NEW					
NEXT	N.	NE.	NEX.		
PAUSE		PA.	PAU.	PAUS.	
PRINT	P.	PR.	PRI.	PRIN.	
PRINT#	P.#	PR.#	PRI.#	PRIN.#	
RADIAN		RA.	RAD.	RADI.	RADIA.
REM					
RETURN		RE.	RET.	RETU.	RETUR.
RUN	R.	RU.			
SGN		SG.			
SIN		SI.			
STEP			STE.		
STOP	S.	ST.	STO.		
TAN		TA.			
THEN	T.	TH.	THE.		
USING	U.	US.	USI.	USIN.	

APPENDIX D

Error Codes

Error Code Number	Remarks
1	Caused by: Grammatical (syntax) error – Check formats used in offending lines. Operational error – Absolute value of some number may be greater than 10^{100} or a division by zero may have been attempted. Error in memory specification – May be a mismatch of string and numerical variables and values assigned.
2	Caused by: Line error – Check lines and labels specified by GOTO, GOSUB, RUN, DEBUG, or LIST statements. (The lines may not exist.)
3	Caused by: Level error – Occurs when level exceeds four stages in a GOSUB or a FOR-NEXT; when you try to execute a RETURN without a previous GOSUB; or when you try to execute a NEXT without a previous FOR.

4	Caused by: Insufficient memory – More program steps may have been used than available program memories; More reserve program steps may have been used than available reserve memories; More dimension memories may have been specified for the remaining memory.
5	Caused by: Control error of cassette tape – Occurs during the execution of tape control statements or commands.
6	Caused by: Error in format – Occurs when a display of numerical data are not in the specified format for use in a PRINT or PAUSE statement.

APPENDIX E

Printer Terms

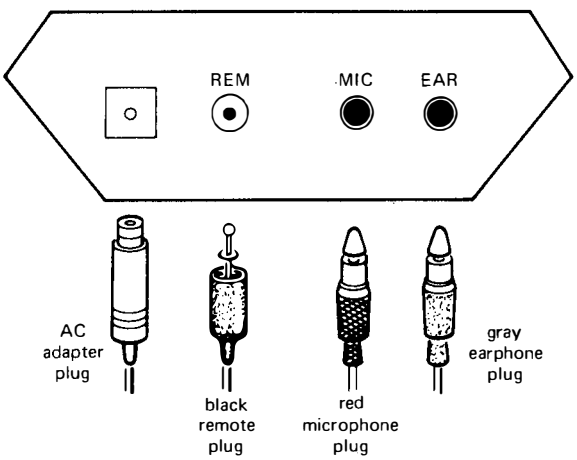
Power – The printer is powered in one of two ways. A rechargeable Ni-Cad battery allows you to use the printer/cassette interface in places that lack normal AC power. An AC adapter is provided and may be used to power the interface where a wall outlet is available.

Recharging the Battery – To recharge the Ni-Cad battery, turn the printer power switch OFF. Then connect the AC adapter to the interface and plug the other end of the AC adapter into a wall outlet. It takes about 15 hours to fully recharge the battery.

Connector Jacks

Adapter Jack – This jack is used to recharge the batteries or to operate the printer/cassette interface from a wall outlet. (See the diagram below.)

Cassette Jacks – The cassette connection cable plugs into the printer/cassette interface as shown in the diagram below. The other ends of the cable are plugged into a cassette recorder.



Controls and Indicators

All controls and indicators for the TRS-80 Pocket Computer printer/cassette interface are located in the lower left corner on the top of the interface and are clearly labeled.

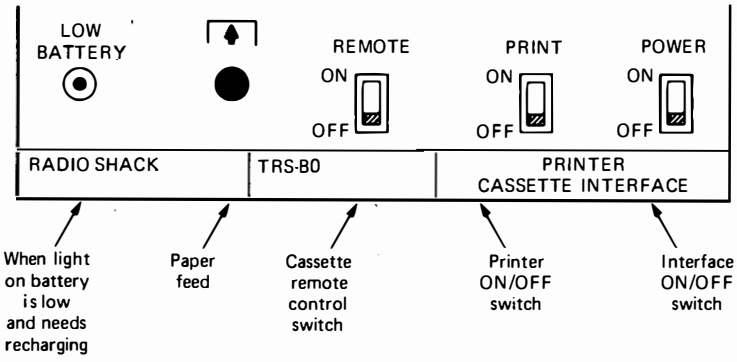
Power Switch – This switch applies power to the interface and must be set in the ON position when you wish to use the printer or the cassette recorder.

Print Switch – To use the printer, both the Power Switch and the Print Switch must be in the ON positions. In addition, the computer must be in the print mode before the computer's PRINT and LIST commands will activate the printer. If the Print or Power switch is OFF, the computer sends the results of PRINT and LIST commands to the LCD display instead of the printer.

Remote Switch – This switch controls the cassette recorder when it is connected to the interface. When the Remote switch is ON, motion of the recorder is controlled by the computer. When it is OFF, the recorder is under the control of the recorder's buttons.

Paper Feed Button – When the Paper Feed Button is pressed once, the printer's paper advances one line. When the button is held down, the paper advances continuously. The Power switch must be on for the Paper Feed function to perform.

Low Battery Indicator – This indicator will blink when the battery becomes too weak to operate the printer. At this time, input and output to the cassette recorder and output to the printer are disabled.



Print Mode – This mode sends PRINT and LIST results to the printer rather than the display. To enter the print mode, first turn ON the Print and Power switches. Then press the computer's ON key twice.

Display Mode – This mode is used to send PRINT and LIST results to the LCD display. To return to the display mode after using the print mode, turn the Print switch OFF and press the computer's ON key twice.

Computer Commands Used With the Printer

LIST – This command prints a listing of the current program in memory. The printer must be in the print mode, and the computer in the PRO operating mode.

PRINT – This statement is used inside a program line to direct program data to the printer rather than the LCD display. A PRINT statement followed by a single space within quotation marks produces a paper advance of one line.

Index

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

COMPUTERS $8.95

PROBLEM-SOLVING ON THE TRS-80™ POCKET COMPUTER

The ultimate in "micro" computing, the new TRS-80™ and SHARP Pocket Computers are small enough to tuck into your pocket—yet give more sophisticated computing than the most advanced programmable calculators.

Now, in the first book ever to teach problem-solving techniques using the Pocket Computer, two experts show you how to solve virtually any problem with this extraordinarily versatile computing tool, giving you mastery of a wide range of educational and practical applications. The book's self-paced format lets you learn at your own speed—all you need is a familiarity with the BASIC language and a TRS-80™ (or SHARP) Computer in your pocket.

First, problem-posing and problem-solving exercises teach you the "nuts and bolts" of working with the Pocket Computer—the machine's own form of BASIC, its keyboard, and its special features. You'll then learn how to use the Pocket Computer as a practical tool in many areas ranging from logic functions to storing, sorting, and searching to randomization and more.

Scores of lively, interesting problems are presented with graded hints that encourage you to develop your own solutions. Like all Wiley Self-Teaching Guides, this volume has been carefully designed to facilitate your learning and includes a wealth of illustrations and diagrams.

Don Inman is an executive with Dymax Corporation in Menlo Park, California. He is co-author of the best-selling guides TRS-80™ BASIC and MORE TRS-80™ BASIC.

Jim Conlan is a Professor of Mathematics at Menlo College in Menlo Park.

STG

More than a million people have learned to program, use, and enjoy microcomputers with Wiley STGs. Look for them all at your favorite bookshop or computer store!

JOHN WILEY & SONS, INC.
605 Third Avenue, New York, N.Y. 10158
New York • Chichester • Brisbane • Toronto

ISBN 0 471 09270-3