FUN AND GAMES WITH THE COMPUTER

Edwin R. Sage

ENTELEK

FUN AND GAMES WITH THE COMPUTER

Edwin R. Sage
Middlesex School
Concord, Massachusetts

ENTELEK
42 Pleasant Street / Newburyport, Massachusetts 01950

ISBN Number 87567-075-X
Printed in the United States

ACKNOWLEDGEMENTS

My sincere thanks go to Albert Hickey of ENTELEK for editing the text; to Jean Stritter and John Davidson of Middlesex School for teaching sections of the original manuscript and making creative suggestions for its improvement; to William Amory of MITRE Corporation for reading the manuscript; and to my wife, Kris, for her many hours of work with me from start to finish. Finally, thanks go to those past and present Middlesex students whose enthusiasm for computer games inspired this book.

Additional **ENTELEK** publications;

Problem Solving with the Computer
by Edwin R. Sage

ENTELEK Computer Based Math Lab
by Norman S. Katz

Programmed Instruction Guide, 3rd edition

Information may be obtained from:
ENTELEK
Newburyport, Massachusetts 01950

Games, I have found, are an excellent medium for teaching computer programming. First, they are fun to play. Second, it is easier to program a game than it is to program the solution to a math problem. That's because you generally know the rules of the game better than you know the rules for solving the math problem. In fact, you do not need any mathematical background at all to learn computer programming by this method.

This book teaches BASIC, a language commonly used to communicate with computers. BASIC (an acronym for Beginner's All-Purpose Symbolic Instruction Code) is easy to learn, easy to use, and available on many computers. In Chapter 1 you learn some elementary words in the BASIC language, two programming techniques called looping and branching, and some ways to fix your program when it doesn't work.

In Chapter 2 you program your first game, one in which the computer uses its random number generator to "think" of a mystery number which *you* try to guess. You then go on to add refinements to your program, including a variety of ways to repeat the game, a method for counting the number of tries you need to hit upon the mystery number, and a technique by which the computer gives you helpful hints, such as "too high" and "too low." In the end, you discover a strategy which reduces by a third the average number of guesses you need to hit upon the mystery number.

In Chapter 3 we turn the tables. You pick the mystery number and the computer tries to guess what it is. It quickly becomes apparent that the computer cannot remember what numbers it has already tried without success. To make the computer more efficient, you teach it how to use a "check list." This is done first by using many IF-THEN statements to prevent the computer from guessing the same number twice. A second method uses a subscripted variable, which enables us to reduce the number of steps in the program from 59 to 14.

You then go on to add to your new program the same refinements you added to the program you developed in Chapter 2. After making these additions, you further sharpen the computer's ability to guess your mystery number by teaching it each of the two strategies you discovered in Chapter 2. The first is called the Random Method and the second the Midpoint Method. As the final step in the chapter, you compare the efficiency of the two strategies. To do this, you first play five games in which the computer uses the Random Method and five games in which it uses the Midpoint Method. Five games are not enough to obtain truly reliable results, however. It would be better to play a

thousand games each way. But that would take a human player about 70 hours, so you teach the computer to play your role as well as its own, and thereby cut the playing time to about ten minutes.

Chapter 4 presents a change of pace. Instead of teaching the computer to play games, you teach it to draw pictures. More important, you learn some additional words in BASIC: TAB, READ, DATA, GO SUB and RETURN. You are also introduced to the concept of a subroutine and get a lot more experience building loops using the FOR-NEXT command.

In Chapter 5, you program the computer to play Big Wheel, a game in which you bet on the number at which a spinning carnival wheel will come to rest. The wheel you use will be of your own design. You divide it into several pie-shaped pieces, all of equal area, use the random number generator to pick the number at which the wheel is to stop, and construct the pay-off table.

Once you get the wheel to spin, you add a variety of features to your program. One feature, for example, prevents a player from betting more money than he has in his pocket. Finally, you learn how to document a program, including the use of the BASIC word REM for "remark" to identify the location of each feature in the program.

In Chapter 6, you again use the random number generator, this time to "toss" coins and "roll" dice. In the process you learn how to determine the various outcomes and the probabilities of their occurrence. These fundamental operations are then incorporated in two game programs, one called Two Coin and the other Craps.

Chapter 7 is about card games. After teaching the computer to "deal" from a deck of cards, you teach the computer to play three card games, the most popular of which is Blackjack or Twenty-one.

This is a book for the novice. To insure that you do not close the book with the false impression that the computer is limited to games of chance and strategy, we conclude with a guide to further exploration of computer applications in mathematics, science, business and education.

Fun and Games with the Computer is designed to be used in a one-semester course. As you read the first few chapters you'll discover that you can grasp the essentials of computer programming even if you do not have immediate access to a computer. Sooner or later, however, you'll probably want to try your skill on a real computer. Most of the games in the text can be run on a minimum 4K computer and the remainder on a somewhat larger 8K system. You will find it easier to do the work outlined in this book if your computer system provides on-line program storage on disk or magnetic

tape, avoiding the requirement to dump and reload your programs from paper tape. In any case, you'll also need a manual which describes the operation of your particular computer and the dialect of BASIC your system understands.

This book can also be used as a supplementary text or workbook to provide dynamic on-line experience with a computer as part of a course in computer literacy, a course which describes hardware, software, and applications.

The book is also a useful adjunct to a conventional course in probability and statistics, giving the student direct access to probabalistic events employing large numbers.

Despite the foregoing references to mathematics, statistics and computer literacy, the book does not demand a background in mathematics. It may seem like a bold statement, but any teacher who has a curiosity about computers and adequately prepares himself should be able to teach the course successfully.

Edwin R. Sage
March 15, 1975

TABLE OF CONTENTS

```
***** THIS IS A FABULOUS PROGRAM *****

YOU TYPE IN A NUMBER AND I RESPOND WITH A MESSAGE.

HOW MANY MESSAGES WOULD YOU LIKE TYPED? 5

HERE WE GO. FOLLOW INSTRUCTIONS.


TYPE 1,2,OR 3? 1
COMPUTERS ARE FUN.

TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.

TYPE 1,2,OR 3? 3
I LIKE YOU.

TYPE 1,2,OR 3? 4
FOLLOW DIRECTIONS DUMMY.

TYPE 1,2,OR 3? 3
I LIKE YOU.

TYPE 1,2,OR 3? 1
COMPUTERS ARE FUN.

THAT'S ALL FOLKS.
```

1

MASTERING YOUR OBEDIENT SERVANT

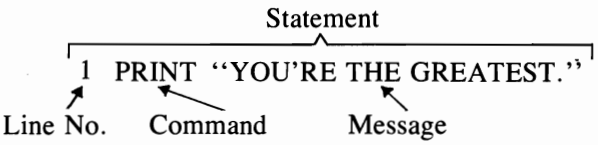
1.1 WRITING THREE EASY PROGRAMS

An easy and fun way to begin programming is to write little programs which instruct the computer to type out a message. Let's begin with the message YOU'RE THE GREATEST. Later on you'll have a chance to program the computer to type out your favorite message.

Before getting into the details of your first program, let me describe briefly the concept of a program. A program is essentially a set of statements or instructions which are to be executed by the computer. The instructions are written in a language which the computer can understand. There are many languages by which humans can communicate with computers. In this book we will use the BASIC language. BASIC is an acronym for Beginner's All-Purpose Symbolic Instruction Code. It uses ordinary English words, such as LET, PRINT, GO TO, STOP, END, READ, DATA, INPUT, FOR, NEXT, IF-THEN, to trigger particular operations by the computer. Each chapter in this book will introduce you to a few words in BASIC until you have mastered the complete BASIC vocabulary. Now back to writing your first program.

In BASIC each line of a program must begin with a whole number known as the line number. The line number is followed by a command consisting of one or two English words, such as LET or PRINT. Some commands, such as STOP and END, are sufficient by themselves, but most must be followed by additional information. Using these rules to structure each line, let's write the program to accomplish the objective given above. Write the program on paper before typing it into the computer.

First we need a line number. Let's use 1. Next we need a command. To instruct the computer to type out the message YOU'RE THE GREATEST we use the English word PRINT. Following the command you must insert, in quotes, the text of the message itself: "YOU'RE THE GREATEST." The first line in your program looks like this.



When the computer executes this statement, it will type out the string of characters between the quotes. If you misspell one of the words in the message, it will be typed back with the misspelling. The computer does not know the meaning of what you're trying to say; it is only your obedient servant.

The message is to be typed out only once. Since the computer is not very bright, we must tell it when its job is done. We use the command END, as follows.

```
2 END
```

Once typed, the complete two-line program will look like this.

GREATEST/1 ***PROGRAM***

```
1 PRINT "YOU'RE THE GREATEST."
2 END
```

When you want the computer to carry out the two steps, you type RUN.

```
RUN
```

OUTPUT

```
YOU'RE THE GREATEST.
```

Suppose you want to insert ahead of the message YOU'RE THE GREATEST another message which says THIS IS A FABULOUS PROGRAM. But you've already used line 1. You must retype the program GREATEST/1 as follows:

FABULOUS/1 ***PROGRAM***

```
1 PRINT "THIS IS A FABULOUS PROGRAM."
2 PRINT "YOU'RE THE GREATEST."
3 END
```

OUTPUT

```
THIS IS A FABULOUS PROGRAM.
YOU'RE THE GREATEST.
```

The old lines 1 and 2 were automatically erased in the computer and replaced by the new lines 1 and 2.

You'll find that many times you would like to insert one or more lines between two others. To make such insertions without retyping existing lines, it is much better to select as the original line numbers a sequence of numbers such as 5, 10, 15, 20, . . . or 10, 20, 30, 40. . . . Then you can insert one or more new lines between two old lines. Incidentally, you don't have to type in the lines of a program in numerical order. For example, the last program could have been typed like this.

FABULOUS/2 ***PROGRAM***

```
2 PRINT "YOU'RE THE GREATEST."
3 END
1 PRINT "THIS IS A FABULOUS PROGRAM."
```

When it comes time to run your program, the computer will execute the program steps in numerical order.

OUTPUT

```
THIS IS A FABULOUS PROGRAM.
YOU'RE THE GREATEST.
```

To erase lines 1, 2 and 3, in fact all previous program lines, type SCRATCH. Here is the program GREATEST/1 rewritten using line numbers 10 and 20:

```
10 PRINT "YOU'RE THE GREATEST."          PROGRAM        GREATEST/2
20 END

                YOU'RE THE GREATEST.     OUTPUT
```

To add the message THIS IS A FABULOUS PROGRAM before line 10, select a line number between 1 and 9 inclusive, say 5, and type in:

5 PRINT "THIS IS A FABULOUS PROGRAM."

You now have the following new program.

```
 5 PRINT "THIS IS A FABULOUS PROGRAM."   PROGRAM        FABULOUS/3
10 PRINT "YOU'RE THE GREATEST."
20 END

          THIS IS A FABULOUS PROGRAM.    OUTPUT
          YOU'RE THE GREATEST.
```

Look at the output. To separate the two messages, you can direct the computer to skip a line between the first and second lines of the output by inserting

7 PRINT

Used alone, the command PRINT will cause the computer to print a blank line, as shown here.

```
 5 PRINT "THIS IS A FABULOUS PROGRAM."   PROGRAM        FABULOUS/4
 7 PRINT
10 PRINT "YOU'RE THE GREATEST."
20 END

          THIS IS A FABULOUS PROGRAM.    OUTPUT

          YOU'RE THE GREATEST.
```

So much for our first program. Now let's instruct the computer to type the same message many times.

Write a program that instructs the computer to type out the message YOU'RE THE GREATEST endlessly. Here is the new program accompanied by a sample output.

PROGRAM

```
10 PRINT "YOU'RE THE GREATEST."
20 GO TO 10
30 END
```

OUTPUT **GREATEST/ENDLESS**

```
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE T
↑C
```

As you will quickly discover, the message YOU'RE THE GREATEST continues to be typed out until you signal the computer to stop. I signal my computer to stop by pressing the "control" key and the letter C, a combination called CONTROL C. This causes the computer to stop, but only after it prints the symbols ↑ C (read "up arrow C"), as shown in the last output. How do you signal your computer to stop running a program?

The new program GREATEST/ENDLESS contains a new command, GO TO. GO TO is used to transfer control of the computer from one line in a program to another. It is always followed by the number of the line to which control is being transferred. Of course the line to which control is being transferred must exist in the program. For example, if we had mistakenly typed 20 GO TO 5, the computer would reply with an error message equivalent to LINE 5 UNDEFINED.

In the program GREATEST/ENDLESS, after printing the message, the computer moves to the next line, line 20, sees the GO TO 10 command, and transfers control back to line 10. This process continues endlessly until the human intervenes, the power fails, or the computer "crashes." So much for our second program.

Now let's get the computer to type out the same message a prescribed number of times, in this case seven.

Write a program that instructs the computer to type out the message YOU'RE THE GREATEST seven times.

GREATEST/7 ***PROGRAM***

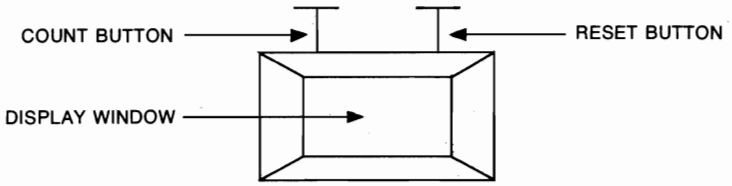
```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 LET C=C+1
40 IF C<7 THEN 20
50 END
```

OUTPUT

```
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
```

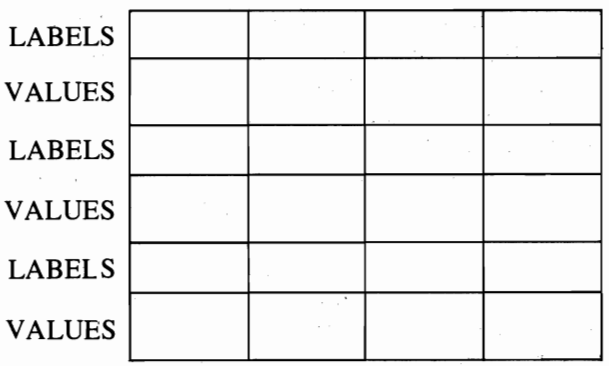
To understand the construction and operation of this program, visualize a mechanical counter. The counter has two buttons. One button, when pressed, adds 1 to whatever value is in the display window of the counter

at that time. The second button is a "reset" button which allows the user to set the counter back to zero. Here is a picture of the counter:

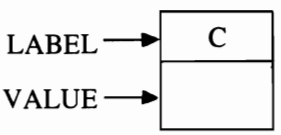


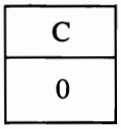
How would you use such a counter in real life? Suppose you wanted to count the number of people entering a theatre. You could stand at the door with your counter, having first made sure you had reset the counter to zero, and each time a person entered, depress the "count" button. After five people had passed, the number 5 would be in the window, and after twelve people had passed, 12 would be in the window. I think you have the idea. Well, we're going to do the same thing with the computer. Using a program we're going to set up a "counter" inside the computer to count the number of times the message YOU'RE THE GREATEST is printed. When the number in the "counter window" is 7, we know the message has been printed seven times.

Here is how the computer imitates the mechanical counter. The computer can store numbers in its memory. It will help if you imagine that a computer's memory looks like a set of post office boxes at the local post office. Each post office box has both a place for an identification label, i.e. the P.O. Box No., and a place to store your mail. In the computer, each box is labeled with a letter, and the contents is a number. The box labels are also called variables, a term used in mathematics to refer to a place holder for numbers. Here's a picture of a twelve-box memory.



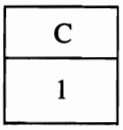
In this particular program I'll need only one box for a counter. I'll label it C. Use as labels letters which help you remember the function of the variable.





To operate the counter, it must first be set at zero. This is easily done in BASIC by saying 10 LET C = 0. In the program GREATEST/7, this was our first instruction to the computer. When the computer executes this step, a 0 is put in the storage box C.

So much for line 10 which sets the counter at zero, or as we say *initializes* it at zero. The next statement is 20 PRINT "YOU'RE THE GREATEST." As soon as the message is printed, the "count button" should be "depressed." The instruction to do this is contained in 30 LET C = C+1. To the computer this means: let the new value of C be equal to the old value of C plus 1.



After the message has been printed once, the contents of the box labeled C is 0 plus 1, or 1.

The statement, 30 LET C = C+1, is a strange looking expression. If you interpret it literally, it makes no sense. How can a number be equal to itself plus 1? But it's not algebra. It's an order to the computer, and the computer reads it from right to left. First figure out what C+1 is, then assign this number to the variable on the left-hand side of the equal sign.

The final step of the program is a test to determine whether or not the message has been printed seven times. Recall the mechanical counter. If the number in the window was less than 7, you knew the event you were counting had not yet occurred the seventh time. The instruction in BASIC is 40 IF C < 7 THEN 20. If the value of C is less than 7, then control of the computer is transferred back to line 20. If the value is not less than 7, that is if it is 7 or more, then the computer drops down to the next and last step: 50 END.

In short, the computer continues to loop back to line 20 as long as the value of C is less than 7. When C equals 7 the loop is satisfied and the computer drops through to the END statement. Both this program, GREATEST/7, and the previous one, GREATEST/ENDLESS, are examples of loops. The program GREATEST/ENDLESS sends the computer around the loop endlessly, while the program GREATEST/7 sends the computer back to print the message only under the condition that the total number of messages printed has not exceeded seven. Thus, GREATEST/7 is an example of a *conditional* loop. You'll write many conditional loops.

Now it's time to go "on line" to your computer. The procedure for getting on line varies from one kind of computer to another. To learn how to get on line to your computer, and how to enter a BASIC program, consult your user's manual or ask someone to show you. Don't be afraid to sit down and take control. In no time at all getting on line and entering a program will be second nature.

EXERCISE SET 1.1

The exercises in this book are specified as either off-line exercises or on-line exercises. Off-line exercises do not require a computer. In on-line exercises you will write, load, run and perfect a program. The number of exercises has been kept to a minimum and you should do them all.

Off-line:

To find the answers to each of these off-line exercises, consult the manual for your computer system or ask a knowledgeable friend.

1. Describe the steps to be followed to get yourself on line to your computer and into BASIC.
2. If your system requests a program name before entering the program, how long can the program name be and what set of characters can you use in the name?
3. Describe the function of the words RUN, LIST, and SCRATCH. Also OLD and NEW if applicable.
4. Suppose you make a typing error as you're typing in your program. How do you correct your mistake?
5. Suppose you've typed in a line which you wish to eliminate. How can you delete this line from your program?

On-line:

6. Compose your own message and write a program that will type out your message exactly once.
7. Compose your own message and write a program that will type out your message endlessly. However, do not let it run forever; stop the program when you've seen enough.
8. Compose your own message and write a program that will type out your message exactly 11 times.

1.2 GETTING THE BUGS OUT

Even experienced programmers make errors. Errors in programming are called "bugs." Bugs can make you unhappy. If you don't get the bugs out of your program quickly, you won't have much fun with the computer. The process of locating and eliminating these errors is known as "debugging." If you've already been on line and have run the three programs requested, you will readily appreciate the following discussion.

The types of errors which you can make are many, but some kinds are more easily spotted and removed than others. Here are three kinds of errors.

1. Typing Errors
2. Sentence Structure Errors
3. Logical Errors

1.2.1 TYPING ERRORS

In your haste to type in a program you will often misspell words, type the wrong symbols, or leave out essential information. Spelling errors are not too important if they occur in the message; the computer will just spell the words the way you do. But if you misspell one of the BASIC command words, such as LET or PRINT, the computer will not recognize it. It will respond with an error message, either immediately after you hit the RETURN key, or later when you type RUN. Here are some examples of common typing errors:

	WRONG	RIGHT
1.	LETT C = 0	LET C = 0
2.	PRIN "YOU'RE THE GREATEST."	PRINT "YOU'RE THE GREATEST."
3.	LET C = C=1	LET C = C+1
4.	IF C < 7 THEN	IF C < 7 THEN 20

A more difficult error to spot occurs when you inadvertently type the letter O for the number zero, or the letter L or I for 1. To help you distinguish zero from the letter O, the zero key will often contain a slash through the zero: Ø. You'll catch most of these careless errors as you type, or by quickly rereading the program before you type RUN. But some typing mistakes are not so easy to detect. Here's an example. This program is intended to print the message YOU'RE THE GREATEST seven times.

GREATEST/BUG 1 ***PROGRAM***

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 LET C=C+1
40 IF C<7 THEN 40
50 END
```

When you enter the program and type RUN, the computer will type out YOU'RE THE GREATEST once and then stop printing. You might sit there for quite some time waiting for additional copies of the message. After a while you might decide that the computer had malfunctioned, or "crashed," when in fact it had not. There's a bug in the program. It's in line 40. The instruction can be executed, but doesn't lead anywhere. The computer is caught in a loop. At line 40 it transfers control to line 40, which in turn transfers control to itself again. It stays on this merry-go-round until you interrupt execution of the program. Here's the point. If you expect a line to be printed and no action occurs within a reasonable length of time, you'd better stop the computer and find out what's happened by rereading your program more carefully. In the above case, for example, the correct statement for line 40 is: 40 IF C < 7 THEN 20.

Here is another typing error which can cause you grief. You enter the following program:

PROGRAM **GREATEST/BUG 2**

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 LET C=C+I
40 IF C<7 THEN 20
50 END
```

With this program, you expect the computer to type out the message seven times. But, when you type RUN, you get:

OUTPUT

```
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREA
↑C
```

After about ten lines have appeared, you might well suspect that something is wrong and halt the computer. Remember, any time you think the computer is going berserk, don't panic and pull the plug. Signal the device to stop. Then find the bug. Read the above program carefully. In line 30 did we use the number 1 or the letter I? Which is correct? Of course it should be the number 1. So you make the correction by retyping the line as: 30 LET C = C + 1. All is well.

However, if your eye does not catch the substitution of the letter I for the number 1, here is a technique worth remembering. Since you wrote the program, you know it should type out the same message seven times. Furthermore, you know the variable C is acting as a counter. As the computer executes your program, the variable C should take on the values 0, 1, 2, 3, 4, 5, 6, and 7 one at a time. To see whether this is happening, we insert the statement PRINT C in the program. Whenever the computer encounters this statement, it will go to the box labeled C and print the contents. But where do you insert the PRINT C statement? In this case, you can do it either before or after line 30, but it must be inserted between lines 20 and 40. I would prefer to look at the value of C after line 30 is executed. Let's use line 35. Don't write just 35 PRINT C, however. To help find the value of C in the printout, write 35 PRINT "C=" C. The characters in quotes will be printed followed by the value of C. Here is a copy of my conversation with the computer as I debugged this program.

GREATEST/BUG 2 ***CONVERSATION 1***

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 LET C=C+I
40 IF C<7 THEN 20
50 END

35 PRINT "C="C

RUN

YOU'RE THE GREATEST.
C= 0
YOU'RE THE GREATEST.
C= 0
YOU'RE THE GREATEST.
C= 0
YOU'RE THE GREATEST.
C= 0
YOU'RE THE GREATEST.
C= 0
YOU'RE THE GREATEST.
C= 0
YOU'RE T
'C
```

As I watch this printed out, I see the number zero appearing as the value of C time after time after time. Something is wrong. C is not increasing by 1 each time the message is printed. This suggests to me that I should look carefully at the line which increases the counter, that is line 30. I retype line 30 as 30 LET C = C + 1. Having corrected line 30, I verify that C now takes on the values 1, 2, 3, 4, 5, 6, and 7 by running the program while retaining the "see through" PRINT statement in line 35.

GREATEST/BUG 2 ***CONVERSATION 2***

```
30 LET C=C+1

RUN

YOU'RE THE GREATEST.
C= 1
YOU'RE THE GREATEST.
C= 2
YOU'RE THE GREATEST.
C= 3
YOU'RE THE GREATEST.
C= 4
YOU'RE THE GREATEST.
C= 5
YOU'RE THE GREATEST.
C= 6
YOU'RE THE GREATEST.
C= 7
```

Convinced that my program is operating correctly, I delete line 35 by typing the number 35 with nothing

after it and have as my "clean copy" the following "bugless" program.

35 ***CONVERSATION 3*** **GREATEST/BUG 2**

```
LIST

  10 LET C=0
  20 PRINT "YOU'RE THE GREATEST."
  30 LET C=C+1
  40 IF C<7 THEN 20
  50 END
```

These two examples show you why programmers sometimes spend more time getting the bugs out than writing the program in the first place.

EXERCISE SET 1.2.1

On-line:

1. Load into your computer the program GREATEST/BUG 1, including the bug in line 40. Run the program to see how your computer reacts. Then debug the program by retyping line 40 and running it again. Finally, get the computer to generate a clean copy of the program.
2. Load into your computer the program GREATEST/BUG 2, including the error in line 30.
   (1) Run this program.
   (2) Insert the "see through" PRINT statement 35 PRINT "C =" C to check on the values of C.
   (3) Run the program again.
   (4) Correct line 30.
   (5) Run the program again to check on the values of C.
   (6) Delete line 35.
   (7) Run the program again.
   (8) Get a clean copy of the program.

1.2.2 SENTENCE STRUCTURE ERRORS

A second error you're likely to make is an error in sentence structure or syntax. As shown in section 1.1, each line of a BASIC program begins with a line number. The line number is followed by a sentence beginning with a BASIC command, such as LET, PRINT, IF-THEN, GO TO, END. A sentence that begins with LET has a different structure than one that begins with GO TO, or any other command. Here are the rules.

1. LET. Here is a LET statement: LET C = C + 1. A LET statement, known as an *assignment* statement, always has the structure:

$$\text{LET}\ \underbrace{\qquad}_{\substack{\text{variable}\\ \text{name}}} = \underbrace{\qquad}_{\substack{\text{arithmetic}\\ \text{expression}}}$$

In the example, what is the variable name? What is the arithmetic expression? For now, we will limit the variable names to single letters of the alphabet. Further, let us agree that an arithmetic expression can be a number or a collection of numbers and known variables joined together by the arithmetic operations of addition (+), subtraction (−), multiplication (*), and division (/).

2. PRINT. Here is a PRINT statement: PRINT "C="C. A PRINT statement, known as an *output* statement, has the structure:

PRINT _____
(variable name)

or, PRINT "_____"
(string of characters)

or, PRINT "_____" _____
(string of characters) (variable name)

or, PRINT _____ "_____"
(variable name) (string of characters)

or, PRINT

3. GO TO. Here is a GO TO statement: GO TO 40. A GO TO statement, also known as an *unconditional transfer* statement, has the structure:

GO TO _____
(program line number)

4. IF-THEN. Here is an IF-THEN statement: IF C < 7 THEN 20. An IF-THEN statement, also known as a *conditional transfer* statement, has the structure:

IF _____ _____ _____ THEN _____
(arithmetic expression) (one of the relations) (arithmetic expression) (line number in program)

= equal to
> greater than
< less than
> = greater than or equal to
< = less than or equal to
< > not equal

5. END. An END statement, known as a *control* statement, has the structure:

END

So much for the rules of sentence structure. Here are some errors you might make.

1. 10 PRINT "YOU'RE THE GREATEST. You forgot the second quotation mark.
2. 20 "YOU'RE THE GREATEST." You forgot the PRINT command.
3. 30 LET C + 1 = C. Should be 30 LET C = C + 1.
4. 20 TYPE "YOU'RE THE GREATEST." Illegal because TYPE is not a BASIC command.
5. LET C = C + 1. You forgot the line number.
6. 40 LET C > C + 1. Should be 40 LET C = C + 1.
7. 30 LET C = C + 1 IF C < 7 THEN 20. You forgot to press the return key at the end of the first statement.
8. 60 RUN. Illegal because RUN is not a BASIC program command; it is a BASIC system command.
9. 30 LET CT = CT + 1. Illegal because a variable name cannot be a two-letter combination.

This is just a sample of the errors you can make in sentence structure. Whenever you make a mistake of this type, the computer will always print out the number of the line where the syntax error has occurred. The computer, you see, cannot execute your program unless it can translate these BASIC statements into machine code, and it cannot translate these statements into machine code unless each statement satisfies the structure required by the lead word in the sentence.

EXERCISE SET 1.2.2

Off-line:

1. Which of the following statements would not be acceptable in a BASIC program? To check each statement for bugs, look at (1) the line number, (2) the command after the line number, and (3) the symbols used with the command. Remember, this list is not a program. Consider each statement individually.

```
A) 13 LET C=1

B) 41 TYPE "YOU'RE THE GREATEST."

C) 25 PRINT

D) 45 DONE

E) 1 PRINT "HELLO

F) 132 IF C<7 THEN

G) 22 PRINT "C="C

H) 150 PRINT "END"

I) 40 GO TO 10

J) 20 PRINT "LOOK AT THE LINE NUMBER."

K) 40 IF 7>C THEN 20

L) 500 END

M) 30.5 GO TO 20

N) 30 PRINT C=C

O) LET C=0

P) 50 WRITE "GOOD BYE."

Q) 70 RUN

R) 90 IF C=7 THEN LET C=C+1

S) 50 GO TO 20

T) 30 LET C=1+C

U) PRINT "HELLO"
```

2. There are three kinds of words you use in communicating with the computer: (1) Words used as commands in a BASIC program. Call them BASIC *program* words. (2) Words which instruct the computer to do something with your program. Call them BASIC *system* words. (3) Words which instruct the computer to accept you as a user, obtain a particular language for you, or perform some other task. Call them *computer system* words. There are many computer systems, each with a unique set of *computer system* words. Consult your computer manual for the computer system words used with your system.

   Here is a list of BASIC program words and BASIC system words. Write each word in the appropriate column. In the third column list the computer system words you have learned so far.

SCRATCH	LET	OLD	CONTROL C (Or the equivalent command on your computer.)
PRINT	BYE	LIST	
RUN	NEW	END	
DELETE	IF-THEN	GO TO	

BASIC program words	BASIC system words	Computer system words

1.2.3 LOGICAL ERRORS

When I wrote the following program, I expected the message to be typed out seven times.

GREATEST/BUG 3 ***PROGRAM***

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 IF C<7 THEN 50
40 STOP
50 LET C=C+1
60 GO TO 20
70 END
```

As a matter of fact the program prints the message eight times.

OUTPUT

```
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
YOU'RE THE GREATEST.
```

I've made a logical error in setting up the program. Can you find my mistake?

The bug is not a result of using the new BASIC word STOP. STOP is used whenever you wish to end a program at some point other than the end. For example, in this program, if C is less than 7 in line 30, the computer should go around the loop again. On the other hand, if C is *not* less than 7, that is, if it is 7 or more, the computer should stop, which it does in the next step: 40 STOP. The advantage of 40 STOP over 40 GO TO 70 (the END statement) is its brevity and its visibility in the program.

Now back to the bug. If you're an experienced programmer you have probably found the bug. If not, let me show you a way to find it. It is the technique of longhand execution. Sit down in a quiet place away from your terminal with a clean, up-to-date copy of your program. Read your program line by line, carrying out in your head each instruction. At the same time, keep track of (1) the values of any variables and (2) the output as it occurs. A piece of paper may be helpful here. Later on, when you have more experience, you'll find it easy to read programs and to keep track of the values of variables in your head. But when you get in trouble, use pencil and paper.

Let me illustrate this process of "playing computer." There is only one variable in this program, so set up on your sheet of paper a column headed C in which to record the values of C as you calculate them in your head. Reserve another area on the paper to record the output. Your work sheet may well look like this:

PROGRAM:

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 IF C<7 THEN 50
40 STOP
50 LET C=C+1
60 GO TO 20
70 END
```

LONGHAND BOOKKEEPING:

C

OUTPUT AREA:

Now execute the program line by line. After you execute line 20 the second time, your work sheet should look like this:

PROGRAM:

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 IF C<7 THEN 50
40 STOP
50 LET C=C+1
60 GO TO 20
70 END
```

LONGHAND BOOKKEEPING:

C
Ø
1

OUTPUT AREA:
YOU'RE THE GREATEST.

Each time you compute a new value of C, cross out the old value and write the new value below it. Instead of writing the message each time, it's sufficient to write it once and then use check marks to keep track of repetitions. Continue executing the program longhand. Here is the picture after line 20 has been executed for the fifth time.

PROGRAM:

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 IF C<7 THEN 50
40 STOP
50 LET C=C+1
60 GO TO 20
70 END
```

LONGHAND BOOKKEEPING:

C

~~0~~
~~1~~
~~2~~
~~3~~
4

OUTPUT AREA:
YOU'RE THE GREATEST.

When you finally reach the STOP statement, your work sheet should look like this:

PROGRAM:

```
10 LET C=0
20 PRINT "YOU'RE THE GREATEST."
30 IF C<7 THEN 50
40 STOP
50 LET C=C+1
60 GO TO 20
70 END
```

LONGHAND BOOKKEEPING:

C

~~0~~
~~1~~
~~2~~
~~3~~
~~4~~
~~5~~
~~6~~
7

OUTPUT AREA:
YOU'RE THE GREATEST.

The message was to have been printed seven times. How many times was it printed? Eight times is one too many! If eight messages is one too many, then C is overshooting the target value by 1. The bug is in line 30.

The test value 7 should be 6. To eliminate this bug you retype line 30 as: 30 IF C < 6 THEN 50. This one change will fix the program.

There is a second way to fix the program which more closely follows the sequence of steps you used to operate the mechanical counter: (1) Set the counter at zero, (2) display the message, (3) increase the counter, and (4) test the counter. To effect this change:

(1) In line 25, insert the step to increase the counter.

```
25 LET C = C + 1
```

(2) In line 30, if C is less than 7, control should be transferred back to the PRINT statement in line 20. Change the line number after the word THEN from 50 to 20:

```
30 IF C < 7 THEN 20
```

(3) Lines 40, 50 and 60 are no longer needed. Delete them.

GREATEST/BUG 3 now looks like this.

```
10 LET C = 0
20 PRINT "YOU'RE THE GREATEST."
25 LET C = C +1
30 IF C < 7 THEN 20
70 END
```

To sum up, when writing conditional loops, there is less chance of a bug creeping into your program if you put the four steps in this order.

(1) A LET statement which initializes the control variable at some beginning value. In this case: 10 LET C = 0.
(2) The statement which is to be repeated a specified number of times. In this case: 20 PRINT "YOU'RE THE GREATEST."
(3) A LET statement which increases (or even decreases) the control variable. In this case: 25 LET C = C + 1.
(4) An IF-THEN statement, called an exit test, which tests the value of the control variable to determine if the loop has been satisfied or completed. In this case: 30 IF C < 7 THEN 20.

Now for some practice at debugging.

EXERCISE SET 1.2.3

Off-line:

1. Each of the following programs was intended to type out the message I'M AN IDIOT exactly eight times. There are no errors due to spelling or sen-

tence structure. "Play computer" to determine which programs produce the desired output. If the program does not print the message I'M AN IDIOT eight times, correct it by making as few corrections as possible.

A.
```
10 LET C=0
20 PRINT "I'M AN IDIOT."
30 LET C=C+1
40 IF C<8 THEN 20
50 END
```

B.
```
1 LET C=0
2 PRINT "I'M AN IDIOT."
3 LET C=C+1
4 IF C<=8 THEN 2
5 END
```

C.
```
100 LET C=1
110 PRINT "I'M AN IDIOT."
120 IF C<8 THEN 110
130 LET C=C+1
140 END
```

D.
```
13 LET C=1
26 LET C=C+1
39 PRINT "I'M AN IDIOT."
52 IF C<=8 THEN 26
65 END
```

E.
```
 5 LET C=1
10 PRINT "I'M AN IDIOT."
15 IF C<=8 THEN 25
20 STOP
25 LET C=C+1
30 GO TO 10
35 END
```

F.
```
10 LET C=1
20 PRINT "I'M AN IDIOT."
30 LET C=C+1
40 IF C<=10 THEN 20
50 END
```

G.
```
20 LET C=1
40 PRINT "I'M AN IDIOT."
60 LET C=C+1
80 IF C<=8 THEN 20
100 END
```

On-line:

2. Each of the following programs was intended to type out the message HELLO exactly five times. There are no errors due to syntax or spelling. Load and run each program. If a program does not generate the message HELLO five times, correct it on line, making as few corrections as possible.

A.
```
10 LET X=0
20 PRINT "HELLO"
30 LET X=X+1
40 IF X<=4 THEN 20
50 END
```

B.
```
10 LET P=1
20 PRINT "HELLO"
30 LET P=P+1
40 IF P<=5 THEN 20
50 END
```

C.
```
10 LET D=0
20 PRINT "HELLO"
30 IF D=5 THEN 60
40 LET D=D+1
50 GO TO 20
60 END
```

D.
```
10 LET C=36
20 PRINT "HELLO"
30 LET C=C+1
40 IF C<=41 THEN 20
50 END
```

E.
```
10 LET M=0
20 PRINT "HELLO"
30 LET M=M+1
40 IF M<=4 THEN 50
50 GO TO 20
60 END
```

F.
```
10 LET C=1
20 PRINT "HELLO"
30 IF C<6 THEN 50
40 STOP
50 LET C=C+1
60 GO TO 20
70 END
```

1.3 LOOP BUILDING

At the end of the last section we listed the four steps used in building a conditional loop. Now I'll show you an easier way to build conditional loops using two new BASIC words, FOR and NEXT. To illustrate, here is an old friend, the program to type out our favorite message YOU'RE THE GREATEST seven times.

PROGRAM **GREATEST/7**

```
C      10 LET C=0
LOOP   20 PRINT "YOU'RE THE GREATEST."
       30 LET C=C+1
       40 IF C<7 THEN 20
       50 END
```

In the following program the two statements that begin with FOR and NEXT perform the same functions as LET C = 0, LET C = C + 1, and IF C < 7 THEN 20 in the above program.

PROGRAM **GREATEST/FOR**

```
C      10 FOR C=0 TO 6
LOOP   20 PRINT "YOU'RE THE GREATEST."
       30 NEXT C
       40 END
```

The statement FOR C = 0 TO 6 in line 10 instructs the computer to perform the following operations. The corresponding statements in the old program are given in parentheses after each operation.

1. C is the control variable and its beginning value is 0. (LET C = 0).
2. The increment or step between successive values of the control variable is plus 1. (LET C = C + 1)
3. The final value of the control variable is 6. (IF C < 7 THEN 20)

The step to be repeated, the PRINT statement, follows the FOR statement. The statement NEXT C on line 30 ends the loop.

When the computer executes the program GREATEST/FOR, C is set at 0, the message is printed, the next C, 1, is generated, the message is printed again, the next C, 2, is generated, the message is printed again, etc. This process continues while C is less than or equal to 6. When C exceeds 6, that is, when it becomes 7, the loop is satisfied and the computer drops through to the next statement, which in this case is the END statement.

I hope you like the new words, FOR and NEXT, because they do make it easier to write conditional loops. Also the logic is clearer. We just define our FOR statement, put down the message or steps to be repeated, and end the loop with the NEXT statement.

Here is another program which used FOR and NEXT to type out a message seven times.

PROGRAM **GREATEST/FOR 1-7**

```
C      10 FOR C=1 TO 7
LOOP   20 PRINT "YOU'RE THE GREATEST."
       30 NEXT C
       40 END
```

Look at the FOR statement in line 10. Instead of starting C at zero, the computer starts C at 1. Instead of stopping C at 6, it stops C at 7. From this point on, if I wish to print a message or perform some other task seven times I will initialize the control variable at 1. The final value of C will always be the same as the number of times the computer goes around the loop.

In summary, the syntax of a FOR statement is:

FOR_____ = _____ TO _____,

Name of control variable	Initial value of control variable	Final value of control variable

where the increment is understood to be plus 1. The syntax of the NEXT statement is:

NEXT_____

Name of control variable in matching FOR statement

EXERCISE SET 1.3

Off-line:

1. Play computer and determine the output of each program.

A.
```
10 FOR C=1 TO 5
20 PRINT "I LIKE YOU."
30 NEXT C
40 END
```

B.
```
10 FOR C=0 TO 6
20 PRINT "COMPUTERS ARE FUN."
30 IF C=4 THEN 50
40 NEXT C
50 END
```

C.
```
10 FOR C=25 TO 29
20 PRINT "HELLO"
30 NEXT C
40 FOR C=6 TO 7
50 PRINT
60 NEXT C
70 FOR C=10 TO 12
80 PRINT "GOOD BYE"
90 NEXT C
100 END
```

2. Look at the output you wrote down while playing computer with the program listed in part C above. Write a new program which will generate the same output, but make the initial value of the control variable in each loop 1.

On-line:

3. Write a program which contains three conditonal loops, one which produces the message COMPUTERS ARE FUN three times, one which produces the message I'M FASTER THAN YOU five times, and one which produces the message I LIKE YOU six times. You may substitute other messages if you wish. The output generated by your program should look like the output shown at the left.

```
COMPUTERS ARE FUN.
COMPUTERS ARE FUN.
COMPUTERS ARE FUN.

I'M FASTER THAN YOU.
I'M FASTER THAN YOU.
I'M FASTER THAN YOU.
I'M FASTER THAN YOU.
I'M FASTER THAN YOU.

I LIKE YOU.
I LIKE YOU.
I LIKE YOU.
I LIKE YOU.
I LIKE YOU.
I LIKE YOU.
```

1.4 BRANCHING: WRITING A PROGRAM FROM A FLOW CHART

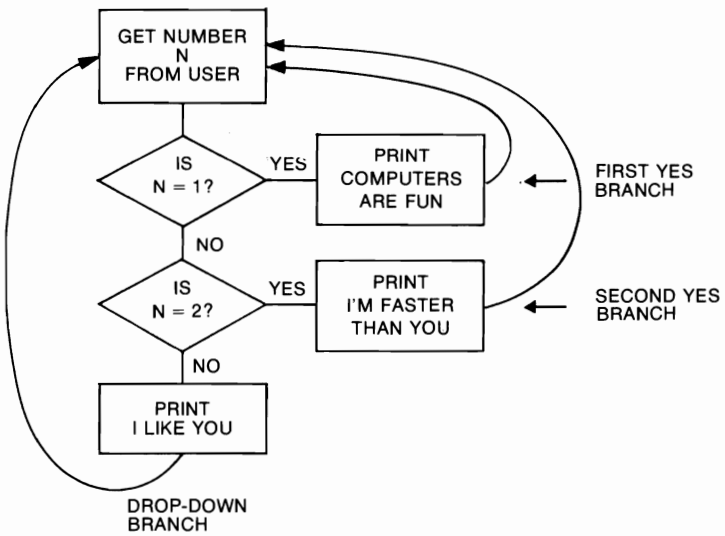
In the last section you learned two new BASIC words, FOR and NEXT, and how to use them to build simple loops. In this section you will learn how to write programs which contain several IF-THEN statements. IF-THEN statements direct the computer to follow one of two possible routes, or branches, in response to a question. Programs which contain branches can be more easily written if you work from a flow chart.

Write a program that allows the user to pick a whole number, 1, 2, or 3, so that, if he picks 1, the computer types back COMPUTERS ARE FUN, if 2, the response is I'M FASTER THAN YOU, and if 3, the response is I LIKE YOU.

What is a flow chart? In simple terms a flow chart is nothing more than a graphic description of the steps to be followed in arriving at the solution to a problem. A flow chart is a valuable aid in helping you sort out your thoughts and method of solution, by defining and displaying your thoughts in proper order.

In meeting the above objective, we must first instruct the computer to receive the number the user has picked, then determine whether the number is 1, 2, or 3, and finally print the corresponding message. Since the computer must receive a number from the user, you must designate a variable to hold a place for the number. Let's use the variable name N for the number. Look at the following flow chart.

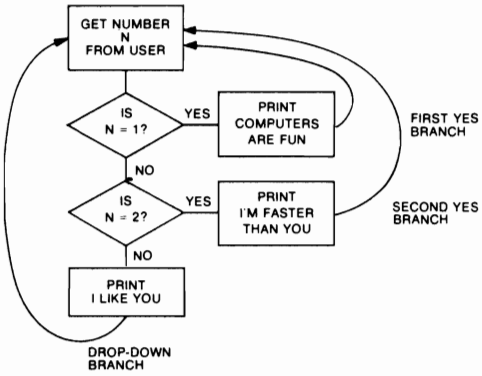
3-MESSAGE/ELEMENTARY FLOW CHART



Look at the geometric shapes used in the flow chart. Diamonds are used for questions, and rectangles are used for other instructions or directions. Now look at the

questions in the diamonds. If the answer to the question is YES, you exit from the diamond at the right. If the answer to the question is NO, you exit from the diamond at the bottom. The lines that connect the rectangles and diamonds show the path to take after completing the operation specified in a diamond or rectangle. How many possible routes or paths are there?

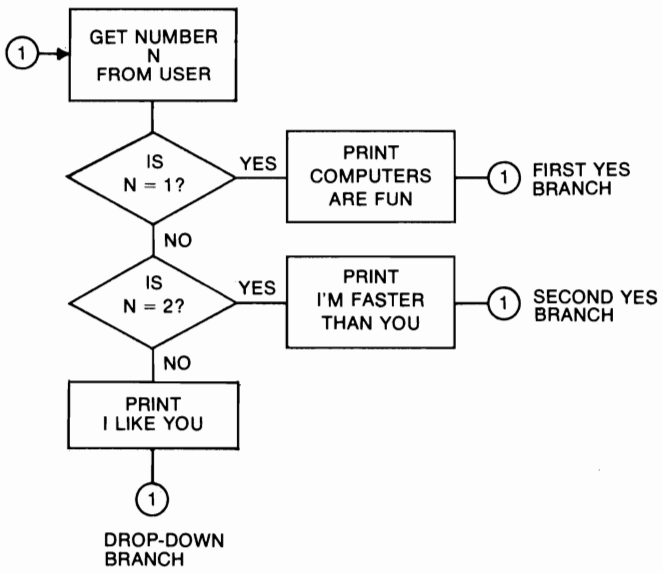
3-MESSAGE/ELEMENTARY FLOW CHART



Look at the first rectangle at the top of the flow chart. Assume the value of N is 1, and trace the path through the flow chart to find the message that will be printed. You should pass through the FIRST YES BRANCH. Trace the path to find the message that will be printed if N is 2. You should pass through the SECOND YES BRANCH. Finally, trace the path to find the message that will be printed if N is 3. You should pass through the DROP-DOWN BRANCH. In short, this flow chart has three branches, and for each value of N, 1, 2, or 3, you end up on one of the three branches. After the message is typed out, you return to the top rectangle and start again.

We can get rid of the long lines that go from the end of each branch back to the top of the flow chart. Attach to the end of each branch a small circle with the number 1 in it, and attach to the first box in the flow chart a similar circle containing the number 1. This indicates transfer of control from one point in the flow chart to another without long lines. Here is the flow chart with this refinement.

3-MESSAGE FLOW CHART

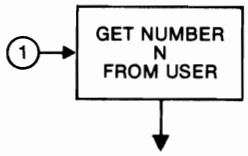


We're now ready to translate the flow chart into a program. If your flow chart is detailed enough, the translation process is one for one. That is, each instruction or question in the flow chart corresponds to one BASIC statement. Here we go. First place some

line numbers down the left-hand margin for organizational purposes.

```
10
20
30
40
50
60
70
80
90
100
```

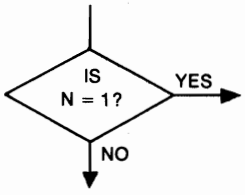
The first instruction,



is represented in BASIC by INPUT N. When the computer executes this instruction, it prints a question mark at the user's terminal and then waits for him to type in a number. The incomplete program at this point looks like this:

```
10 INPUT N
20
30
40
50
60
70
80
90
100
```

Now, we come to our first diamond.



The YES exit is translated into an IF-THEN statement as follows:

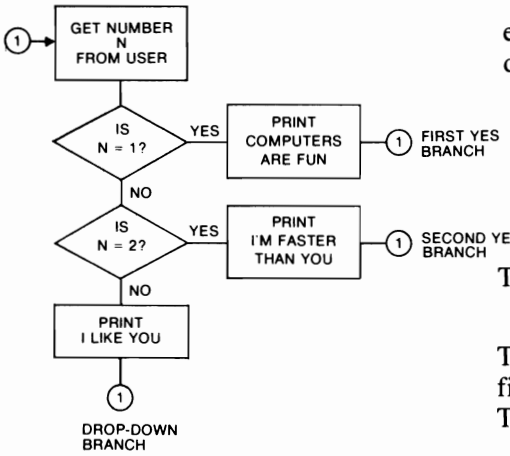
IF N =1 THEN_____

As I explained earlier, a line number is required after THEN. We'll go back and insert the line number of PRINT "COMPUTERS ARE FUN" when we add that PRINT statement to the program.

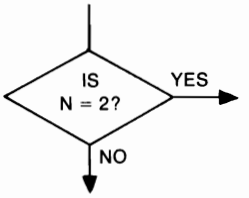
There is no program line corresponding to the NO exit. If N is not 1, control drops to the next line in the program. We'll get to that line, line 30, in a minute. Meanwhile, here is our incomplete program at this point.

```
10 INPUT N
20 IF N=1 THEN_____
30
40
50
60
70
80
90
100
```

3-MESSAGE FLOW CHART



Look again at the flow chart. When we take the NO exit from the first diamond, we drop down to the second diamond.



The YES exit from the second diamond is translated into

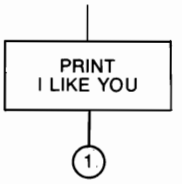
IF N=2 THEN_____

This time, the blank after THEN will eventually be filled with the line number of PRINT "I'M FASTER THAN YOU."

Again, there is no program line corresponding to the NO exit. If N is not 2, control drops to the next line in the program. We'll write that line, line 40, next. Meanwhile, the incomplete program looks like this.

```
10 INPUT N
20 IF N=1 THEN_____
30 IF N=2 THEN_____
40
50
60
70
80
90
100
```

Look again at the flow chart. When we take the NO exit from the second diamond, we drop down to the following box.



The box is translated as line 40:

PRINT "I LIKE YOU."

We're at the end of the drop-down branch. The tag hanging from the bottom of the box indicates that control is to be transferred back to the beginning of the program. To do this, we add the next step, GO TO 10, at line 50. Now our program looks like this.

```
10 INPUT N
20 IF N=1 THEN_____
30 IF N=2 THEN_____
40 PRINT "I LIKE YOU."
50 GO TO 10
60
70
80
90
100
```

That takes care of one of the three branches, the "straight through" or "drop down" branch. Let's program another branch. It could be either of the other two. I think it's easier to work back up the flow chart. Backtrack to the last diamond you passed through. It contains the question IS N=2? This question is already represented by an IF-THEN statement in line 30. Complete line 30 by filling in the blank with the next available line number; that's 60. Then write in line 60: PRINT "I'M FASTER THAN YOU." That brings us to the end of another branch. Transfer back to the beginning of the program by writing in line 70, GO TO 10. The incomplete program now looks like this.

```
10 INPUT N
20 IF N=1 THEN_____
30 IF N=2 THEN 60
40 PRINT "I LIKE YOU."
50 GO TO 10
60 PRINT I'M FASTER THAN YOU."
70 GO TO 10
80
90
100
```

That takes care of two of the three branches. Backtrack to the first diamond in the flow chart. It contains the question IS N=1? This question is already represented by an IF-THEN statement in line 20. To complete line 20, fill in the blank with the next available line number; that's 80. On line 80 write PRINT "COMPUTERS ARE FUN." This brings us to the end of the branch. Transfer control back to the beginning of the program by writing in line 90, GO TO 10. Adding the END statement, we have as our final program:

```
10 INPUT N
20 IF N=1 THEN 80
30 IF N=2 THEN 60
40 PRINT "I LIKE YOU."
50 GO TO 10
60 PRINT "I'M FASTER THAN YOU."
70 GO TO 10
80 PRINT "COMPUTERS ARE FUN."
90 GO TO 10
100 END
```

To check the accuracy of the new program, load and run it, using all the numbers, 1, 2, and 3, to test all the branches. Don't assume that if the program works for

one number, it will produce correct results for the other numbers also. If your program is correct, you will get the output shown below.

3-MESSAGE ***PROGRAM***

```
10 INPUT N
20 IF N=1 THEN 80
30 IF N=2 THEN 60
40 PRINT "I LIKE YOU."          <-- IF N = 3
50 GO TO 10
60 PRINT "I'M FASTER THAN YOU." <-- IF N = 2
70 GO TO 10
80 PRINT "COMPUTERS ARE FUN."   <-- IF N = 1
90 GO TO 10
100 END
```

OUTPUT

```
? 1                             <-- TEST N = 1
COMPUTERS ARE FUN.
? 2                             <-- TEST N = 2
I'M FASTER THAN YOU.
? 3                             <-- TEST N = 3
I LIKE YOU.
? 2                             <-- TEST N = 2
I'M FASTER THAN YOU.
? 3                             <-- TEST N = 3
I LIKE YOU.
? 1                             <-- TEST N = 1
COMPUTERS ARE FUN.
?
^C
```

If you do not get this output, you have made either (1) an error in logic, as represented in the flow chart, or (2) an error in encoding the program. You already know how to stamp out bugs.

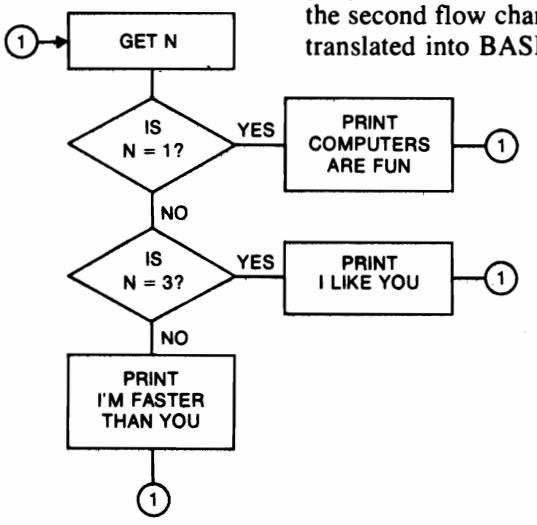
EXERCISE SET 1.4

Off-line:

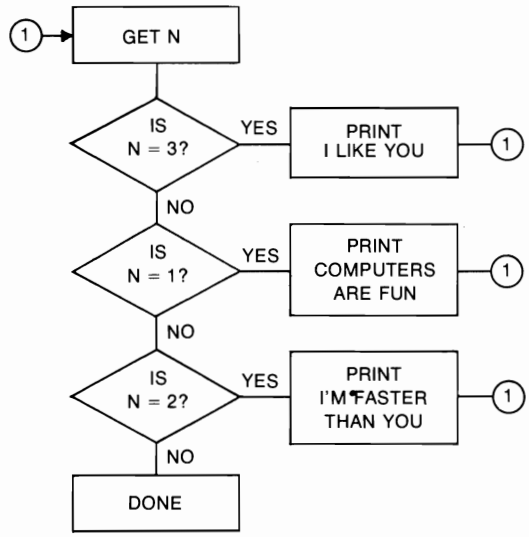
1. The following flow charts illustrate two solutions that are logically equivalent to the flow chart 3-MESSAGE. In other words, although the questions are different, the results are the same. Write a program from each of the new flow charts. Note in the second flow chart that the instruction DONE is translated into BASIC as STOP.

FLOW CHART

A.

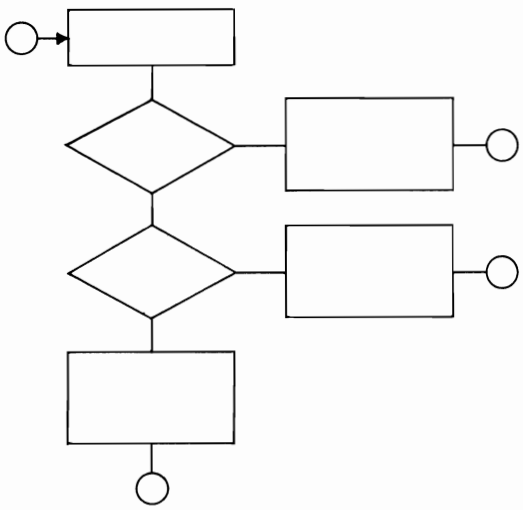


B.



FLOW CHART

2. That makes three logically equivalent solutions for 3-MESSAGE. But there are more. Fill in the following chart so as to define one of the remaining solutions. Then write a program from your flow chart.



3. Each of the following six programs was intended to produce a message in response to a number, as shown here.

INPUT (N)	OUTPUT
1	COMPUTERS ARE FUN
2	I'M FASTER THAN YOU
3	I LIKE YOU

Not all the programs generate the desired output, however. Play computer to determine the output of

each program. If the output is not as intended, make the necessary corrections to the program.

A.

```
10 INPUT N
20 IF N=1 THEN 60
30 IF N=2 THEN 50
40 PRINT "I LIKE YOU."
50 PRINT "I'M FASTER THAN YOU."
60 PRINT "COMPUTERS ARE FUN."
70 END
```

B.

```
10 INPUT N
20 IF N=1 THEN 30
30 PRINT "COMPUTERS ARE FUN."
40 GO TO 10
50 IF N=2 THEN 60
60 PRINT "I'M FASTER THAN YOU."
70 GO TO 10
80 IF N=3 THEN 90
90 PRINT "I LIKE YOU."
100 GO TO 10
110 END
```

C.

```
10 INPUT N
20 IF N=1 THEN 80
30 IF N=2 THEN 80
40 PRINT "I LIKE YOU."
50 GO TO 10
60 PRINT "COMPUTERS ARE FUN."
70 GO TO 10
80 PRINT "I'M FASTER THAN YOU."
90 END
```

D.

```
10 INPUT N
20 IF N=1 THEN 30
30 IF N=2 THEN 40
40 PRINT "I'M FASTER THAN YOU."
50 GO TO 10
60 PRINT "COMPUTERS ARE FUN."
70 GO TO 10
80 PRINT "I LIKE YOU."
90 GO TO 10
100 END
```

E.

```
10 INPUT N
20 IF N=3 THEN 80
30 IF N=1 THEN 60
40 PRINT "I LIKE YOU."
50 GO TO 10
60 PRINT "I'M FASTER THAN YOU."
70 GO TO 10
80 PRINT "COMPUTERS ARE FUN."
90 GO TO 10
100 END
```

F.

```
10 INPUT N
20 IF N<3 THEN 50
30 PRINT "I LIKE YOU."
40 GO TO 10
50 IF N<2 THEN 80
60 PRINT "I'M FASTER THAN YOU."
70 GO TO 10
80 PRINT "COMPUTERS ARE FUN."
90 GO TO 10
100 END
```

On-line:

4. Load and run the program you wrote for the second flow chart (B) in exercise 1 to verify that your program is correct.
5. Load and run the program you wrote in exercise 2 to verify the logic displayed in the flow chart.

1.5 COMPUTER PERSONALITY AND LAYOUT

Look at the output for the program 3-MESSAGE. If someone else were to run that program, he would have no idea what to do when confronted with the question mark generated by the INPUT statement. Instruct the user to type 1, 2, or 3, by adding to the program 5 PRINT "TYPE 1, 2, OR 3". To insure that line 5 is

executed each time, replace GO TO 10 in lines 50, 70 and 90 with GO TO 5.

PROGRAM

```
→  5 PRINT "TYPE 1,2,OR 3"
   10 INPUT N
   20 IF N=1 THEN 80
   30 IF N=2 THEN 60
   40 PRINT "I LIKE YOU."
→  50 GO TO 5
   60 PRINT "I'M FASTER THAN YOU."
→  70 GO TO 5
   80 PRINT "COMPUTERS ARE FUN."
→  90 GO TO 5
  100 END
```

OUTPUT **3-MESSAGE/INSTRUCT**

```
TYPE 1,2,OR 3
? 1
COMPUTERS ARE FUN.
TYPE 1,2,OR 3
? 2
I'M FASTER THAN YOU.
TYPE 1,2,OR 3
? 3
I LIKE YOU.
TYPE 1,2,OR 3
? 2
I'M FASTER THAN YOU.
TYPE 1,2,OR 3
?
↑C
```

To have the question mark appear at the end of the instruction instead of on the line following, place a semi-colon at the end of line 5. The semi-colon is a signal to the computer that there is more to be printed on the same line. In this case, the additional characters come from the INPUT statement on line 10.

PROGRAM

```
 5 PRINT "TYPE 1,2,OR 3";  ←
10 INPUT N
20 IF N=1 THEN 80
30 IF N=2 THEN 60
40 PRINT "I LIKE YOU."
50 GO TO 5
60 PRINT "I'M FASTER THAN YOU."
70 GO TO 5
80 PRINT "COMPUTERS ARE FUN."
90 GO TO 5
100 END
```

OUTPUT **3-MESSAGE/SEMI**

```
TYPE 1,2,OR 3? 1
COMPUTERS ARE FUN.
TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.
TYPE 1,2,OR 3? 3
I LIKE YOU.
TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.
TYPE 1,2,OR 3?
↑C
```

The output of the program 3-MESSAGE/SEMI is crowded. Skipping a line after each message would make the output more readable. A PRINT command with nothing after it causes the computer to skip a line in the output. Add to the program lines 45, 65, and 85, each containing a blank PRINT statement.

PROGRAM

```
   5 PRINT "TYPE 1,2,OR 3";
  10 INPUT N
  20 IF N=1 THEN 80
  30 IF N=2 THEN 60
  40 PRINT "I LIKE YOU."
→ 45 PRINT
  50 GO TO 5
  60 PRINT "I'M FASTER THAN YOU."
→ 65 PRINT
  70 GO TO 5
  80 PRINT "COMPUTERS ARE FUN."
→ 85 PRINT
  90 GO TO 5
 100 END
```

OUTPUT **3-MESSAGE/SKIP**

```
TYPE 1,2,OR 3? 1
COMPUTERS ARE FUN.

TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.

TYPE 1,2,OR 3? 3
I LIKE YOU.

TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.

TYPE 1,2,OR 3?
↑C
```

You might also like to praise the efforts of the programmer by adding this introductory message: (YOUR NAME) WROTE THIS TERRIFIC PROGRAM.

3-MESSAGE/TERRIFIC

PROGRAM

```
→1 PRINT "TED SAGE WROTE THIS TERRIFIC PROGRAM."
→2 PRINT
  5 PRINT "TYPE 1,2,OR 3";
 10 INPUT N
 20 IF N=1 THEN 80
 30 IF N=2 THEN 60
 40 PRINT "I LIKE YOU."
 45 PRINT
 50 GO TO 5
 60 PRINT "I'M FASTER THAN YOU."
 65 PRINT
 70 GO TO 5
 80 PRINT "COMPUTERS ARE FUN."
 85 PRINT
 90 GO TO 5
100 END
```

OUTPUT

```
TED SAGE WROTE THIS TERRIFIC PROGRAM.

TYPE 1,2,OR 3? 1
COMPUTERS ARE FUN.

TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.

TYPE 1,2,OR 3? 3
I LIKE YOU.

TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.

TYPE 1,2,OR 3?
↑C
```

A person who has little or nothing to say is rarely very interesting. On the other hand, a person who monopolizes the conversation is a bore. The frequency of interaction between the player and the computer, and the choice of vocabulary are two factors under your control. Balance the dialogue between the computer and the user. Don't let the computer do all the talking, and don't make the user do all the typing. Use a vocabulary which reflects your personality, or someone else's. People will like to play your games if they can laugh and have fun. But don't spend too much time on personality and readability until you are sure that your basic program works correctly.

EXERCISE SET 1.5

Off-line:

1. Play computer and determine the output of the following program.

```
10 PRINT "THIS IS A USELESS PROGRAM."
15 PRINT
20 PRINT "HOW MANY LINES";
25 INPUT N
30 FOR C=1 TO N
35 PRINT
40 NEXT C
45 PRINT "I TOLD YOU IT WAS USELESS."
50 PRINT
55 PRINT "BYE"
60 END
```

On-line:

2. The following program was written to print out the message YOU'RE THE GREATEST as many times as requested by the user. The variable N represents the number of times the message is to be printed.

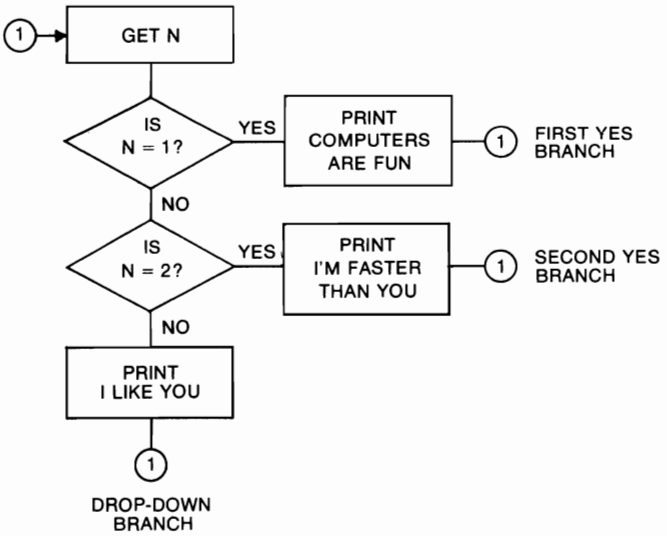
```
10 INPUT N
20 FOR C=1 TO N
30 PRINT "YOU'RE THE GREATEST."
40 NEXT C
50 END
```

Add instructions and personality to this program. Then load and test it.

1.6 "PLEASE FOLLOW DIRECTIONS"

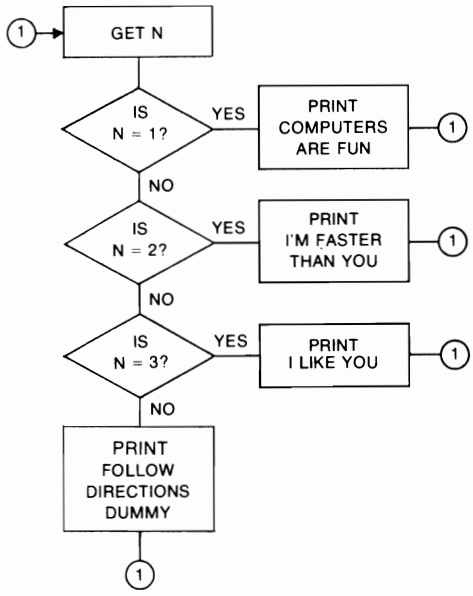
Look at the flow chart for the program 3-MESSAGE.

3-MESSAGE FLOW CHART



Suppose the user ignores the instruction to type in 1, 2 or 3, and types in 4. What message will be printed? Since N is not 1, the computer drops to the next diamond, and compares N with 2. Since N is not 2, the computer drops down to print I LIKE YOU. But I LIKE YOU is not an

3-MESSAGE/DUMMY FLOW CHART



appropriate response for a user who cannot follow directions. PLEASE FOLLOW DIRECTIONS or FOLLOW DIRECTIONS DUMMY would be more to the point. To work this into the flow chart, we need to add a third diamond in which the computer asks IS N = 3?

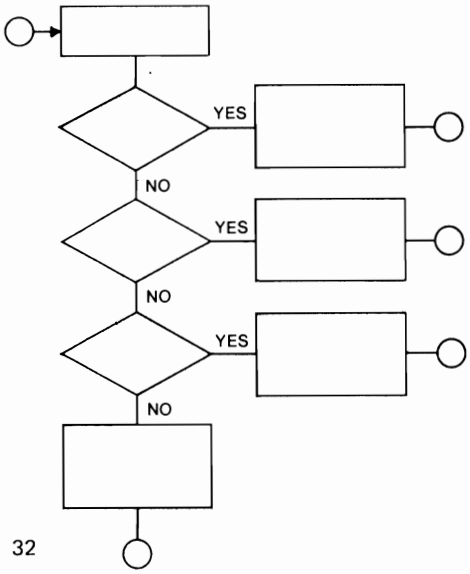
From this flow chart the following program is encoded and tested.

PROGRAM

```
5 PRINT "TYPE 1,2,OR 3";
10 INPUT N
20 IF N=1 THEN 110
30 IF N=2 THEN 90
40 IF N=3 THEN 70
50 PRINT "FOLLOW DIRECTIONS DUMMY."
60 GO TO 5
70 PRINT "I LIKE YOU."
80 GO TO 5
90 PRINT "I'M FASTER THAN YOU."
100 GO TO 5
110 PRINT "COMPUTERS ARE FUN."
120 GO TO 5
130 END
```

OUTPUT

```
TEST N = 1 ---> TYPE 1,2,OR 3? 1
                COMPUTERS ARE FUN.
TEST N = 2 ---> TYPE 1,2,OR 3? 2
                I'M FASTER THAN YOU.
TEST N = 3 ---> TYPE 1,2,OR 3? 3
                I LIKE YOU.
TEST N = 5 ---> TYPE 1,2,OR 3? 5
                FOLLOW DIRECTIONS DUMMY.
TEST N = -7 ---> TYPE 1,2,OR 3? -7
                FOLLOW DIRECTIONS DUMMY.
TEST N = 36 ---> TYPE 1,2,OR 3? 36
                FOLLOW DIRECTIONS DUMMY.
                TYPE 1,2,OR 3?
                ?C
```



EXERCISE SET 1.6

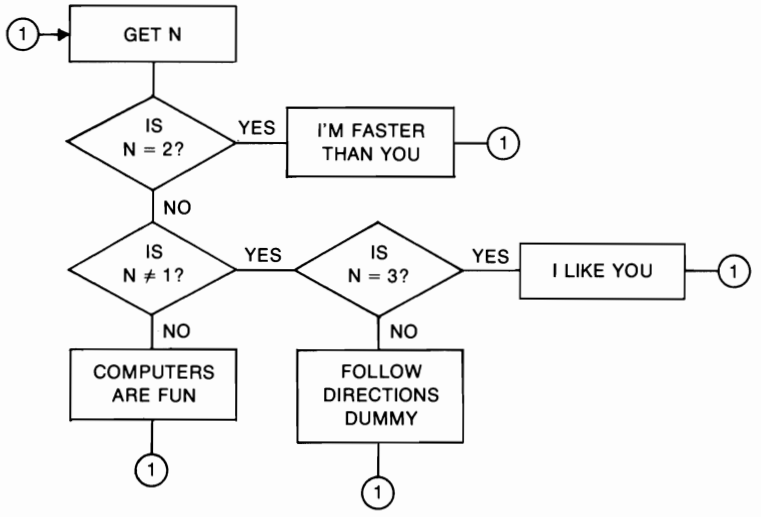
Off-line:

1. Look at the flow chart 3-MESSAGE/DUMMY. It represents only one of several logically equivalent solutions.
   A. Fill in the chart at the left to define another of these solutions.

   B. Now write a program from this flow chart.

2. Here is another flow chart which is logically equivalent to 3-MESSAGE/DUMMY. The symbol ≠ means unequal and is translated in BASIC as < >. The question IS N ≠ 1? is written IF N < > 1 THEN_____.
Write a program from this flow chart.

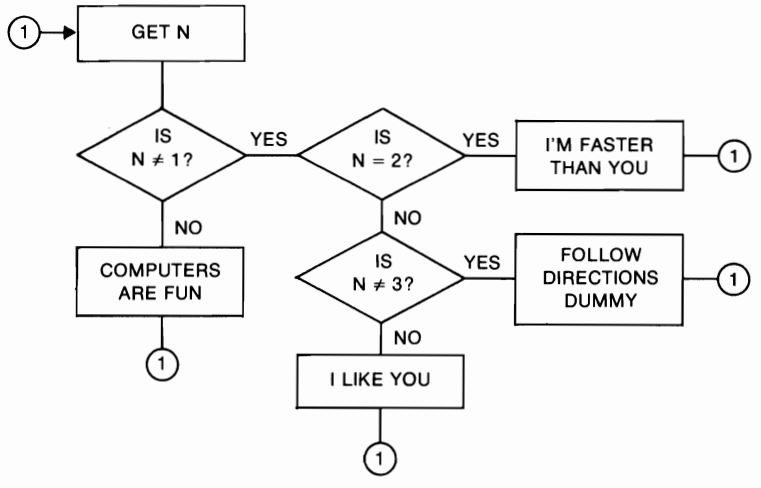
FLOW CHART



On-line:

3. Here is another flow chart which is logically equivalent to 3-MESSAGE/DUMMY.

FLOW CHART



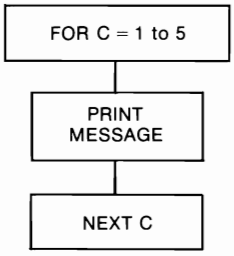
A. Write a program from this flow chart.
B. Load and run to test the validity of the logic.
C. Of the four logically equivalent solutions to the 3-MESSAGE/DUMMY problem, which do you prefer?

1.7 COMBINING LOOPS AND BRANCHES: DRAWING A FLOW CHART

When you run your 3-MESSAGE program, each time the computer types out a message it promptly asks you to pick another number, 1, 2 or 3, and out comes another message. It doesn't take long before you're bored with that activity. Five messages is about enough. To limit the computer to five messages, we'll put the 3-MESSAGE program in a loop that will type out any of the three messages, COMPUTERS ARE FUN, I'M FASTER THAN YOU, and I LIKE YOU five times, then stop. The message FOLLOW DIRECTIONS DUMMY, if it occurs, is not counted.

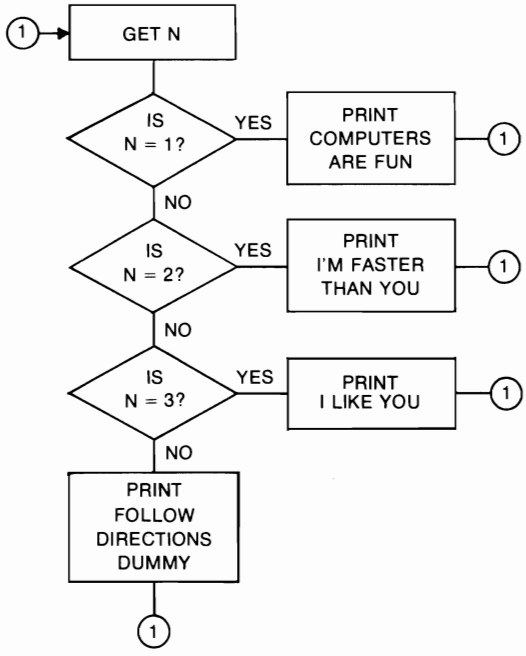
In section 3 of this chapter, Loop Building, we used a FOR-NEXT loop to type out one message a specified number of times. If only one message is to be typed five times, for example, the flow chart looks like this.

1-MESSAGE/LOOP FLOW CHART

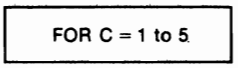


Roughly speaking, to print any three messages five times, a 3-message branching structure must be substituted for the 1-message box in the above loop. The 3-message branching structure looks like this. The fourth message is for the fellow who can't follow directions.

3-MESSAGE/DUMMY FLOW CHART

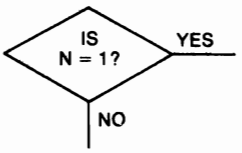


You can't just plug one flow chart into the other, however. The new flow chart is assembled this way. First draw the box

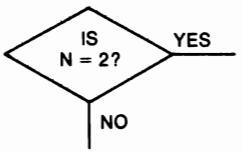


Next ask the user for a value for N.

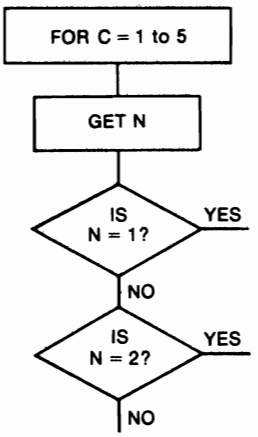
Now it is time to ask questions about the value of N. To see if N is 1, draw a diamond with the question IS N = 1? inside. Label the right exit YES and the bottom exit NO.



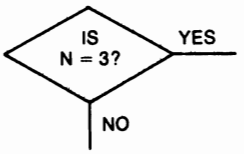
First, let's take the NO or bottom exit. If N is *not* 1, we check to see if it is 2.



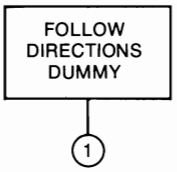
Up to this point the flow chart looks like this:



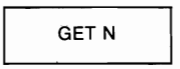
Assume N is not 2. If it is not 1 or 2, we check to see if it is 3. Draw a third diamond.



Assume N is not 3. If it is not 1, 2, or 3, the user did not follow directions. I drew the box

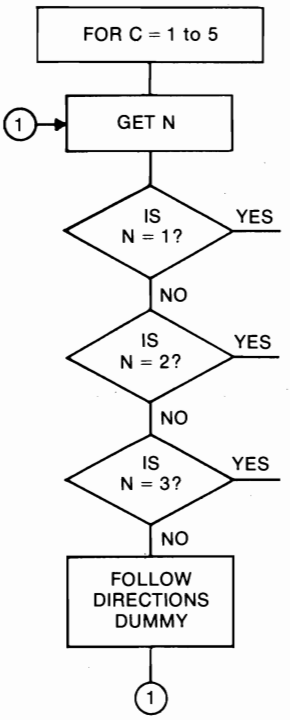


After the computer prints this message, it should ask the user for another value for N. It is directed to return to GET N by attaching an identification tag to the bottom of the FOLLOW DIRECTIONS DUMMY box. Since this is the first transfer in the flow chart, I'll put the number 1 in the small circle. I must also attach a tag to the box



Here is the flow chart to this point:

FLOW CHART



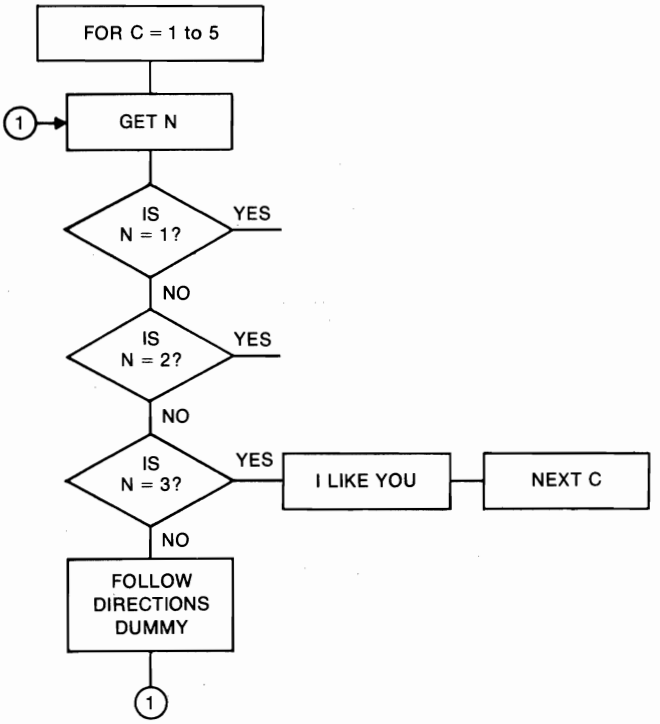
The DROP-DOWN BRANCH is now complete. Three branches, the three YES exits, remain to be completed. Let's work our way back from the bottom. This brings me to the diamond IS N=3? The YES branch of this diamond needs to be finished. If N=3 then the message I LIKE YOU should be printed. I draw

and place it at the right exit.

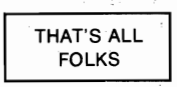
After printing the message I LIKE YOU, the computer should advance the loop counter, C, by 1. That is done by drawing the box

The flow chart now looks like this.

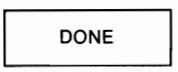
FLOW CHART



Suppose that, after printing the message I LIKE YOU, a total of five messages have been printed. The loop is therefore satisfied and the computer should stop. Before it stops, however, I want it to sign off with the message THAT'S ALL FOLKS. I add to the chart

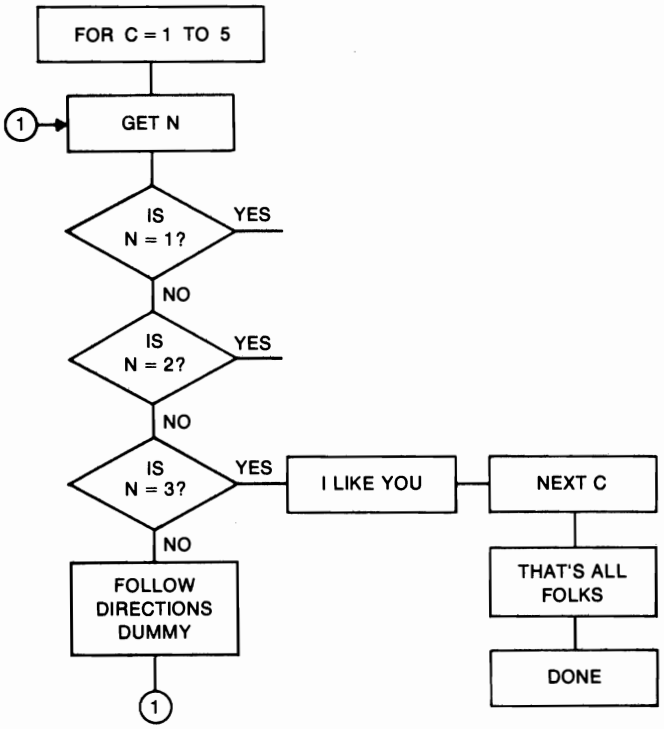


Following this, I draw

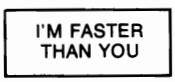


The flow chart now looks like this.

FLOW CHART

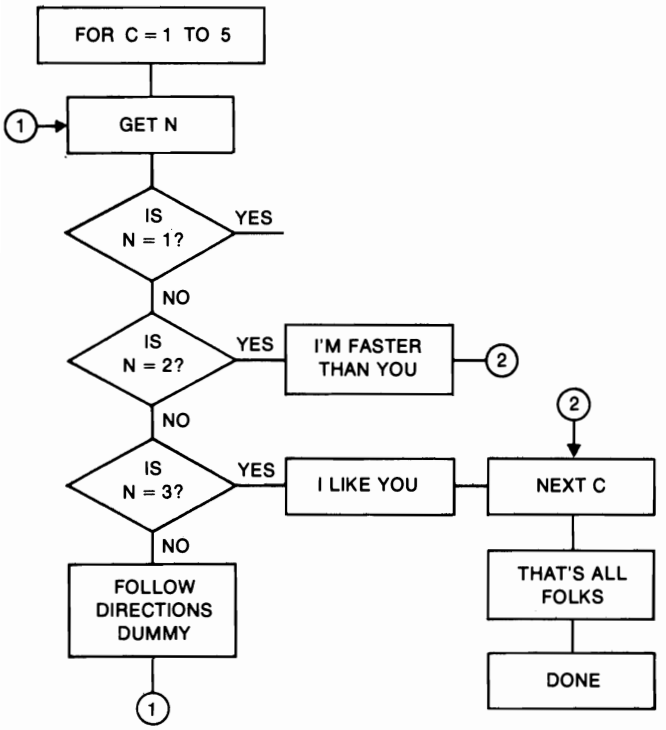


The THIRD YES BRANCH is now complete. Work back through the diamond IS N=3? to the diamond IS N=2? Take the YES exit and draw



Again we're at the end of a branch. The computer has printed another message. Instruct the computer to advance the loop counter by 1. One way to do this is to add another NEXT C box to the chart, but we already have one such box in the THIRD YES BRANCH, and in BASIC you cannot have more than one NEXT statement for each FOR statement. To jump to the NEXT C box in the THIRD YES BRANCH from the SECOND YES BRANCH, attach a tag with the number 2 to the end of the SECOND YES BRANCH and to the NEXT C box in the THIRD YES BRANCH. The flow chart looks like this.

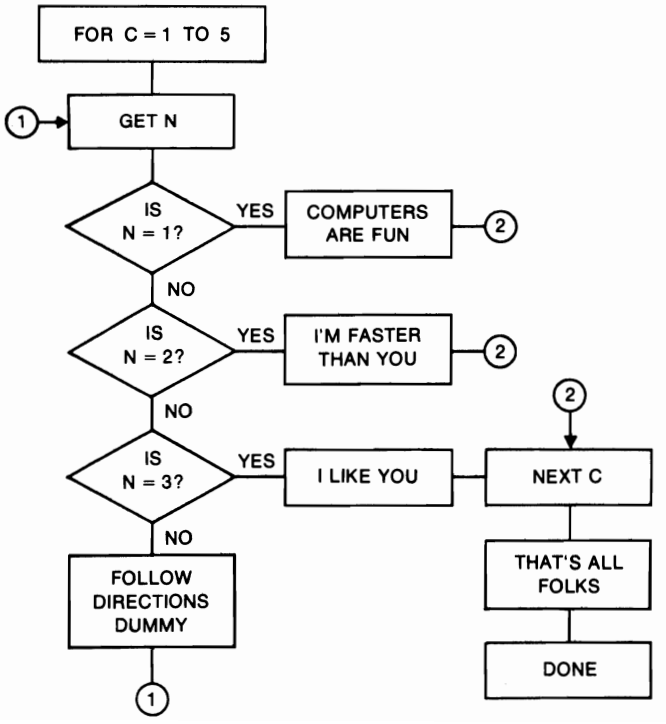
FLOW CHART



Now back up to the first diamond, IS N=1? After taking the right exit, I draw

When I reach the end of the FIRST YES BRANCH, I activate the counter by jumping to the THIRD YES BRANCH. Here is the final flow chart.

3-MESSAGE/LOOP FLOW CHART



Now let's write the program. Put down some line numbers.

```
10
20
30
40
50
60
70
80
90
100
```

The first two boxes are represented in BASIC as:

```
10 FOR C=1 TO 5
20 INPUT N
```

The first diamond is translated as:

```
30 IF N=1 THEN_____
```

Leaving the right exit incomplete, we take the bottom exit and drop down to the second diamond. The second diamond is translated as:

```
40 IF N=2 THEN_____
```

Leaving the right exit incomplete, we take the bottom exit and drop down to the third diamond.

```
50 IF N=3 THEN_____
```

Leaving the right exit incomplete, we take the bottom exit and complete the DROP-DOWN BRANCH.

```
60 PRINT "FOLLOW DIRECTIONS DUMMY."
70 GO TO 20
```

Here is our incomplete program.

```
10 FOR C=1 TO 5
20 INPUT N
30 IF N=1 THEN_____
40 IF N=2 THEN_____
50 IF N=3 THEN_____
60 PRINT "FOLLOW DIRECTIONS
   DUMMY."
70 GO TO 20
```

Only the DROP-DOWN BRANCH is complete. Back up to line 50. Fill in the blank after the THEN with the next available line number, 80. Beginning at line 80, encode the instructions for the THIRD YES BRANCH.

```
 80 PRINT "I LIKE YOU."
 90 NEXT C
100 PRINT "THAT'S ALL FOLKS."
110 STOP
```

Here is our incomplete program.

```
 10 FOR C=1 TO 5
 20 INPUT N
 30 IF N=1 THEN_____
 40 IF N=2 THEN_____
 50 IF N=3 THEN 80
 60 PRINT "FOLLOW DIRECTIONS
    DUMMY."
 70 GO TO 20
 80 PRINT "I LIKE YOU."
 90 NEXT C
100 PRINT "THAT'S ALL FOLKS."
110 STOP
```

Two branches are now complete. Back up to line 40. Insert after THEN the next available line number, 120. Beginning at line 120, encode the steps for the SECOND YES BRANCH.

```
120 PRINT "I'M FASTER THAN YOU."
130 GO TO 90
```

Three branches are now complete. There is only one to go. Finish the FIRST YES BRANCH as follows:

```
140 PRINT "COMPUTERS ARE FUN."
150 GO TO 90
```

Add the END statement:

```
160 END
```

Here is the complete program.

```
 10 FOR C=1 TO 5
 20 INPUT N
 30 IF N=1 THEN 140
 40 IF N=2 THEN 120
 50 IF N=3 THEN 80
 60 PRINT "FOLLOW DIRECTIONS
    DUMMY."
 70 GO TO 20
 80 PRINT "I LIKE YOU."
 90 NEXT C
100 PRINT "THAT'S ALL FOLKS."
110 STOP
120 PRINT "I'M FASTER THAN YOU."
130 GO TO 90
140 PRINT "COMPUTERS ARE FUN."
150 GO TO 90
160 END
```

Two additions will help. First, give the user some instructions by inserting 15 PRINT "TYPE 1, 2, OR 3";. Second, to remind the user who types in some number other than 1, 2, or 3 that he should TYPE 1, 2, OR 3, replace 70 GO TO 20 with 70 GO TO 15. Here is the program with these two changes. A typical run follows.

3-MESSAGE/LOOP ***PROGRAM***

C LOOP

```
10 FOR C=1 TO 5
15 PRINT "TYPE 1,2,OR 3";
20 INPUT N
30 IF N=1 THEN 140
40 IF N=2 THEN 120
50 IF N=3 THEN 80
60 PRINT "FOLLOW DIRECTIONS DUMMY."
70 GO TO 15
80 PRINT "I LIKE YOU."
90 NEXT C
100 PRINT "THAT'S ALL FOLKS."
110 STOP
120 PRINT "I'M FASTER THAN YOU."
130 GO TO 90
140 PRINT "COMPUTERS ARE FUN."
150 GO TO 90
160 END
```

OUTPUT

```
TYPE 1,2,OR 3? 1
COMPUTERS ARE FUN.
TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.
TYPE 1,2,OR 3? 3
I LIKE YOU.
TYPE 1,2,OR 3? -7
FOLLOW DIRECTIONS DUMMY.
TYPE 1,2,OR 3? 2
I'M FASTER THAN YOU.
TYPE 1,2,OR 3? 3
I LIKE YOU.
THAT'S ALL FOLKS.
```

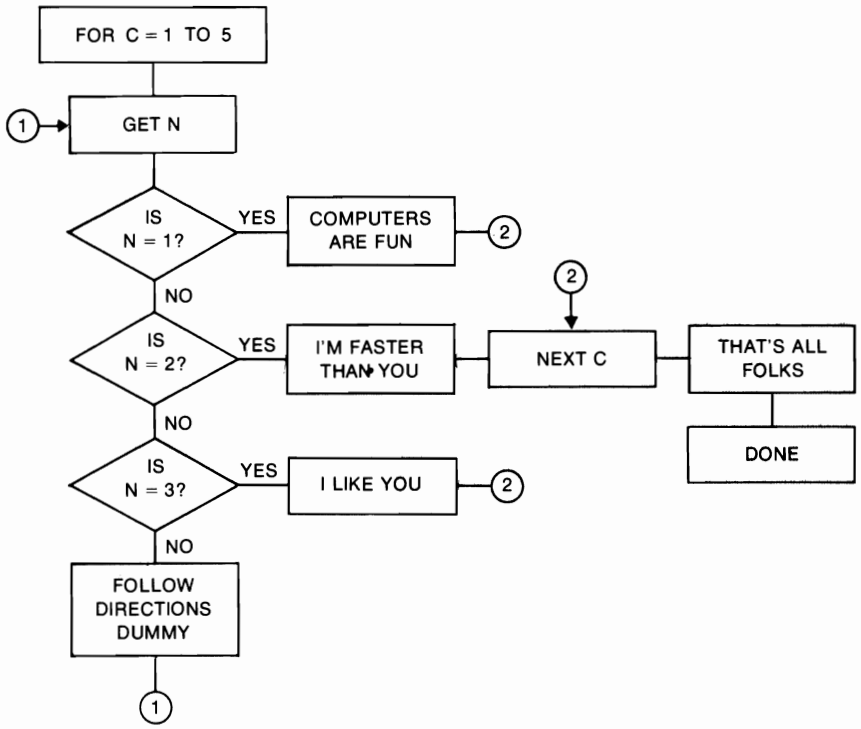
1ST MESSAGE → COMPUTERS ARE FUN.
2ND MESSAGE → I'M FASTER THAN YOU.
3RD MESSAGE → I LIKE YOU.
4TH MESSAGE → I'M FASTER THAN YOU.
5TH MESSAGE → I LIKE YOU.

EXERCISE SET 1.7

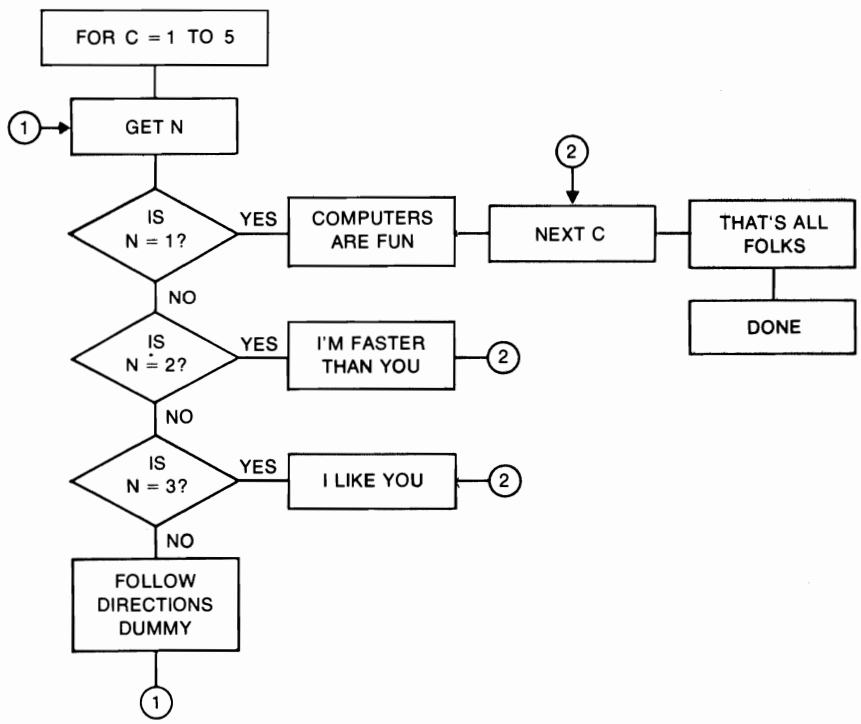
Off-line:

1. The following flow charts are logically equivalent to the flow chart 3-MESSAGE/LOOP. Write a program from each.

A.


FLOW CHART
FOR C = 1 TO 5
1
GET N
IS N = 1?
YES
COMPUTERS ARE FUN
2
NO
IS N = 2?
YES
I'M FASTER THAN YOU
2
NEXT C
THAT'S ALL FOLKS
DONE
NO
IS N = 3?
YES
I LIKE YOU
2
NO
FOLLOW DIRECTIONS DUMMY
1

B.


FLOW CHART
FOR C = 1 TO 5
1
GET N
2
IS N = 1?
YES
COMPUTERS ARE FUN
NEXT C
THAT'S ALL FOLKS
DONE
NO
IS N = 2?
YES
I'M FASTER THAN YOU
2
NO
IS N = 3?
YES
I LIKE YOU
2
NO
FOLLOW DIRECTIONS DUMMY
1

2. It was intended that each of the following programs produce a message in response to a number, as shown here.

INPUT (N)	OUTPUT
1	COMPUTERS ARE FUN
2	I'M FASTER THAN YOU
3	I LIKE YOU

As in 3-MESSAGE/LOOP, the limit on the total number of messages is five. However, not all the programs generate the desired output or produce a total of five messages. Play computer to determine the output and operation of each program. If the output is not as intended, make the necessary corrections.

A.

```
10 FOR C=1 TO 5
15 PRINT
20 PRINT "TYPE 1 2 OR 3";
30 INPUT N
40 IF N=1 THEN 130
50 IF N=2 THEN 110
60 IF N=3 THEN 90
70 PRINT "FOLLOW DIRECTIONS DUMMY."
80 GO TO 20
90 PRINT "I LIKE YOU."
100 NEXT C
110 PRINT "I'M FASTER THAN YOU."
120 NEXT C
130 PRINT "COMPUTERS ARE FUN."
140 NEXT C
150 PRINT "THAT'S ALL FOR NOW."
160 END
```

B.

```
10 PRINT "TYPE 1,2,OR 3";
15 PRINT
20 FOR C=1 TO 5
30 INPUT N
40 IF N=1 THEN 130
50 IF N=2 THEN 110
60 IF N=3 THEN 90
70 PRINT "FOLLOW DIRECTIONS DUMMY."
80 GO TO 20
90 PRINT "I LIKE YOU."
100 GO TO 140
110 PRINT "I'M FASTER THAN YOU."
120 GO TO 140
130 PRINT "COMPUTERS ARE FUN."
140 NEXT C
150 PRINT "THAT'S ALL FOR NOW."
160 END
```

C.

```
10 FOR C=1 TO 5
15 PRINT
20 PRINT "TYPE 1,2,OR 3";
30 INPUT N
40 IF N=1 THEN 130
50 IF N=2 THEN 110
60 IF N=3 THEN 90
70 PRINT "FOLLOW DIRECTIONS DUMMY."
80 GO TO 140
90 PRINT "I LIKE YOU."
100 GO TO 140
110 PRINT "I'M FASTER THAN YOU."
120 GO TO 140
130 PRINT "COMPUTERS ARE FUN."
140 NEXT C
150 PRINT "THAT'S ALL FOR NOW."
160 END
```

D.

```
10 FOR C=1 TO 5
15 PRINT
20 PRINT "TYPE 1,2,OR 3";
30 INPUT N
40 IF N=1 THEN 130
50 IF N=2 THEN 110
60 IF N=3 THEN 90
70 PRINT "FOLLOW DIRECTIONS DUMMY."
80 GO TO 20
90 PRINT "I LIKE YOU."
100 GO TO 20
110 PRINT "I'M FASTER THAN YOU."
120 GO TO 20
130 PRINT "COMPUTERS ARE FUN."
140 NEXT C
150 PRINT "THAT'S ALL FOR NOW."
160 END
```

E.

```
10 FOR C=1 TO 5
15 PRINT
20 PRINT "TYPE 1,2,OR 3";
30 INPUT N
40 IF N=1 THEN 130
50 IF N=2 THEN 110
60 IF N=3 THEN 90
70 PRINT "FOLLOW DIRECTIONS DUMMY."
80 GO TO 20
90 PRINT "I LIKE YOU."
100 GO TO 120
110 PRINT "I'M FASTER THAN YOU."
120 NEXT C
125 STOP
130 PRINT "COMPUTERS ARE FUN."
140 GO TO 120
150 PRINT "THAT'S ALL FOR NOW."
160 END
```

On-line:

3. Think up four quips or messages of your own. Then draw a flow chart and write a program which will (1) ask the user for the total number of messages he would like printed, and (2) type out one of the four messages in response to an input of 1, 2, 3, or 4. After you have established that your program operates correctly, add steps to make it personable and readable. Your program should produce an output similar to that shown at the beginning of this chapter.

```
WELCOME TO THE YOU GUESS GAME. ARE YOU READY TO
MATCH YOUR WITS AGAINST MY BRAINS?

I'LL THINK OF A NUMBER BETWEEN 1 AND 1000 AND
THEN GIVE YOU A CHANCE TO GUESS THE MYSTERY NUMBER.
IF YOUR GUESS IS ON TARGET, I'LL TELL YOU. IF NOT,
I'LL TELL YOU IF YOUR GUESS WAS TOO HIGH OR TOO LOW
AND GIVE YOU ANOTHER CHANCE.

IF YOU'RE A GENIUS YOU SHOULD ALWAYS BE ABLE TO GUESS
THE MYSTERY NUMBER WITH 10 OR FEWER GUESSES.

WHENEVER YOU WISH TO STOP PLAYING TYPE IN 9999.


GUESS THE MYSTERY NUMBER BETWEEN 1 AND 1000? 200
TOO HIGH GUESS AGAIN? 100
TOO HIGH GUESS AGAIN? 90
TOO HIGH GUESS AGAIN? 80
TOO HIGH GUESS AGAIN? 70
TOO HIGH GUESS AGAIN? 60
TOO HIGH GUESS AGAIN? 50
TOO HIGH GUESS AGAIN? 40
TOO HIGH GUESS AGAIN? 30
TOO HIGH GUESS AGAIN? 20
TOO HIGH GUESS AGAIN? 10
TOO HIGH GUESS AGAIN? 5
TOO LOW GUESS AGAIN? 6
CORRECT.
IT TOOK YOU  13  GUESSES.

YOU NEED MORE PRACTICE. IF YOU'RE USING
YOUR HEAD YOU SHOULD NOT NEED MORE THAN
10 GUESSES.


A NEW GAME----HERE WE GO.

GUESS THE MYSTERY NUMBER BETWEEN 1 AND 1000? 9999
THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

2

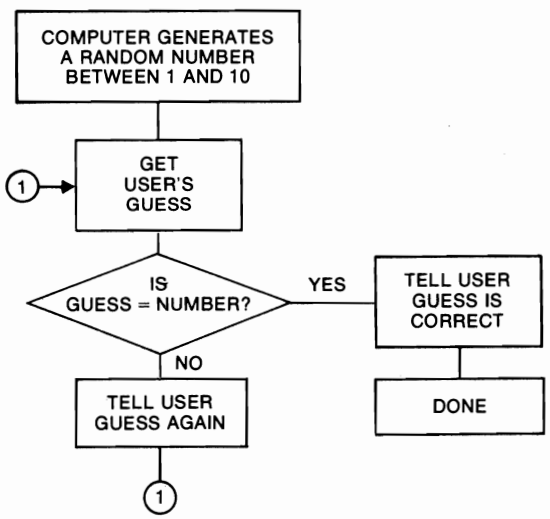
YOU GUESS THE COMPUTER'S NUMBER

2.1 THE GAME: YOU GUESS

In this chapter you will build your first game program. In the game, called YOU GUESS, the computer "thinks" of a mystery number and you try to guess it. First you have to teach the computer to think of a number by using its random number generator. You must also show the computer how to go back and play the game a second time. Then you teach it to count the number of guesses you used to hit upon the number. At the end of the chapter I show you two strategies you can use to reduce the number of guesses needed to hit upon the mystery number.

Draw a flow chart and write a program that instructs the computer to (1) pick a whole number at random between 1 and 10 inclusive and (2) let the user guess the number. If the user's guess is incorrect, give him the opportunity to guess again. If his guess is correct, tell him. Here is the solution or logic for the game:

YOU GUESS/ELEMENTARY FLOW CHART



2.1.1 USING THE RANDOM NUMBER GENERATOR

The first step in the new game is to get the computer to pick a number between 1 and 10 inclusive. How do we get the computer to do that? There is an expression in BASIC, denoted by RND(X), which generates a number at random between 0 and 1, but excluding 0 and 1. RND(X) generates at random *one* number which

belongs to the interval indicated by the thick part of the following number line.

To demonstrate RND(X), here is a short program which lists 20 random numbers between 0 and 1.

RND/20

PROGRAM

```
10 FOR C=1 TO 20
20 PRINT RND(X)
30 NEXT C
40 END
```

OUTPUT

```
.2431684
.2988412
.7295008
.3125257
.3095865
.04493979
.4834217
.4961024
.5010026
.04103271
.2373254
.3046887
.1923863
.9121199
.241212
.9882844
.2587987
.03323189
.871119
.9248194
```

If you prefer to have the numbers typed across the page in five columns instead of being typed one under the other, insert a comma after RND(X) in line 20.

RND/ACROSS

PROGRAM

```
10 FOR C=1 TO 20
20 PRINT RND(X),
30 NEXT C
40 END
```

OUTPUT

```
.2431684    .2988412    .7295008    .3125257    .3095865
.04493979   .4834217    .4961024    .5010026    .04103271
.2373254    .3046887    .1923863    .9121199    .241212
.9882844    .2587987    .03323189   .871119     .9248194
```

A comma at the end of a PRINT statement is a signal to the computer that the next value printed out is to be printed in the next column to the right. The maximum number of columns is five. When a sixth number is to be printed, it appears in the first column on the next line.

Look at the two outputs. All the numbers are decimal numbers between 0 and 1, and 0 and 1 do not appear in the list. All the numbers consist of six or seven non-zero digits to the right of the decimal point. On your machine you might get fewer or more digits.

RND(X) yields only numbers between 0 and 1. To generate random whole numbers between 1 and 10 inclusive we must first multiply RND(X) by 10. This will move the decimal point one place to the right and give us numbers which belong to the heavy part of the number line below.

To demonstrate that this is the case, we need only change line 20 in the last program.

```
10 FOR C=1 TO 20
20 PRINT 10*RND(X),
30 NEXT C
40 END
```

PROGRAM RND/EXPAND

OUTPUT

```
2.431684      2.988412      7.295008      3.125257      3.095365
.4493979      4.834217      4.961024      5.010026      .4103271
2.373254      3.046887      1.923863      9.121199      2.41212
9.882844      2.587987      .3323169      8.71119       9.248194
```

Look at the output. All numbers are now between 0 and 10, exclusive of 0 and 10. Multiplication of RND(X) by 10 has just expanded or stretched the interval from 0 to 1 to 0 to 10.

The list above consists of mixed numbers (such as 2.431684). We want whole numbers (such as 2). A new expression, INT(X), converts mixed numbers into whole numbers (integers). INT(X) finds the largest whole number *less than or equal to* X. To see what INT(X) does, let's write a new program which takes either a whole or a mixed number, X, and types back the greatest whole number.

PROGRAM

```
 5 PRINT "GIVE ME NUMBER";
10 INPUT X
20 PRINT "INT" X "IS" INT(X)
25 PRINT
30 GO TO 5
40 END
```

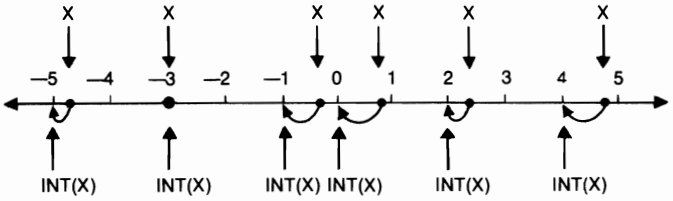
OUTPUT INT

```
GIVE ME NUMBER? 7
INT 7 IS 7

GIVE ME NUMBER? 0
INT 0 IS 0

GIVE ME NUMBER? -12
INT-12 IS-12

GIVE ME NUMBER? 5.78
INT 5.78 IS 5

GIVE ME NUMBER? -1.1
INT-1.1 IS-2

GIVE ME NUMBER? -4.9
INT-4.9 IS-5

GIVE ME NUMBER?
↑C
```

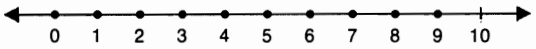
Look at the output. The computer appears to use the following rules to determine INT(X):

1. If X is a whole number (positive, zero, negative), INT(X) produces that whole number.
2. If X is a positive mixed number, INT(X) is obtained by chopping off all digits to the right of the decimal point.
3. If X is a negative mixed number, INT(X) is obtained by chopping off all digits to the right of the decimal point and then subtracting 1.

On a number line, INT(X) produces the largest whole number just to the left of the number X.



Now that we know how INT(X) works, let's return to the task of generating random whole numbers between 1 and 10 inclusive. We have already used the expression 10*RND(X) to create random mixed numbers between 0 and 10. To transform these mixed numbers into whole numbers we use the expression INT(10*RND(X)). This will give us numbers indicated by the solid circles on the following line.



To demonstrate INT(10*RND(X)), we further modify the program RND/EXPAND:

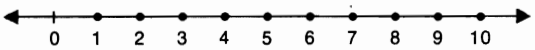
RND/INT ***PROGRAM***

```
10 FOR C=1 TO 20
20 PRINT INT(10*RND(X)),
30 NEXT C
40 END
```

OUTPUT

```
 2     2     7     3     3
 0     4     4     5     0
 2     3     1     9     2
 9     2     0     8     9
```

Look at the output. All the numbers are whole numbers between 0 and 9 inclusive. But we want whole numbers between 1 and 10 inclusive. To get whole numbers between 1 and 10 inclusive, we add 1 to the expression INT(10*RND(X)). The final expression to generate random whole numbers between 1 and 10 inclusive is: INT(10*RND(X))+1. This will produce numbers indicated by the solid circles on the following number line.



To demonstrate INT(10*RND(X))+1, we change line 20 in our program RND/INT.

```
10 FOR C=1 TO 20
20 PRINT INT(10*RND(X))+1,
30 NEXT C
40 END
```

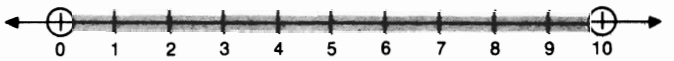
PROGRAM **RND/1 TO 10**

OUTPUT

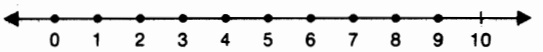
3	3	8	4	4
1	5	5	6	1
3	4	2	10	3
10	3	1	9	10

Here is a summary of the steps we just took to generate random whole numbers between 1 and 10 inclusive using the random number generator, RND(X). When used alone, RND(X) will yield numbers between 0 and 1.

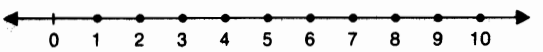
If RND(X) is multiplied by 10, 10*RND(X) will generate mixed numbers between 0 and 10.



If the "greatest integer" expression is used, INT(10*RND(X)) will generate random whole numbers between 0 and 9 inclusive.



Finally, the expression INT (10*RND (X))+1 will generate whole numbers between 1 and 10 inclusive.



A parallel procedure will generate random whole numbers between 1 and 6, 1 and 52, or between 1 and any larger whole number.

Here is a list of the first numbers generated in each demonstration of the five RND(X) programs:

PROGRAM	FIRST NUMBER
RND/20	.2431684
RND/ACROSS	.2431684
RND/EXPAND	2.431684
RND/INT	2
RND/1 TO 10	3

Despite the fact that we ran the computer five different times, the first number in each output is .2431684, or a number developed from .2431684. Why does the computer always start with .2431684? That's hardly a random process.

Although this is not the actual case, you can imagine that the computer contains a list of randomly selected numbers. Each time you run a program which uses the random number generator, the first number picked by the computer for RND(X) is the first number at the top of the list. In the machine I used to generate the examples in this chapter, the first number in the list is .2431684, which becomes 3 when the expression INT(10*RND(X))+1 is used. If the output is to be truly random the computer should start at a randomly selected point on the list. The computer will do this if you place at the beginning of your program the word RANDOMIZE, or just RANDOM. Here is an illustration of the difference.

RND

PROGRAM

```
10 FOR C=1 TO 5
20 PRINT RND(X)
30 NEXT C
40 END
```

OUTPUT

```
RUN

 .2431684
 .2988412
 .7295008
 .3125257
 .3095865

READY

RUN

 .2431684
 .2988412
 .7295008
 .3125257
 .3095865

READY

RUN

 .2431684
 .2988412
 .7295008
 .3125257
 .3095865

READY

RUN

 .2431684
 .2988412
 .7295008
 .3125257
 .3095865

READY
```

RND/RANDOM

PROGRAM

```
 1 RANDOM
10 FOR C=1 TO 5
20 PRINT RND(X)
30 NEXT C
40 END
```

OUTPUT

```
RUN

 .4637543
 .975599
 .5097742
 .653346
 .3320474

READY

RUN

 .8652387
 .1650521
 .07813358
 .3584242
 .4472818

READY

RUN

 .7714887
 .8838021
 .4343836
 .5771742
 .1035318

READY

RUN

 .04297309
 .6982552
 .677743
 .1572523
 .8437661

READY
```

The first program, RND was run four times in succession. You can see that the four outputs are the same. The second program, RND/RANDOM, was also run four times in succession, but the outputs are different due to the use of the command RANDOM in line 1.

EXERCISE SET 2.1.1

Off-line:

1. Write a BASIC expression which will generate a random whole number between and including:
   A. 1 and 2 B. 1 and 6 C. 1. and 52

2. Write a BASIC expression which will generate a random whole number between and including:
   A. 0 and 8 B. 0 and 21 C. 0 and 46

3. Write a BASIC expression which will generate a random whole number between and including:
   A. 5 and 22 B. 10 and 37 C. 7 and 13

4. Play computer and give an example of an output which could be generated by the following program.

```
  1 RANDOM
 10 FOR C=1 TO 5
 20 LET R=INT(3*RND(X))+1
 30 IF R=1 THEN 90
 40 IF R=2 THEN 70
 50 PRINT "I LIKE YOU."
 60 GO TO 100
 70 PRINT "'M FASTER THAN YOU."
 80 GO TO 100
 90 PRINT "COMPUTERS ARE FUN."
100 NEXT C
110 PRINT
120 PRINT "THAT'S ALL FOLKS."
130 END
```

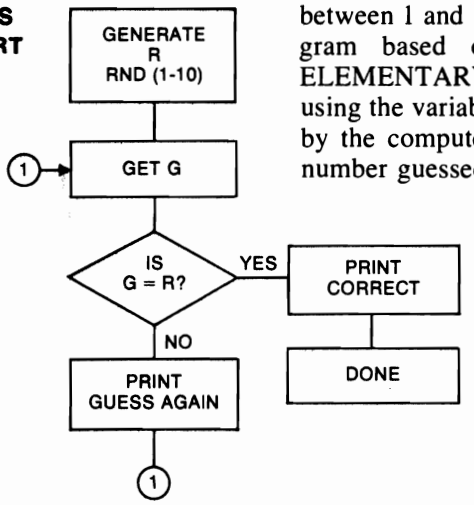
On-line:

5. Draw a flow chart and then write a program which will type out a list of random whole numbers after asking the user for (1) the minimum and maximum random whole number, and for (2) the number of whole numbers in the list. Your program should produce an output like this:

```
I WILL LIST RANDOM WHOLE NUMBERS BETWEEN
A LOWER NUMBER, L, AND AN UPPER NUMBER, U.

GIVE ME LOWER NUMBER? 2
GIVE ME UPPER NUMBER? 17
HOW MANY NUMBERS DO YOU WANT TYPED OUT? 13

 14            17            13             7             9
 10            17             2             6             7
  5             6             3
THAT WAS EASY.
```

2.1.2 PROGRAMMING YOU GUESS

YOU GUESS FLOW CHART



So much for teaching the computer to pick a number between 1 and 10. Let's return to constructing the program based on the flow chart YOU GUESS/ ELEMENTARY. First I'll simplify the flow chart by using the variable R to represent the number generated by the computer and the variable G to represent the number guessed by the user.

Then I write and test a simple program.

PROGRAM

```
 1 RANDOM
20 LET R=INT(10*RND(X))+1
40 INPUT G
60 IF G=R THEN 100
70 PRINT "GUESS AGAIN";
80 GO TO 40
100 PRINT "CORRECT."
120 END
```

OUTPUT

```
? 2
GUESS AGAIN? 7
GUESS AGAIN? 6
GUESS AGAIN? 9
GUESS AGAIN? 4
GUESS AGAIN? 3
CORRECT.
```

Now, assured that the program operates correctly, I'll add some personality. Look at the output of the program as it exists. There are no instructions to the human player. To make the object of the game clear, insert the following text in line 35:

YOU GUESS/INSTRUCT

PROGRAM

```
 1 RANDOM
20 LET R=INT(10*RND(X))+1
35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
40 INPUT G
60 IF G=R THEN 100
70 PRINT "GUESS AGAIN";
80 GO TO 40
100 PRINT "CORRECT."
120 END
```

OUTPUT

```
GUESS THE NUMBER BETWEEN 1 AND 10? 4
GUESS AGAIN? 1
GUESS AGAIN? 8
GUESS AGAIN? 2
GUESS AGAIN? 9
GUESS AGAIN? 3
GUESS AGAIN? 5
CORRECT.
```

You might wish to add some of the following dialogue:

PROGRAM **YOU GUESS/PERSONALITY**

```
SIGN ON
          1 RANDOM
          2 PRINT "THIS PROGRAM IS WRITTEN BY THAT SUPER"
          3 PRINT "PROGRAMMER-----TED SAGE."
          4 PRINT
          5 PRINT "WELCOME TO YOU GUESS. ARE YOU READY TO MATCH"
          6 PRINT "YOUR WITS AGAINST MY BRAINS?"
          7 PRINT
          8 PRINT "I'LL THINK OF A NUMBER BETWEEN 1 AND 10 AND"
          9 PRINT "THEN GIVE YOU A CHANCE TO GUESS THE MYSTERY"
         10 PRINT "NUMBER. IF YOUR GUESS IS ON TARGET I'LL TELL"
         11 PRINT "YOU. IF NOT, I'LL GIVE YOU ANOTHER CHANCE TO"
         12 PRINT "GUESS THE NUMBER."
         13 PRINT
         14 PRINT "GOOD LUCK."
         15 PRINT
         20 LET R=INT(10*RND(X))+1
         35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
         40 INPUT G
         60 IF G=R THEN 100
         70 PRINT "GUESS AGAIN";
         80 GO TO 40
        100 PRINT "CORRECT."
        120 PRINT
SIGN OFF →121 PRINT "YOU'RE A GENIUS. COME BACK SOON."
        130 END
```

OUTPUT

```
THIS PROGRAM IS WRITTEN BY THAT SUPER
PROGRAMMER-----TED SAGE.

WELCOME TO YOU GUESS. ARE YOU READY TO MATCH
YOUR WITS AGAINST MY BRAINS?

I'LL THINK OF A NUMBER BETWEEN 1 AND 10 AND
THEN GIVE YOU A CHANCE TO GUESS THE MYSTERY
NUMBER. IF YOUR GUESS IS ON TARGET I'LL TELL
YOU. IF NOT, I'LL GIVE YOU ANOTHER CHANCE TO
GUESS THE NUMBER.

GOOD LUCK.

GUESS THE NUMBER BETWEEN 1 AND 10? 6
GUESS AGAIN? 1
GUESS AGAIN? 3
GUESS AGAIN? 7
GUESS AGAIN? 10
GUESS AGAIN? 9
GUESS AGAIN? 4
GUESS AGAIN? 2
CORRECT.

YOU'RE A GENIUS. COME BACK SOON.
```

As a final feature, you might like to have the bell on the terminal ring when the user hits upon the number. Look in your manual for the BASIC expression you use to make the bell on your terminal ring. On my particular machine the expression which rings the bell is CHR$(135) following the PRINT command. Nothing is actually printed when the bell is rung. To make the bell ring once after the message "CORRECT" has been typed, I add line 110.

YOU GUESS/1 BELL

PROGRAM

```
  1 RANDOM
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
110 PRINT CHR$(135)  <-- RING BELL
120 END
```

To make the bell ring five times, I write a loop beginning at line 105. Note the use of the semi-colon on line 110. The semi-colon prevents a line feed each time the bell is rung.

YOU GUESS/5 BELL

PROGRAM

```
  1 RANDOM
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 FOR C=1 TO 5          <-- BELL LOOP
110 PRINT CHR$(135);      <-- BELL LOOP
115 NEXT C                <-- BELL LOOP
120 END
```

EXERCISE SET 2.1.2

On-line:

1. Here is a copy of the program YOU GUESS.

```
  1 RANDOM
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
120 END
```

   A. Load and run this program to satisfy yourself that it plays the game as described.
   B. Following some of the suggestions given in the text, add personality to the above program. Save this program for further modification in the next exercise set. For reference purposes I'll name your program YOU/G1.

2. Draw a flow chart and then write a program that
   (1) picks two whole numbers at random between 1 and 10 inclusive,
   (2) asks the user for the sum of these numbers,
   (3) types "correct" if the answer is right, and
   (4) types the correct sum if the answer is wrong.
   Your program should produce an output like this:

```
THIS PROGRAM GIVES YOU PRACTICE AT ADDING
NUMBERS BETWEEN 1 AND 10.

HOW MUCH IS 1  +  8 ? 9
CORRECT.
```

SAMPLE RUN #1

```
THIS PROGRAM GIVES YOU PRACTICE AT ADDING
NUMBERS BETWEEN 1 AND 10.

HOW MUCH IS 10  +  5 ? 14
SORRY--ANSWER IS 15
```

SAMPLE RUN #2

Call this program ADD 1 and save for further modification in the next exercise set.

2.2 "PLAY IT AGAIN, SAM?"

When the user plays YOU GUESS, the game ends when he guesses the number. If he wants to play the game again, he must type the command RUN. This is a nuisance. There are several ways to restart the game automatically.

2.2.1 OPTION 1: AUTOMATIC REPLAY — THE UNCONDITIONAL LOOP

From a programming standpoint, the easiest way to restart the game is to insert in the program a GO TO statement which causes the game to start over automatically after the user finds the mystery number. See line 110 below. If we adopt this method, however, it will be necessary for the user to halt the computer when he wants to *stop* playing. Incidentally, you may want to use a blank line to separate successive games. See line 105 in the following program.

YOU GUESS/AUTO

PROGRAM

```
  1 RANDOM
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 PRINT                    <-- SKIP A LINE
110 GO TO 20                 <-- RESTART GAME
120 END
```

OUTPUT

```
GUESS THE NUMBER BETWEEN 1 AND 10? 3
GUESS AGAIN? 2
GUESS AGAIN? 6
GUESS AGAIN? 7
GUESS AGAIN? 8
GUESS AGAIN? 1
GUESS AGAIN? 4
GUESS AGAIN? 5
GUESS AGAIN? 9
GUESS AGAIN? 10
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10? 5
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10? 2
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10?
↑C
```

2.2.2 OPTION 2: ASK THE USER — THE STRING VARIABLE

Another way to restart the game is to ask the user, after he guesses the number, if he wishes to play again. There are two ways to do this depending on whether or not your version of BASIC allows for the input of the words "yes" or "no." When the user is asked to input a set of characters like "yes" or "no" in response to a question mark, you can instruct the computer to interpret the string of characters as an alphanumeric string. An alphanumeric string is a sequence of numbers and/or characters from the alphabet. For example, YES, NO, ABCDEF, A1B345, 6HO103, FUN99. As you know, if the computer is to interpret the entry as a number, you use a single letter of the alphabet as the variable name. If the computer is to interpret the entry as an alphanumeric string, you again use a letter of the alphabet as a variable name, but follow it by the dollar sign ($). Thus A$, B$, C$, . . ., Z$ denote alphanumeric string variables, or just string variables. The maximum length of the string on my system is six characters, certainly enough to handle words like YES and NO. Consult your manual to determine the maximum length of the character string your system will accept.

If your version of BASIC does not accept alphanumeric strings, you can tell the user that a response of 1 is equivalent to a YES answer and a response of 0 is equivalent to a NO answer. Likewise you can just as easily have 1 mean NO and 2 mean YES. Let's write two programs, one in which 0 means NO and 1 means YES and another in which we use A$ to allow the user to type in the words YES and NO.

To adopt the convention 0 means NO and 1 means YES, we add to the YOU GUESS program lines 110-135 as shown below. In line 125 we cope with the person who refuses to type 0 or 1 in response to the instructions.

PROGRAM **YOU GUESS/0 OR 1**

```
  1 RANDOM
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 PRINT
110 PRINT "WOULD YOU LIKE TO PLAY AGAIN?(0=NO;1=YES)";
115 INPUT A
116 PRINT
120 IF A=1 THEN 20
121 IF A=0 THEN 130
125 PRINT "TYPE 0 OR 1 DUMMY!"
126 GO TO 110
130 PRINT "THANKS FOR PLAYING WITH ME."
135 PRINT "COME BACK SOON."
140 END
```

PLAY AGAIN 0 = NO 1 = YES

SIGN OFF

OUTPUT

```
GUESS THE NUMBER BETWEEN 1 AND 10? 8
GUESS AGAIN? 5
CORRECT.

WOULD YOU LIKE TO PLAY AGAIN?(0=NO;1=YES)? 1

GUESS THE NUMBER BETWEEN 1 AND 10? 2
GUESS AGAIN? 1
GUESS AGAIN? 4
GUESS AGAIN? 7
GUESS AGAIN? 6
GUESS AGAIN? 10
CORRECT.

WOULD YOU LIKE TO PLAY AGAIN?(0=NO;1=YES)? 9

TYPE 0 OR 1 DUMMY!
WOULD YOU LIKE TO PLAY AGAIN?(0=NO;1=YES)? 0

THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

TEST A = 1

TEST A = 9

TEST A = 0

Now let's use the string variable A$. Look at line 115 in the above program, YOU GUESS/0 OR 1. To permit the player to type in the words YES or NO, rather than 1 or 0, the variable A must be replaced by the string variable A$. The new line looks like this:

115 INPUT A$

Now any set of characters entered by the user is given the label A$. To determine if the user in fact typed in the word YES, you write an IF-THEN statement which looks like this:

IF A$ = "YES" THEN_____

The word YES must be in quotes. This tells the computer to compare the string of characters labelled A$ with the string of characters between the quotes, in this case YES.

If the characters entered by the user are not YES, the computer is instructed to test for NO. If the user has not typed in either YES or NO, the computer asks him to follow instructions. As before, these steps are all carried out in lines 110 to 135 in the following program.

YOU GUESS/YES OR NO

PROGRAM

```
  1 RANDOM
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 PRINT
110 PRINT "WOULD YOU LIKE TO PLAY AGAIN?(YES OR NO)";
115 INPUT A$
116 PRINT
120 IF A$="YES" THEN 20
121 IF A$="NO" THEN 130
125 PRINT "ANSWER YES OR NO DUMMY!"
126 GO TO 110
130 PRINT "THANKS FOR PLAYING WITH ME."
135 PRINT "COME BACK SOON."
140 END
```

PLAY AGAIN YES OR NO

← SIGN OFF

OUTPUT

```
GUESS THE NUMBER BETWEEN 1 AND 10? 2
CORRECT.

WOULD YOU LIKE TO PLAY AGAIN?(YES OR NO)? YES

GUESS THE NUMBER BETWEEN 1 AND 10? 8
GUESS AGAIN? 2
GUESS AGAIN? 1
GUESS AGAIN? 5
GUESS AGAIN? 7
GUESS AGAIN? 4
GUESS AGAIN? 9
CORRECT.

WOULD YOU LIKE TO PLAY AGAIN?(YES OR NO)? MAYBE

ANSWER YES OR NO DUMMY!
WOULD YOU LIKE TO PLAY AGAIN?(YES OR NO)? NO

THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

TEST A$ = "YES" →

TEST A$ = "MAYBE" →

TEST A$ = "NO" →

2.2.3 OPTION 3: FLAG 99 — BREAKING OUT OF THE LOOP

Option 3 is a refinement of option 1, the automatic replay. Instead of having the user type CONTROL C to stop the action, we instruct him to type "99" or some other code signal. The choice of 99 is arbitrary. Any number will do as long as it is not a whole number between 1 and 10. Why? The code number signals the computer to follow a particular route. In this case it instructs the computer to jump to the sign-off message. In programming jargon, the code number is known as a flag. Here is the program. Note that the test for the flag (line 50) comes immediately after the input statement (line 40). You want to have the computer look for the flag immediately after input.

YOU GUESS/FLAG 99

PROGRAM

```
  1 RANDOM
 10 PRINT "WHENEVER YOU WISH TO STOP PLAYING"
 11 PRINT "TYPE IN THE NUMBER 99."
 15 PRINT
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 50 IF G=99 THEN 130 <---------- TEST FOR FLAG 99
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 PRINT
110 GO TO 20
130 PRINT "THANKS FOR PLAYING WITH ME."  <-- SIGN OFF
135 PRINT "COME BACK SOON."
140 END
```

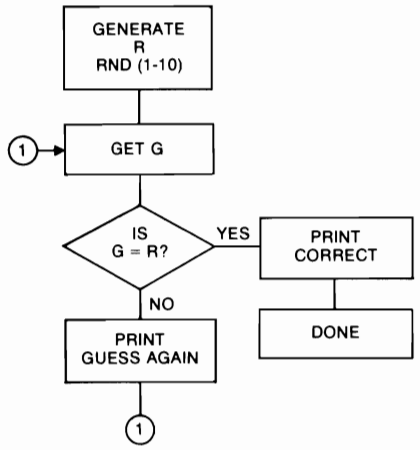
OUTPUT

```
WHENEVER YOU WISH TO STOP PLAYING
TYPE IN THE NUMBER 99.

GUESS THE NUMBER BETWEEN 1 AND 10? 2
GUESS AGAIN? 8
GUESS AGAIN? 1
GUESS AGAIN? 9
GUESS AGAIN? 3
GUESS AGAIN? 4
GUESS AGAIN? 5
GUESS AGAIN? 6
GUESS AGAIN? 7
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10? 8
GUESS AGAIN? 3
GUESS AGAIN? 2
GUESS AGAIN? 5
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10? 99 <-- TEST G = 99
THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

2.2.4 OPTION 4: HOW MANY GAMES? — THE CONDITIONAL LOOP

In this fourth and last option you ask the user to specify at the outset the number of games he would like to play. Here is a copy of the old flow chart YOU GUESS:

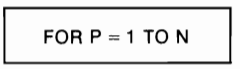
YOU GUESS FLOW CHART



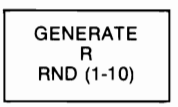
To play the number of games, N, specified by the user, we add three boxes to the flow chart. First, before the game starts, the computer asks the user the number of games he wants to play.

To repeat the game routine automatically N times, we build a loop around the game routine itself with the words FOR and NEXT. The FOR statement counts off the number of times, P, the game is played. When P is equal to N, the loop is satisfied and the computer signs off.

Draw the box

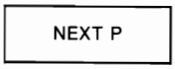


and then the box



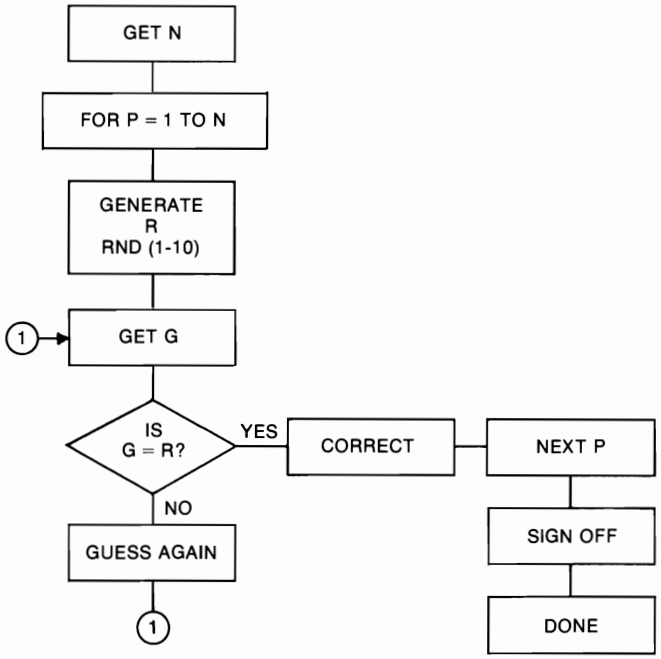
At the end of the game routine, after

add



Here is the revised flow chart for YOU GUESS/HOW MANY.

YOU GUESS/HOW MANY FLOW CHART



From this flow chart the following program is encoded. A typical run follows.

PROGRAM

```
  1 RANDOM
 10 PRINT "HOW MANY GAMES WOULD YOU LIKE TO PLAY";
 11 INPUT N
 15 PRINT
 16 FOR P=1 TO N
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 PRINT
110 NEXT P
130 PRINT "THANKS FOR PLAYING WITH ME."
135 PRINT "COME BACK SOON."
140 END
```

N GAMES LOOP

OUTPUT

```
TEST N = 3 ⟶ HOW MANY GAMES WOULD YOU LIKE TO PLAY? 3

GUESS THE NUMBER BETWEEN 1 AND 10? 8
GUESS AGAIN? 9
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10? 4
GUESS AGAIN? 2
CORRECT.

GUESS THE NUMBER BETWEEN 1 AND 10? 5
GUESS AGAIN? 7
GUESS AGAIN? 3
GUESS AGAIN? 1
GUESS AGAIN? 10
GUESS AGAIN? 9
GUESS AGAIN? 6
GUESS AGAIN? 2
CORRECT.

THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

EXERCISE SET 2.2

Off-line:

1. Draw a flow chart for the program YOU GUESS/0 OR 1. I want to be sure you understand the logic of the replay option used in this program. Omit computer personality from the flow chart.
2. Draw a flow chart for the program YOU GUESS/ FLAG 99.

On-line:

3. Take the program YOU/G1 saved in exercise set 2.1.2 and add the replay option of your choice. Call this new program YOU/G2 and save it for further modification in the next exercise set.
4. Take the program ADD 1 saved in exercise set 2.1.2 and add replay option "99." Call this new program ADD 2 and save it for further modification in the next exercise set.

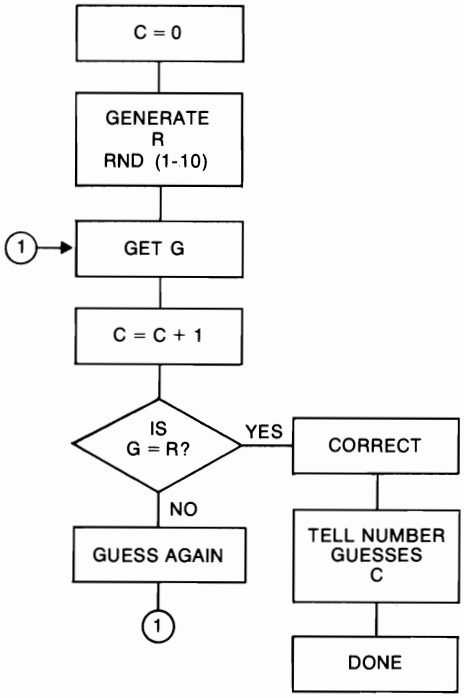
2.3 COUNTING THE NUMBER OF GUESSES

The whole point of a game in which you try to guess the computer's number, or the computer tries to guess your number, is to see how many guesses it takes to hit upon the mystery number. We will now modify the program YOU GUESS to count the number of guesses needed by the user to hit upon the number picked by the computer.

Up to this point, when a counter has been used in a program, it has *controlled* the number of times a particular event occurred. For example, in chapter 1 a counter was used to control the number of messages printed out. In the last section a counter was used in a FOR-NEXT loop to control the number of times the YOU GUESS

game was played. Now we will use a counter, not to *control* but to *count* the number of times the user guesses. Here is the modified flow chart for YOU GUESS/COUNT:

YOU GUESS/COUNT FLOW CHART



In the first box the counter C is set at zero. In the box GET G the user makes a guess. Next comes the box C = C + 1 which increases the counter by 1. When the user finally hits upon the number, the message CORRECT is printed. Then we have the computer tell the user the number of guesses he made by adding to the flow chart the box TELL NUMBER GUESSES, C. Here are the resultant program and a typical output.

PROGRAM

```
  1 RANDOM
 10 LET C=0 <-------------------------- INITIALIZE COUNTER
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 50 LET C=C+1 <------------------------ INCREASE COUNTER
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
110 PRINT "IT TOOK YOU" C "GUESSES." <-- PRINTOUT COUNTER
120 END
```

OUTPUT

```
GUESS THE NUMBER BETWEEN 1 AND 10? 4
GUESS AGAIN? 3
GUESS AGAIN? 1
GUESS AGAIN? 7
GUESS AGAIN? 6
GUESS AGAIN? 5
GUESS AGAIN? 9
GUESS AGAIN? 10
GUESS AGAIN? 8
CORRECT.
IT TOOK YOU 9 GUESSES.
```

EXERCISE SET 2.3

On-line:

1. Take program YOU/G 2 which you saved in exercise set 2.2 and add steps which will tell the user the number of guesses he required to hit upon the number picked by the computer. Call this program YOU/G 3 and save.
2. Take program ADD 2 which you saved in exercise set 2.2 and add steps which tell the user, after he signals he is finished, the number of problems attempted and the number answered correctly. To do this first draw a new flow chart to establish the logic of the program and the correct placement of the two new counters. Save the program as ADD 3. Your program should produce an output like this:

```
THIS PROGRAM GIVES YOU PRACTICE AT ADDING
NUMBERS BETWEEN 1 AND 10.

WHENEVER YOU WISH TO QUIT TYPE 99.

HOW MUCH IS 3  +  4 ? 7
CORRECT.

HOW MUCH IS 8  +  4 ? 12
CORRECT.

HOW MUCH IS 4  +  3 ? 6
SORRY--ANSWER IS 7

HOW MUCH IS 2  +  7 ? 9
CORRECT.

HOW MUCH IS 2  +  8 ? 99
YOU GOT 3 PROBLEMS OUT OF 4 RIGHT.
HOPE YOU HAD FUN.

GOOD LUCK IN ARITHMETIC.
```

2.4 "TOO HIGH TOO LOW": BRANCHING REVISITED

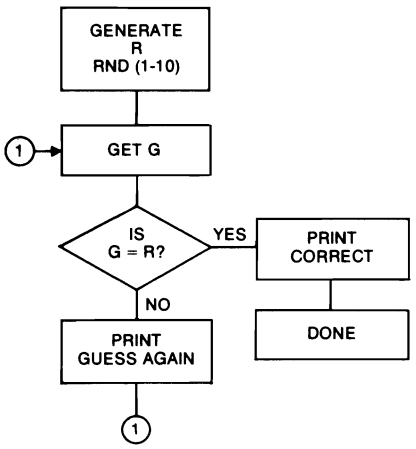
Up to now, when playing YOU GUESS, you had to rely on sheer luck to outwit the computer. When asked

to make your first guess, you selected some number between 1 and 10 and hoped for the best. More often than not you had to guess again. Your second guess was also a blind guess, although we assume you avoided the number you picked the first time. You kept up this process of blind guessing until by luck you hit on the number selected by the computer. Of course if your first nine guesses were wrong, your tenth guess had to be right. You simply guessed the one number between 1 and 10 which you had not yet tried.

We can help the user hit upon the mystery number more quickly if we give him more information about each guess he makes. Instead of just telling him his guess was wrong, we instruct the computer to tell the user whether it was too high or too low. With this feedback the user knows whether to make his next guess higher or lower than his last one.

Let's start with a copy of the YOU GUESS flow chart.

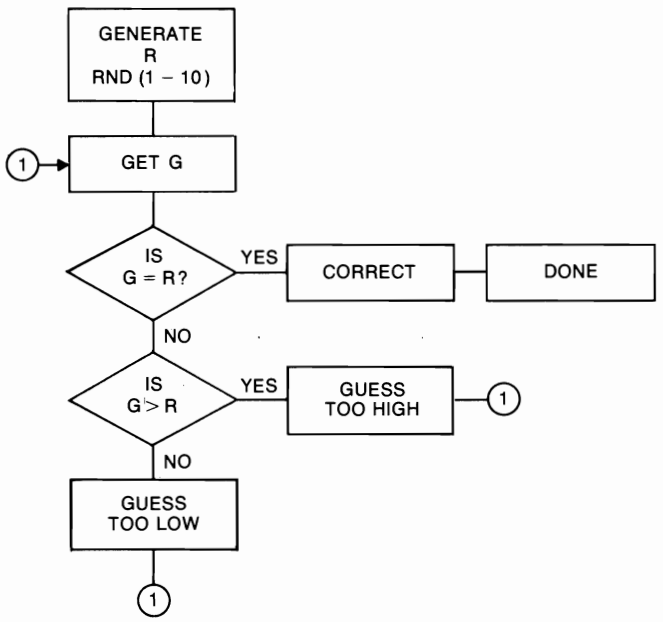
YOU GUESS FLOW CHART



The flow chart already asks the question: IS G = R? If the answer is NO, one or the other of two questions will establish whether the guess is high or low: (1) IS G > R? (2) IS G < R? That makes three questions. But it isn't necessary to include all three questions in the flow chart. Since there are only three possible categories into which the user's guess can fall, if it doesn't fit in two of them, it must belong in the third. One or the other of the two additional questions is inserted in a new diamond at the bottom exit of the old diamond, IS G = R? Does it matter which question is used? No, it doesn't. If question (1) is used, the flow

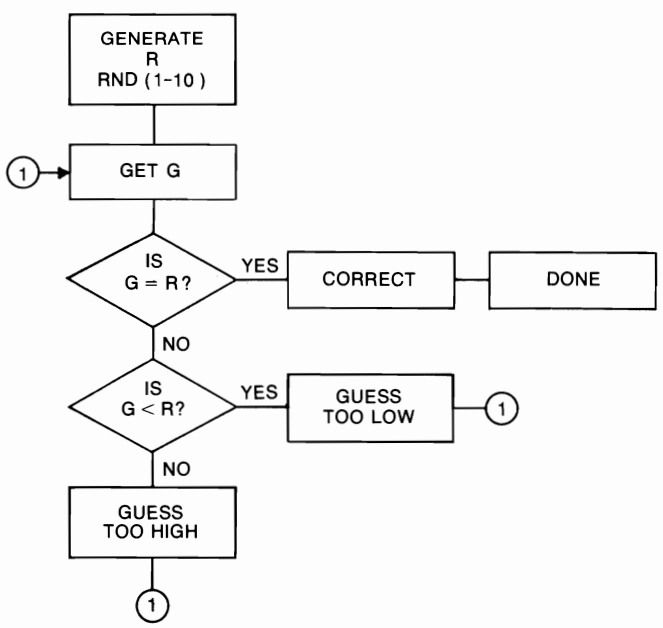
chart looks like this:

YOU GUESS/G > R FLOW CHART



If question (2) is used, the flow chart looks like this:

YOU GUESS/G < R FLOW CHART



Here is a program based on the second flow chart.

PROGRAM

```
 1 RANDOM
20 LET R=INT(10*RND(X))+1
35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
40 INPUT G
60 IF G=R THEN 100
70 IF G<R THEN 90
80 PRINT "TOO HIGH. GUESS AGAIN.";   <-- IF G > R
85 GO TO 40
90 PRINT "TOO LOW. GUESS AGAIN.";    <-- IF G < R
95 GO TO 40
100 PRINT "CORRECT."                 <-- IF G = R
120 END
```

OUTPUT

```
GUESS THE NUMBER BETWEEN 1 AND 10? 5
TOO LOW. GUESS AGAIN.? 7
TOO HIGH. GUESS AGAIN.? 6
CORRECT.
```

EXERCISE SET 2.4

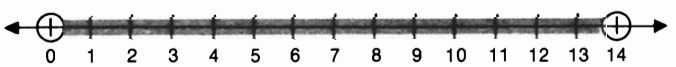
On-line:

1. Take the program YOU/G3 which you saved in exercise set 2.3 and add steps which will inform the user if his guess is too high or too low. Save this new program as YOU/G4 for future modification.

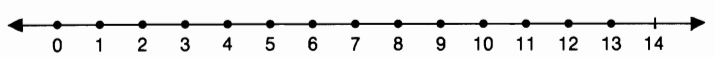
2.5 EXPANDING THE INTERVAL

With the addition of the helpful hints, TOO HIGH, TOO LOW, the number of guesses needed to hit upon the computer's number is reduced. Now it is feasible to have the computer pick its number from a larger set of numbers. In fact, we can let the user decide how difficult he wants to make the game by having him set the limits within which the computer will "think" of a mystery number.

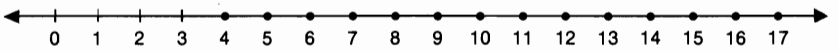
Suppose the user wants to play YOU GUESS between 4 and 17 inclusive. To generate a whole number between 4 and 17 inclusive the computer first calculates the number of whole numbers between 4 and 17 inclusive by subtracting 4 from 17 and adding 1. The result is 17 − 4 + 1, or 14. Next the computer multiplies 14 times RND(X). This expression will generate mixed numbers indicated by the heavy part of the line below.



The next step is to use INT(X) to obtain whole numbers. INT(14*RND(X)) will generate whole numbers indicated by the solid circles.



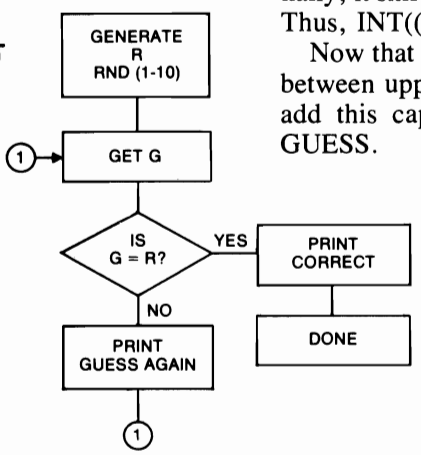
This is not yet the set of 14 numbers you want. To obtain the desired set, the lower limit, 4, must be added to each of the above whole numbers. This produces the expression INT(14*RND(X))+4 which will generate the whole numbers indicated below.



Now let's derive a general expression for generating whole numbers between a lower number, L, and an upper number, U. The computer first calculates the number of whole numbers by using the expression U−L+1. Next it multiplies this expression times RND(X). This gives (U−L+1)*RND(X). The third step is to convert the mixed numbers to whole numbers using INT(X). Thus, INT(U−L+1)*RND(X)). Finally, it shifts the number by adding the lower limit, L. Thus, INT((U−L+1)*RND(X))+L.

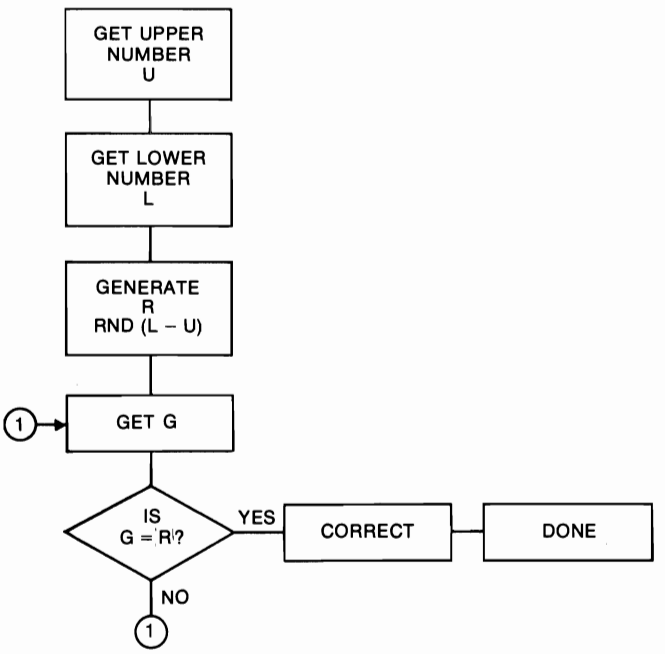
Now that we have a way to generate whole numbers between upper and lower limits set by the user, we'll add this capability to the old flow chart for YOU GUESS.

YOU GUESS FLOW CHART



The new flow chart looks like this:

YOU GUESS/EXPAND FLOW CHART



The first box we added is GET UPPER NUMBER, U. The second box is GET LOWER NUMBER, L. Then we modified the box GENERATE R RND (1-10) to read GENERATE R RND (L – U).

From this flow chart we write the following program:

PROGRAM

```
    1 RANDOM
→  10 PRINT "GIVE ME LOWER AND UPPER NUMBERS";
→  14 INPUT L,U
→  20 LET R=INT((U-L+1)*RND(X))+L
→  35 PRINT "GUESS THE NUMBER BETWEEN" L "AND" U;
   40 INPUT G
   60 IF G=R THEN 100
   70 PRINT "GUESS AGAIN";
   80 GO TO 40
  100 PRINT "CORRECT."
  120 END
```

Look at line 20. It contains the expression we just developed to generate random whole numbers between L and U inclusive. Here is a typical game played under the new rules.

OUTPUT

```
GIVE ME LOWER AND UPPER NUMBERS? 10,16
GUESS THE NUMBER BETWEEN 10 AND 16 ? 12
GUESS AGAIN? 14
GUESS AGAIN? 16
GUESS AGAIN? 15
CORRECT.
```

EXERCISE SET 2.5

On-line:

1. Take your addition program, ADD 3, which you saved in exercise set 2.3 and add steps to allow the user to specify the two numbers between which the computer is to pick its two numbers. Make sure your flag signalling the computer to stop running your program is not a possible correct answer to an addition problem. Your program should produce an

output like this:

```
THIS PROGRAM GIVES YOU PRACTICE AT ADDING
NUMBERS BETWEEN TWO WHOLE NUMBERS.

GIVE ME SMALLER NUMBER? 4
GIVE ME LARGER NUMBER? 17

WHENEVER YOU WISH TO QUIT TYPE 35

HOW MUCH IS 10  +  16 ? 26
CORRECT.

HOW MUCH IS 6  +  12 ? 18
CORRECT.

HOW MUCH IS 6  +  12 ? 17
SORRY--ANSWER IS 18

HOW MUCH IS 6  +  13 ? 19
CORRECT.

HOW MUCH IS 7  +  5 ? 35
YOU GOT 3 PROBLEMS OUT OF 4 RIGHT.
HOPE YOU HAD FUN.

GOOD LUCK IN ARITHMETIC.
```

It won't be necessary to save this program. It will not be used in any future exercises.

2.6 PLAYING THE GAME: DEVELOPING A STRATEGY

By now you've probably discovered that there is a smart way and a dumb way to play YOU GUESS. No doubt you're using the smart way. In any case, in this section I'm going to spell out for you two strategies, a smart one and a not-so-smart one.

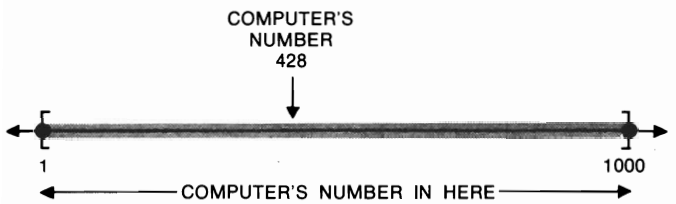
2.6.1 THE RANDOM METHOD

Here is a typical game I played with my computer. The computer is using a program similar to your program YOU/G 4 which you developed in exercise set 2.4. That was the one in which you added the hints TOO HIGH and TOO LOW. The current program differs from YOU/G 4, however, in that I have expanded the interval from 1 to 10, to 1 to 1000 and changed the flag from 99 to 9999.

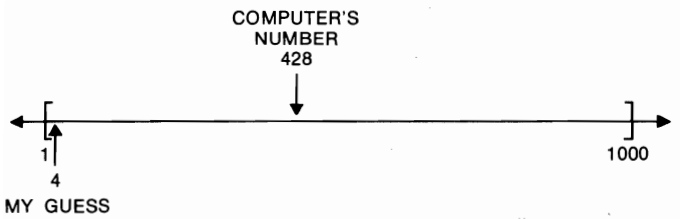
```
WELCOME TO YOU GUESS THE MYSTERY NUMBER.      OUTPUT 1          YOU/G4

WHENEVER YOU WISH TO QUIT TYPE 9999.

GUESS THE NUMBER BETWEEN 1 AND 1000? 4
TOO LOW. GUESS AGAIN? 687
TOO HIGH. GUESS AGAIN? 432
TOO HIGH. GUESS AGAIN? 401
TOO LOW. GUESS AGAIN? 402
TOO LOW. GUESS AGAIN? 421
TOO LOW. GUESS AGAIN? 428
CORRECT.
IT TOOK YOU 7 GUESSES.

GUESS THE NUMBER BETWEEN 1 AND 1000? 9999
THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

Only seven guesses! Can you figure out the method I used to pick the numbers 4, 687, 432, 401, 402, 421, 428?

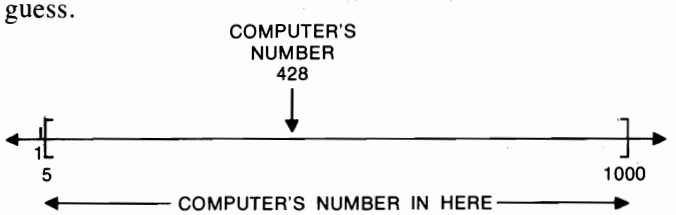
Here's what I did. At the start I knew the computer's number was a whole number in the interval indicated by the heavy line below. The end points are solid circles because the computer could have picked 1 or 1000.



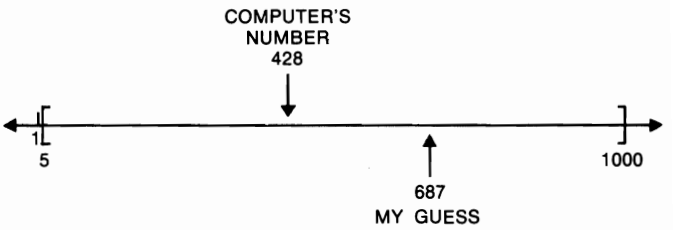
For my first guess I picked a number at random from the interval 1 to 1000. I picked 4.



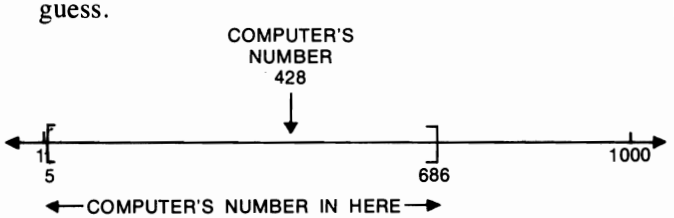
The computer told me this guess was too low. I now knew the computer's number belonged to the following interval. The left end point is 5, one more than the last guess.



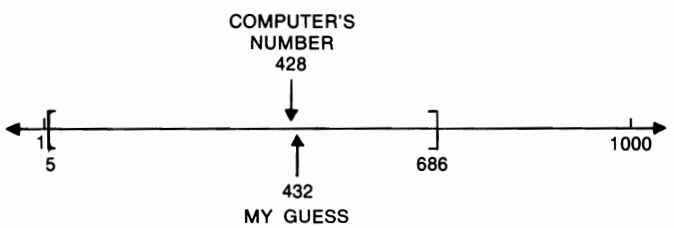
For my second guess I picked at random the number 687 from the interval 5 to 1000.



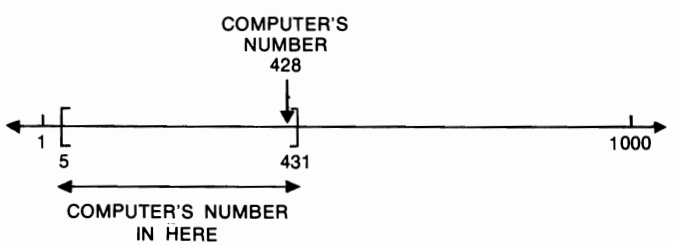
The computer told me this was too high. The computer must be thinking of a number in the following interval. The right end point is 686, or one less than the last guess.



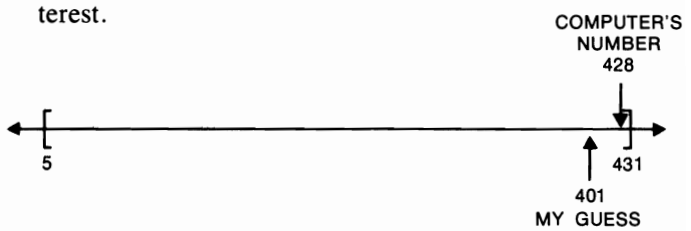
My next guess was 432.



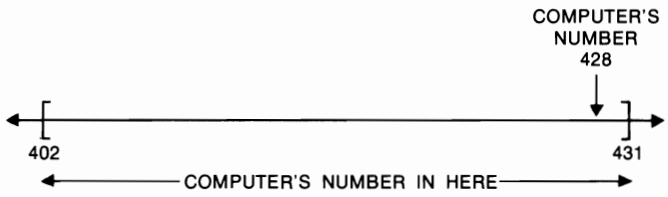
The computer told me this was too high. The computer's number must be in the following interval.



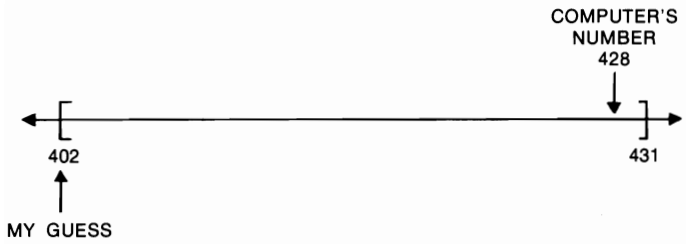
My fourth guess was 401. Let me change the scale of the number line and only show the interval of current interest.



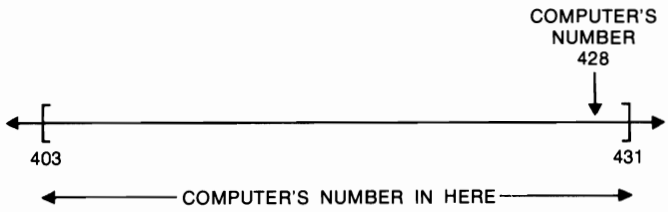
This guess was too low. The computer's number must be between 402 and 431.



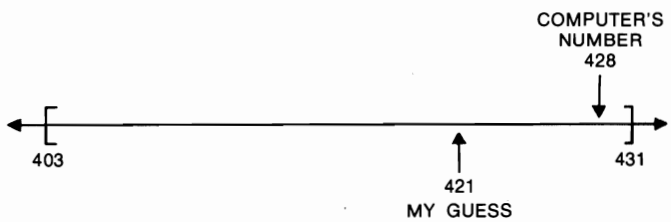
My fifth random guess was 402.



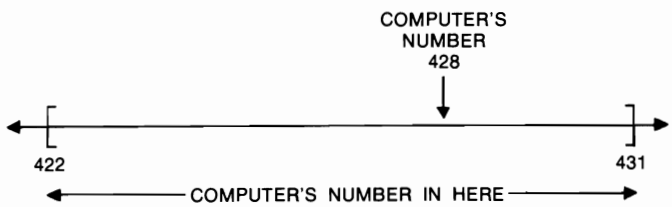
This was also too low. The computer's number must be in the interval 403 to 431.



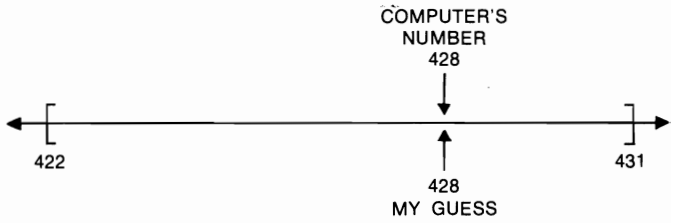
My sixth guess was 421.



This was too low. The computer's number is now in the interval 422 to 431.



My seventh guess was 428.



Right on! I hit upon the computer's number in seven guesses. Not bad considering the scheme I used. Let me call the routine or algorithm just described the Random Method.

2.6.2 THE MIDPOINT METHOD

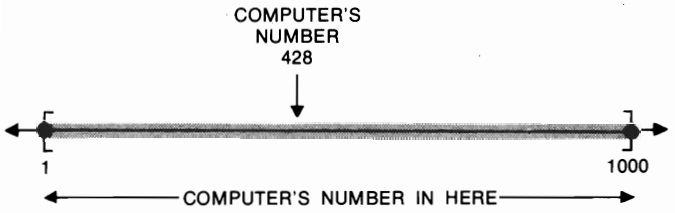
There's another strategy which can be used to guess the computer's number. In fact, I wouldn't be surprised if you already have tried it. Let me show you a game in which I used the new strategy. To make it easy to compare the new method with the Random Method, I asked the computer to "think" of the same number, 428.

YOU/G4 OUTPUT 2

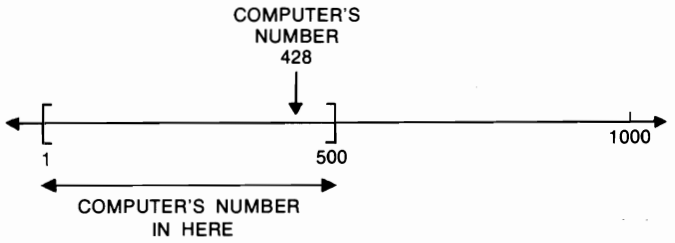
```
WELCOME TO YOU GUESS THE MYSTERY NUMBER.

WHENEVER YOU WISH TO QUIT TYPE 9999.

GUESS THE NUMBER BETWEEN 1 AND 1000? 500
TOO HIGH. GUESS AGAIN? 250
TOO LOW. GUESS AGAIN? 375
TOO LOW. GUESS AGAIN? 437
TOO HIGH. GUESS AGAIN? 406
TOO LOW. GUESS AGAIN? 421
TOO LOW. GUESS AGAIN? 429
TOO HIGH. GUESS AGAIN? 425
TOO LOW. GUESS AGAIN? 427
TOO LOW. GUESS AGAIN? 428
CORRECT.
IT TOOK YOU 10 GUESSES.

GUESS THE NUMBER BETWEEN 1 AND 1000? 9999
THANKS FOR PLAYING WITH ME.
COME BACK SOON.
```

It would appear that the new strategy is not as efficient as the Random Method since it took three more guesses. Let me use number lines to illustrate the new algorithm, which I'll call the Midpoint Method.

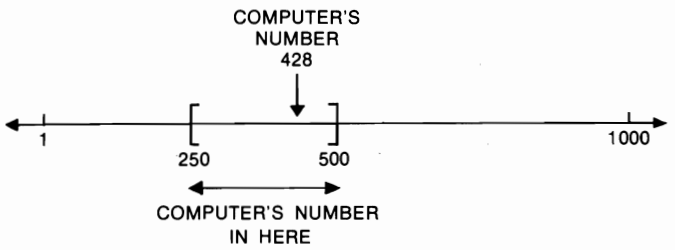
At the start the computer's number is a whole number in the following interval.



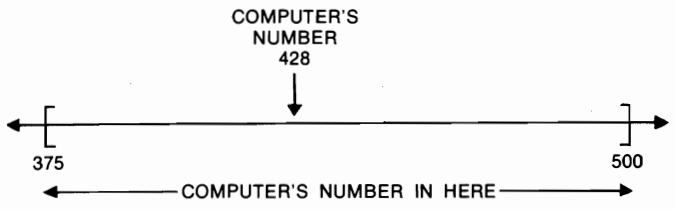
For my first guess I picked 500, the number "midway" between 1 and 1000. (The midpoint number is really 500.5) This was too high, so I knew the computer's number was between 1 and 500.



My second guess was 250. It was too low. The computer's number must be between 250 and 500.



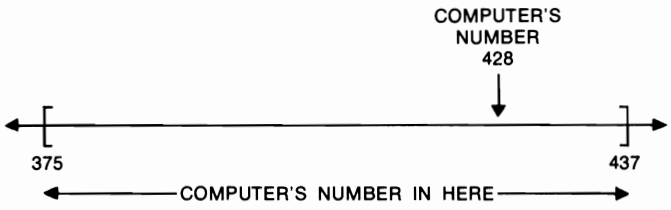
My third guess was 375, halfway between 250 and 500. It was too low. Note the change in scale.



I had to do a little arithmetic to make my fourth guess. I wanted the number halfway between 375 and 500.

$$\frac{375 + 500}{2} = \frac{875}{2} = 437.5$$

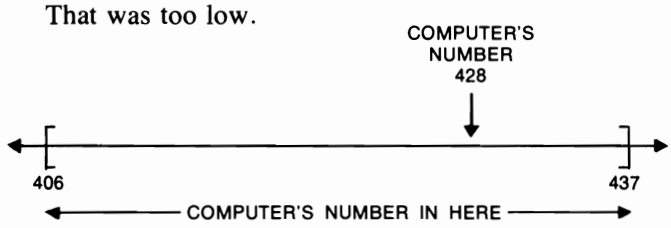
I chopped 437.5 off to 437. That's equivalent to finding INT(437.5). My fourth guess of 437 was too high.



My fifth guess was the average of 375 and 437, or 406.

$$\frac{375 + 437}{2} = \frac{812}{2} = 406.$$

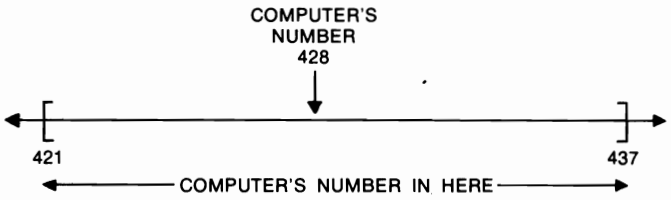
That was too low.



My sixth guess was the average of 406 and 437.

$$\frac{406 + 437}{2} = \frac{843}{2} = 421.5$$

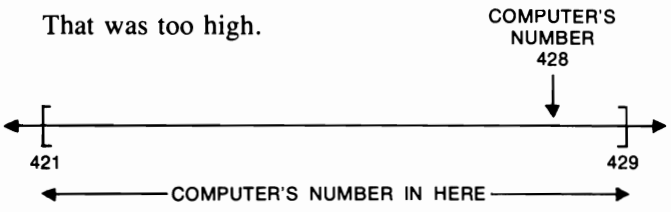
I chopped 421.5 off to 421. That was too low.



The seventh guess was the average of 421 and 437, or 429.

$$\frac{421 + 437}{2} = \frac{858}{2} = 429$$

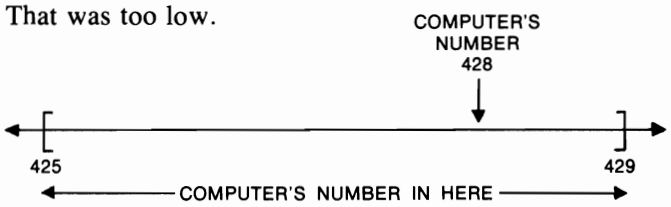
That was too high.



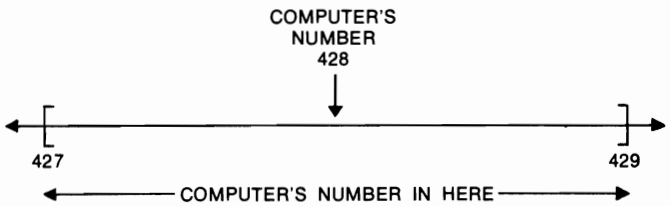
The eighth guess was 425, the average of 421 and 429.

$$\frac{421 + 429}{2} = \frac{850}{2} = 425.$$

That was too low.



The ninth guess was 427, the number halfway between 425 and 429. That was too low.



The tenth guess, 428, halfway between 427 and 429, was correct.

Step by step we trapped the computer's number in the middle of an interval 2 units in length. What is the maximum number of guesses necessary to hit upon the mystery number by this method? I'll give you the answer to this question in the next chapter when I compare the efficiency of the Midpoint and Random Strategies.

EXERCISE SET 2.6

On-line:

1. Take your program YOU/G4, which you saved in exercise set 2.4 and (1) make the interval from which the computer picks its number 1 to 1000, and (2) make the code number to signal the end of play a number outside the interval. After saving this program as YOU/G6, perform the following experiments.
   - A. Run your program YOU/G6 and play enough games using the Random Method to answer the following questions. Determine
     (1) the least number of guesses,
     (2) the most number of guesses, and
     (3) the average number of guesses needed to hit upon the mystery number.
   - B. Run your program YOU/G6 and play enough games using the Midpoint Method to answer the following questions. Determine
     (1) the least number of guesses,
     (2) the most number of guesses, and
     (3) the average number of guesses needed to hit upon the mystery number.

```
*****WELCOME TO THE PERFECT GUESSING MACHINE*****

THINK OF A WHOLE NUMBER BETWEEN 1 AND 1000
AND KEEP IT IN YOUR HEAD.

I WILL TRY TO GUESS IT WITHIN 10 TRIES.

YOU MUST TELL ME IF MY GUESS IS HIGH, LOW, OR
RIGHT. TYPE H (FOR HIGH), L (FOR LOW), R (FOR RIGHT).

WHENEVER YOU WISH TO STOP PLAYING TYPE IN
THE WORD DONE.

AWAY WE GO.


THINK OF A NUMBER BETWEEN 1 AND 1000.
I GUESS 500 ? H
I GUESS 250 ? L
I GUESS 375 ? L
I GUESS 437 ? H
I GUESS 406 ? H
I GUESS 390 ? L
I GUESS 398 ? L
I GUESS 402 ? H
I GUESS 400 ? R
I'M SMART!
I TOOK 9 GUESSES.


THINK OF A NUMBER BETWEEN 1 AND 1000.
I GUESS 500 ? L
I GUESS 750 ? H
I GUESS 625 ? H
I GUESS 562 ? L
I GUESS 593 ? H
I GUESS 577 ? H
I GUESS 569 ? H
I GUESS 565 ? H
I GUESS 563 ? H
I GUESS 562 ? H
YOU CHEATED. JUST FOR THAT I QUIT.
```

3

THE COMPUTER GUESSES YOUR NUMBER

In the last chapter we programmed the computer to play the game YOU GUESS. You learned how to use RND (X) and INT(X) to generate numbers at random. Furthermore, you saw a short program enlarged to include a replay option, a counter for the number of guesses, and the helpful hints of too high and too low. You then expanded the interval within which the computer picked the mystery number and learned two strategies you could use to find it. In this chapter we turn the tables; you pick the mystery number and the computer tries to find it. You will learn to use a subscripted variable to set up "check lists" and "tally lists." In the last section you will compare the efficiency of the Random and Midpoint methods by using the speed of the computer to play one thousand COMPUTER GUESS games by each method.

3.1 TURNING THE TABLES: COMPUTER GUESS

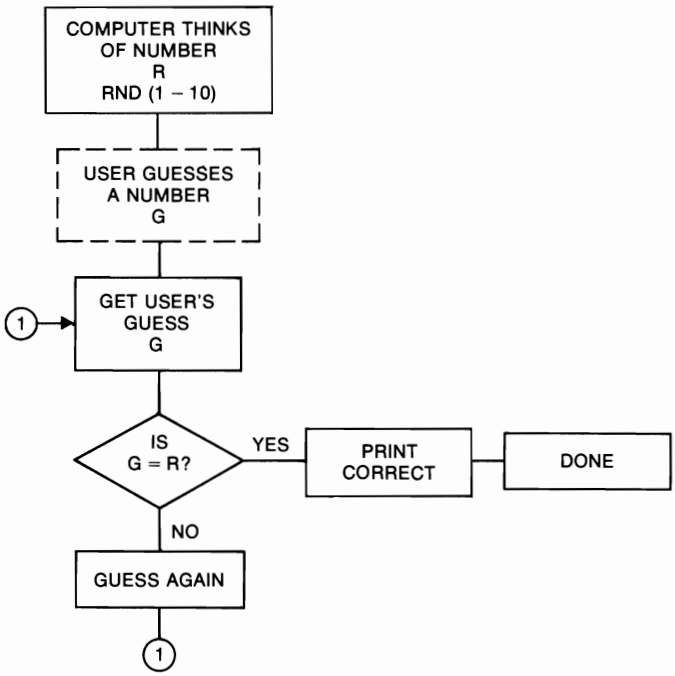
We will develop three programs, each of which guides the computer in finding the mystery number. In following the first program, however, the computer, unlike the human, is unable to remember any of the numbers it has already tried. The second and third programs remedy that defect. The computer does not guess the same number twice and behaves more like a human. The two "memory" programs differ only in length. The first one is based on your current knowledge of BASIC; the second takes advantage of a new capability, the subscripted variable, to accomplish the same results in only one-quarter of the steps.

3.1.1 REPEATING WRONG GUESSES

The first new COMPUTER GUESS program will be similar to YOU GUESS, the program developed in Chapter 2. To see the similarities and differences between these two games, look at the elementary flow

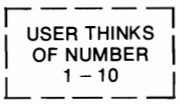
chart for YOU GUESS:

YOU GUESS FLOW CHART

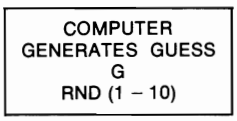


In this flow chart of YOU GUESS the steps taken by you, the human player, are printed in boxes with dashed lines. As before, the boxes with solid lines contain the steps taken by the computer.

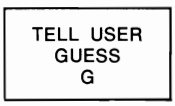
In the first step of COMPUTER GUESS you, the user, think of a mystery number between 1 and 10. Since this is your move, we'll enclose it in dashed lines.



Now the computer takes its first step.

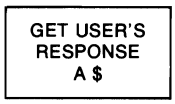


Having generated a guess, the computer must next print the number so you can read it.

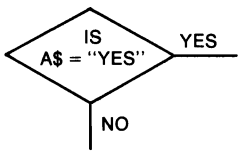


Next you, the user, tell the computer if it has hit upon the mystery number. If the computer is correct, you type

in the word YES. If it is incorrect you type in NO. To allow for the input of the words YES and NO, I'll use the string variable A$.

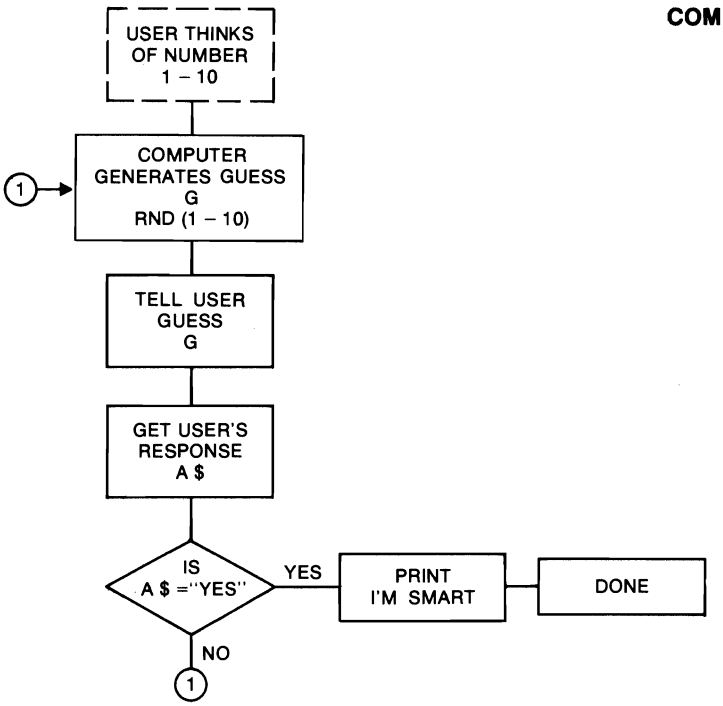


After you type in YES or NO, the computer compares your input with the word YES.



If your answer is YES, the computer has correctly guessed the mystery number and the game is over. If your answer is NO, the computer missed the target and must guess again. Here's the complete flow chart for the new game COMPUTER GUESS/ELEMENTARY.

COMPUTER GUESS/ELEMENTARY FLOW CHART



I have written the following program based on the above flow chart. Line 10 prompts you, the user, to take the action specified in the box with the dashed lines.

PROGRAM

```
 1 RANDOM
10 PRINT "THINK OF A NUMBER FROM 1 TO 10."
20 LET G=INT(10*RND(X))+1
40 PRINT "I GUESS" G;
60 INPUT A$
80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
140 END
```

Here is a typical game played using my program. The mystery number is 6.

OUTPUT

```
THINK OF A NUMBER FROM 1 TO 10.
I GUESS 5 ? NO
I GUESS 8 ? NO
I GUESS 10 ? NO
I GUESS 10 ? NO
I GUESS 2 ? NO
I GUESS 7 ? NO
I GUESS 5 ? NO
I GUESS 3 ? NO
I GUESS 2 ? NO
I GUESS 9 ? NO
I GUESS 7 ? NO
I GUESS 8 ? NO
I GUESS 3 ? NO
I GUESS 4 ? NO
I GUESS 7 ? NO
I GUESS 1 ? NO
I GUESS 8 ? NO
I GUESS 7 ? NO
I GUESS 8 ? NO
I GUESS 5 ? NO
I GUESS 4 ? NO
I GUESS 9 ? NO
I GUESS 8 ? NO
I GUESS 4 ? NO
I GUESS 10 ? NO
I GUESS 10 ? NO
I GUESS 2 ? NO
I GUESS 6 ? YES
I'M SMART.
```

Although the computer says, "I'M SMART," I'm not so sure. The computer required 28 guesses before it hit upon the number 6 in the narrow range from 1 to 10. Why did it take the computer so long to guess the number 6 when it followed this particular program?

EXERCISE SET 3.1.1

On-line:

1. Add the following features to the program COMPUTER GUESS/ELEMENTARY. Then load and run your new program.
   A. Steps which give the user more instructions on how to play the game.

B. Steps which cause the bell to ring five times after I'M SMART is printed. Be sure to suppress the line feed after the ringing of each bell.
C. Steps to handle words other than YES or NO in response to the computer's guess. For example, the user might type in MAYBE. (If you're stuck, refer to the program YOU GUESS/YES OR NO in section 2.2.2.)

3.1.2 PREVENTING DUPLICATE GUESSES

In the typical run of COMPUTER GUESS shown in the last section, the computer often tried the same number, for example 8, several times. Can we invent a scheme whereby the computer keeps track of its previous guesses and does not guess the same number twice? When you played YOU GUESS in the last chapter, you probably kept track of previous guesses by using a mental check list. Or you looked at the printout to make sure you didn't duplicate any of your previous guesses. In this section we'll program the computer to keep a check list. After it picks a number at random, it must consult the list to see if it has already tried that number. If it has not, it will ask the user if that number is the mystery number, at the same time putting a "check mark" next to the number in the list to indicate it has been tried.

If I were to use a check list, it would look like this:

Number	Check When Tried
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

If I pick 3 as my guess, I look at the list, see no check mark in the "Check When Tried" column, and thus know the number has not been previously tried. After several tries, the list might look like this.

Number	Check When Tried
1	√
2	
3	√
4	√
5	
6	√
7	
8	
9	
10	√

Now I pick 6 as my next guess. I look at the list, see the check mark in the second column, and know that 6 has been tried before. I must keep on until I get a number which is not 1, 3, 4, 6, or 10. I keep generating numbers randomly and eventually select 5. I put a check mark in the second column next to 5 before proceeding.

The guessing routine followed by a human player is clear. Let's teach it to the computer. First, we assign a unique variable name to each of the numbers from 1 to 10. For example, let N1 represent the number 1, N2 represent the number 2, etc. Second, instead of a check mark, we'll use the number one (1) to indicate a number has been tried. We'll use the number zero (0) if it has not been tried. The numbers 1 and 0 are used as flags. Remember you used 99 as a flag to terminate play of YOU GUESS in Chapter 2.

Here is the computer's check list at the start of the game COMPUTER GUESS:

Variable	Check When Tried
N1	Ø
N2	Ø
N3	Ø
N4	Ø
N5	Ø
N6	Ø
N7	Ø
N8	Ø
N9	Ø
NØ	Ø

You just learned something new. Numerical variable names in BASIC may be denoted not only by a single letter of the alphabet but also by a single letter of the alphabet followed by a digit Ø–9. Thus, A6, Z9, Q8, B3 are valid variable names in BASIC. Notice that we used NØ to represent 10.

After several guesses the computer's check list looks like this:

Variable	Check When Tried
N1	1
N2	Ø
N3	1
N4	1
N5	Ø
N6	1
N7	Ø
N8	Ø
N9	Ø
NØ	1

The above list is represented in the computer this way:

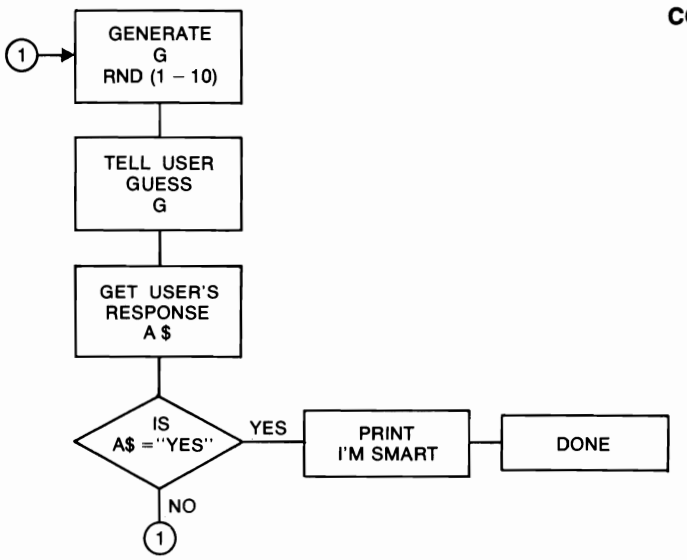
LABEL	N1	N2	N3	N4	N5	N6	N7	N8	N9	NØ
CONTENTS	1	0	1	1	0	1	0	0	0	1

The computer's next guess is 6. It looks at the box labeled N6 and sees the number 1, the signal that 6 has already been tried. The computer continues picking numbers at random until it gets a number which is not 1, 3, 4, 6, or 10. The number is 5. It puts a 1 in the box labeled N5 and asks the user if 5 is the mystery number.

Now that we have a way to prevent duplicate guessing, we shall write two new programs to play the game COMPUTER GUESS. Both programs are modifications of our old one. The first new program will be long but easy to follow. Most programs develop this way. The first version is long and clumsy, and is is not until the programmer has "lived" with it a while that he discovers a more elegant form.

To write the new programs, let us look again at the flow chart for the elementary version of COMPUTER GUESS. The dashed box, USER THINKS OF NUMBER 1-10, has been omitted.

COMPUTER GUESS/ELEMENTARY FLOW CHART

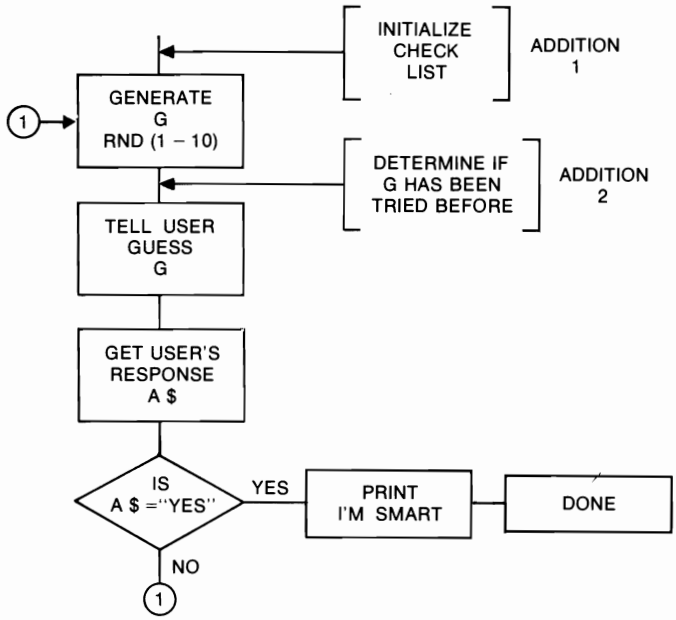


Two additions to this flow chart are required to implement the check list.

1. Set up the check list. At the top of the flow chart the variables N1, N2, N3, . . ., NØ must be defined and given initial values of zero. This is called "initialization of the check list."

2. After the computer generates a number, it must determine if the number has been previously tried.

COMPUTER GUESS/CHECK



The additions to the flow chart are encoded in BASIC as follows:

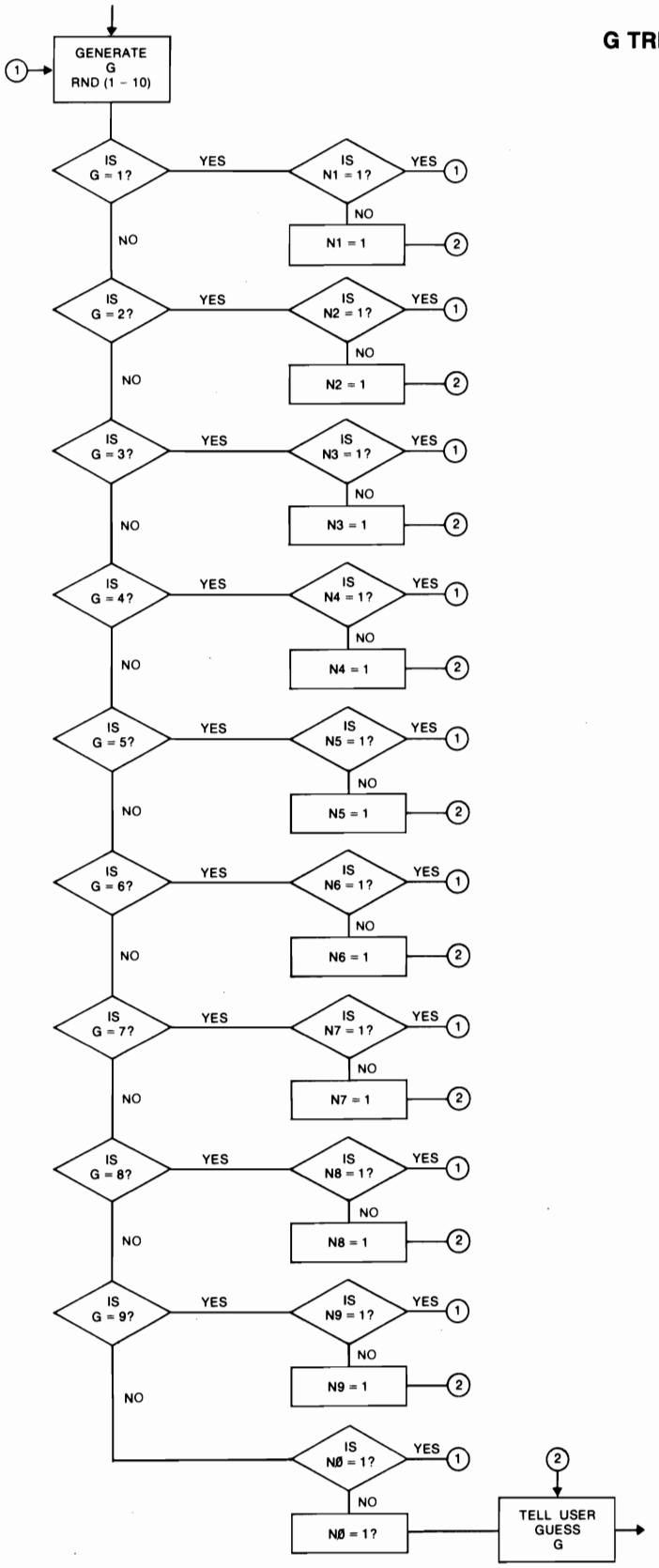
1. The first addition, INITIALIZE CHECK LIST, can be programmed as:

```
10 LET N1=0
11 LET N2=0
12 LET N3=0
13 LET N4=0
14 LET N5=0
15 LET N6=0
16 LET N7=0
17 LET N8=0
18 LET N8=0
19 LET N0=0
```

I have used line numbers 10 to 19 so these new lines will fit into my old program.

2. The second addition, DETERMINE IF G HAS BEEN TRIED BEFORE, is more complicated and it is wise to draw a flow chart first. The procedure is as follows. Each time the computer generates a random number, G, it must determine what number it generated: 1, 2, 3, . . ., or 10. Then it must examine the contents of the box corresponding to that number. If there is a 1 in the box, the number has already been tried and the computer must go back and pick another number. If it is Ø, the computer inserts a 1 in the box and asks the user if the number generated is the mystery number. When the interval is 1 to 10, nineteen diamonds are required in the flow chart, nine to determine the value of G and ten to check the contents of the boxes.

G TRIED BEFORE/19 DIAMONDS FLOW CHART



I shall now encode the foregoing flow chart in BASIC and add it to my old program. Since the new routine will not fit between lines 20 and 40 of my old program I will add line 30, GO TO 200, and at line 200 I'll begin the new routine.

```
200 IF G=1 THEN 310
201 IF G=2 THEN 320
202 IF G=3 THEN 330
203 IF G=4 THEN 340
204 IF G=5 THEN 350
205 IF G=6 THEN 360
206 IF G=7 THEN 370
207 IF G=8 THEN 380
208 IF G=9 THEN 390
300 IF N0=1 THEN 20
301 LET N0=1
302 GO TO 40
310 IF N1=1 THEN 20
311 LET N1=1
312 GO TO 40
320 IF N2=1 THEN 20
321 LET N2=1
322 GO TO 40
330 IF N3=1 THEN 20
331 LET N3=1
332 GO TO 40
340 IF N4=1 THEN 20
341 LET N4=1
342 GO TO 40
350 IF N5=1 THEN 20
351 LET N5=1
352 GO TO 40
360 IF N6=1 THEN 20
361 LET N6=1
362 GO TO 40
370 IF N7=1 THEN 20
371 LET N7=1
372 GO TO 40
380 IF N8=1 THEN 20
381 LET N8=1
382 GO TO 40
390 IF N9=1 THEN 20
391 LET N9=1
392 GO TO 40
```

The instruction to inspect the box labeled N1 is put on line 310. The instruction to inspect box N2 is placed on line 320, etc. That is, I made the 10's digit in the line number equal the digit in the variable name. On which line did I put the instruction to inspect box N9? This mnemonic will help you locate these operations in the program.

Now we're going to insert the routines to INITIALIZE CHECK LIST and DETERMINE IF G HAS BEEN TRIED BEFORE in our old version of COMPUTER GUESS. The END statement must be shifted to line 400 and a STOP statement must be inserted after I'M SMART.

COMPUTER GUESS/19 QUESTIONS

PROGRAM

```
  1 RANDOM
  9 PRINT "THINK OF A NUMBER FROM 1 TO 10."
 10 LET N1=0
 11 LET N2=0
 12 LET N3=0
 13 LET N4=0
 14 LET N5=0
 15 LET N6=0
 16 LET N7=0
 17 LET N8=0
 18 LET N9=0
 19 LET N0=0
 20 LET G=INT(10*RND(X))+1
 30 GO TO 200
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
140 STOP
200 IF G=1 THEN 310
201 IF G=2 THEN 320
202 IF G=3 THEN 330
203 IF G=4 THEN 340
204 IF G=5 THEN 350
205 IF G=6 THEN 360
206 IF G=7 THEN 370
207 IF G=8 THEN 380
208 IF G=9 THEN 390
300 IF N0=1 THEN 20
301 LET N0=1
302 GO TO 40
310 IF N1=1 THEN 20
311 LET N1=1
312 GO TO 40
320 IF N2=1 THEN 20
321 LET N2=1
322 GO TO 40
330 IF N3=1 THEN 20
331 LET N3=1
332 GO TO 40
340 IF N4=1 THEN 20
341 LET N4=1
342 GO TO 40
350 IF N5=1 THEN 20
351 LET N5=1
352 GO TO 40
360 IF N6=1 THEN 20
361 LET N6=1
362 GO TO 40
370 IF N7=1 THEN 20
371 LET N7=1
372 GO TO 40
380 IF N8=1 THEN 20
381 LET N8=1
382 GO TO 40
390 IF N9=1 THEN 20
391 LET N9=1
392 GO TO 40
400 END
```

INITIALIZE CHECK LIST

G TRIED BEFORE

Here is a typical game played using the new program. The mystery number was 6.

```
OUTPUT  THINK OF A NUMBER FROM 1 TO 10.
        I GUESS 7 ? NO
        I GUESS 5 ? NO
        I GUESS 10 ? NO
        I GUESS 4 ? NO
        I GUESS 1 ? NO
        I GUESS 6 ? YES
        I'M SMART.
```

Compare the above output with the output of the original version of COMPUTER GUESS shown earlier in this section. Following the old program, the computer took twenty-eight guesses; following the new program it took only six.

EXERCISE SET 3.1.2

Off-line:

1. Which of the following symbols are valid names for numerical variables? That is, which set of symbols may be used to hold a place for a number in BASIC?

A) T	D) M$	G) Q
B) DD	E) F$2	H) 6T
C) Z6	F) AB6	I) J2

2. Here is the program for a COMPUTER GUESS game in which the human thinks of a number between 1 and 3. Add the check list routine to prevent the computer from trying the same number twice.

```
  1 RANDOM
 10 PRINT "THINK OF A NUMBER FROM 1 TO 3."
 20 LET R=INT(3*RND(X))+1
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
140 END
```

3. Now let's turn the tables back again. Here is a program for a YOU GUESS game in which the computer thinks of a mystery number between 1 and 5. When the user tries to guess the mystery number, he might forget his previous guesses and try the same number a second time. Insert in the program the check list routine developed for COMPUTER GUESS to enable the computer to determine whether the user has tried that number before and, if so, to respond DUMMY! YOU'VE ALREADY GUESSED THAT NUMBER.

```
  1 RANDOM
 10 PRINT "I'M THINKING OF A MYSTERY NUMBER BETWEEN 1 AND 5."
 20 LET R=INT(5*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 5";
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
120 END
```

3.1.3 PREVENTING DUPLICATE GUESSES: THE SHORT WAY

Here is a technique that will dramatically reduce the length of the routine to prevent duplicate guesses. To this point numerical variables have been represented by a letter of the alphabet, or by a letter of the alphabet followed by a digit 0-9. Now we add a third method in which numerical variables are represented by a subscript. Subscripted variables look like A_1, A_2, A_3, A_4, . . . or B_1, B_2, B_3, B_4, . . . or like C_1, C_2, C_3, C_4 . . . , etc. Subscripted variables are written by using a letter of the alphabet and then attaching as a subscript the positive whole numbers 1, 2, 3, 4 and so on. The value of the subscript is only limited by the size of your computer. Check your computer manual for the maximum and minimum value of a subscript. In some versions of BASIC the value of a subscript may be zero.

Most terminals will not accommodate symbols that look like A_1, A_2, A_3, ... So, when programming in BASIC, we indicate subscripted variables by writing A(1), A(2), A(3), ... The computer interprets these symbols, which we read "A sub 1," "A sub 2," "A sub 3," etc. as symbols for subscripted variables. Be sure not to confuse A(1) with A1 or with A. A(1) and A1 are different symbols and have different meanings. With subscripted variables our check list now looks like this:

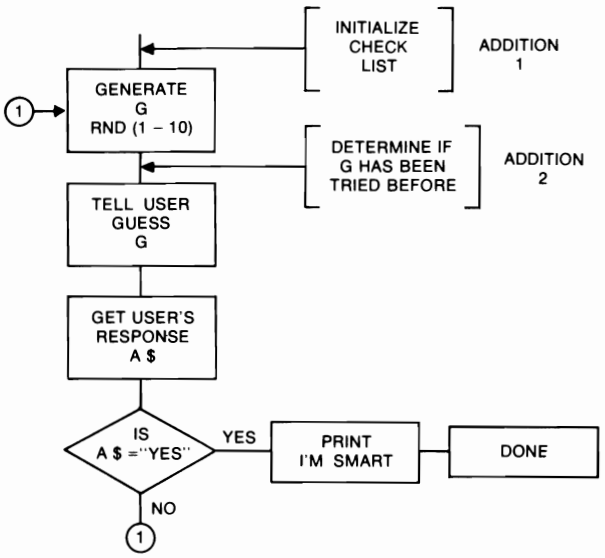
LABEL	N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)
CONTENTS										

In COMPUTER GUESS, N(3) is the label of a box the contents of which must be examined to determine if a 3 has been tried. N(7) is the label of a box the contents of which must be examined to determine if a 7 has been previously selected. What is the label of the box you would look at to determine if the number 2 had been previously selected? The general form of the subscripted variable is N(I), where I

holds a place for the value of the subscript. Of course, any other letter of the alphabet could have been used in place of I.

We are now ready to write a short version of the COMPUTER GUESS program using the subscripted variable. To do this, we refer to the same flow chart we used in writing the long version. Here again is the flow chart, including the two routines INITIALIZE CHECK LIST and DETERMINE IF G HAS BEEN TRIED.

COMPUTER GUESS/CHECK FLOW CHART



As in writing the long version, let us first encode each of the two routines to be added to the elementary version of COMPUTER GUESS. The first routine, INITIALIZE CHECK LIST, puts zeroes in each of the ten boxes. In the long program, ten lines were required to put zeroes in the ten boxes. Using the subscripted variable we do it in three:

```
5 FOR I = 1 TO 10
6 LET N(I) = 0
7 NEXT I
```

To verify that these three lines put a zero in each of the ten boxes, let's play computer and execute these steps longhand. For bookkeeping purposes, set up an array of ten boxes appropriately labeled, and one box to keep track of the value of I.

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I

Look at line 5. The value of I goes from 1 to 10 in steps of 1. When I = 1, LET N(I) = 0 becomes LET N(1) = 0. The computer's memory looks like this.

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø										1

I = 1

When I = 2, LET N(I) = 0 becomes LET N(2) = 0.
The computer's memory now looks like this:

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø									2

I = 2

When I = 3, LET N(I) = 0 becomes LET N(3) = 0.
The computer's memory now looks like this:

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø								3

I = 3

This schematic shows the computer's memory at each step in the initializing routine.

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø							4

I = 4

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø	Ø						5

I = 5

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø	Ø	Ø					6

I = 6

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø	Ø	Ø	Ø				7

I = 7

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø			8

I = 8

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø		9

I = 9

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	I
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	10

I = 10

Those three short steps in your program

```
FOR I = 1 TO 10
LET N(I) = 0
NEXT I
```

did a lot of work in the computer.

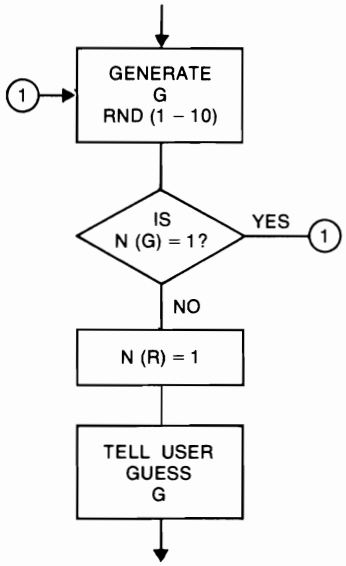
When you use a subscripted variable in a program, you must reserve at the outset an adequate number of storage boxes in the computer's memory. In this particular program we need ten memory spaces set aside. At the beginning of the program we write DIM N(10) where DIM is an abbreviation for the word DIMENSION. If you were programming the computer to play COMPUTER GUESS in the interval 1 - 25, you would write DIM N(25). If you wanted to play COMPUTER GUESS between 1 and some number X, you would have to decide beforehand the maximum value of X, say 100, and write DIM N(100). You cannot write DIM N(X) and later give a value to X by inserting INPUT X in the program. The computer must know at the beginning of the program the maximum number of boxes that may be required, but you don't have to use them all. DIM N(100) would thus allow the user to play COMPUTER GUESS between 1 and any number up to 100. Finally, if you needed three subscripted variables in a program, you could for example write DIM A(16), M(6), D(12) all on one line.

To sum up, here are the four lines we will use to initialize the check list in COMPUTER GUESS 1 TO 10, utilizing the subscripted variable and the DIM statement.

```
2 DIM N(10)
5 FOR I = 1 TO 10
6 LET N(I) = 0
7 NEXT I
```

Now to use the subscripted variable to construct the second routine which will determine if a number has already been tried. The routine DETERMINE IF G HAS BEEN TRIED BEFORE will first be represented in a flow chart. Assume the computer has just generated a number G between 1 and 10. The next step is to determine if the number has already been tried. To do this the computer looks in box N(G). For example, if the computer generated 8 as the value of G, it looks in the box labeled N(8). If it generated 2, it looks in N(2). Suppose G is 7, which box must be inspected? Here is the flow chart for this routine.

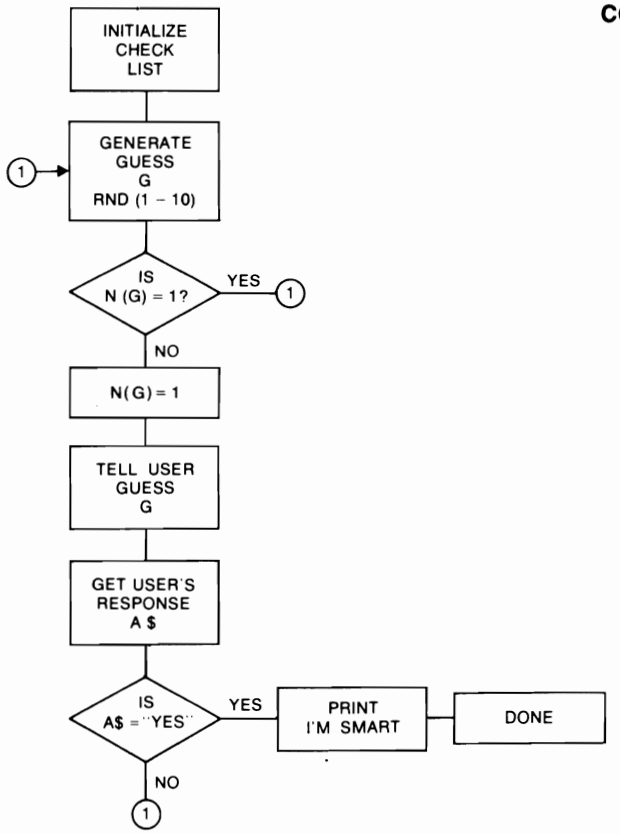
G TRIED BEFORE/1 DIAMOND FLOW CHART



Compare this flow chart with the earlier chart for the same routine. The long way required nineteen diamonds; the short way only one.

Here is the flow chart which combines the two routines, INITIALIZE CHECK LIST and DETERMINE IF G HAS BEEN TRIED BEFORE, with the elementary flow chart for COMPUTER GUESS.

COMPUTER GUESS/1 QUESTION FLOW CHART



The following program is encoded from this flow chart.

```
PROGRAM 1 RANDOM
        2 DIM N(10)
INITIALIZE
CHECK
LIST   10 FOR I=1 TO 10
       11 LET N(I)=0
       12 NEXT I
       15 PRINT "THINK OF A NUMBER FROM 1 TO 10."
       20 LET G=INT(10*RND(X))+1
G TRIED
BEFORE 30 IF N(G)=1 THEN 20
       35 LET N(G)=1
       40 PRINT "I GUESS" G;
       60 INPUT A$
       80 IF A$="YES" THEN 120
      100 GO TO 20
      120 PRINT "I'M SMART."
      140 END
```

```
OUTPUT THINK OF A NUMBER FROM 1 TO 10.
       I GUESS  2 ? NO
       I GUESS  10 ? NO
       I GUESS  4 ? NO
       I GUESS  7 ? NO
       I GUESS  1 ? NO
       I GUESS  5 ? NO
       I GUESS  6 ? YES
       I'M SMART.
```

Compare this program with the long program in the previous section. The long way required 60 steps; the short way only 15!

EXERCISE SET 3.1.3

Off-line:

1. Play computer and determine the output generated by each of the following programs. Then add an appropriate DIM statement which will tell the computer the exact number of boxes to set aside for the subscripted variable(s) in the programs.

A)
```
10 FOR I=1 TO 5
20 LET A(I)=I
30 NEXT I
40 FOR I=1 TO 5
50 PRINT A(I)
60 NEXT I
70 END
```

B)
```
10 FOR J=1 TO 10
20 LET N(J)=3
30 NEXT J
40 PRINT N(2)*N(4)
50 END
```

C)
```
10 FOR A=1 TO 5
20 LET A(A)=2*A
30 LET B(A)=A(A)
40 NEXT A
50 PRINT B(4)*A(3)
60 END
```

D)
```
10 FOR X=5 TO 10
20 LET A(X)=X
30 LET A(X+1)=A(X)
40 NEXT X
50 LET X=11
60 PRINT A(X)*A(X-1)
70 END
```

E)
```
10 LET P(1)=1
20 FOR Q=2 TO 10
30 LET P(Q)=P(Q-1)+Q
40 NEXT Q
50 FOR P=1 TO 10
60 PRINT P(P)
70 NEXT P
80 END
```

2. Here is the elementary version of a YOU GUESS program in which the computer thinks of a number between 1 and 20. When you try to guess the computer's number, you might forget your previous guesses and try the same number twice. Use the subscripted variable to set up a check list routine that will enable the computer to determine whether you have tried that number before and, if so, respond with the message: DUMMY! YOU'VE ALREADY TRIED THAT NUMBER.

```
  1 RANDOM
 10 PRINT "I'M THINKING OF A MYSTERY NUMBER BETWEEN 1 AND 20."
 20 LET R=INT(20*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 20."
 40 INPUT G
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
120 END
```

On-line:

3. Add all of the following features to the most recent version of COMPUTER GUESS, COMPUTER GUESS/1 QUESTION; then load and run your new program. Name this program COM/G1 and save for further modification in the next exercise set.
   A. Steps which give the user more instructions on how to play the game.
   B. Steps which cause the bell to ring five times after I'M SMART is printed. Be sure to suppress the line feed each time the bell rings.
   C. Steps to handle MAYBE, or any word other than YES or NO, in response to the computer's guess.
4. Draw a flow chart and write a program that will produce a listing of all the whole numbers between 1 and 25 inclusive in random order. Each number between 1 and 25 inclusive should appear only once in the list and the list should contain 25 numbers. Your program should generate an output like this.

```
24          12          23          21          20
17          25          5           3           10
6           1           16          4           13
19          9           18          15          2
14          11          7           22          8
```

3.2 "PLAY IT AGAIN, SAM?"

As in YOU GUESS, let's give the user the opportunity to replay COMPUTER GUESS without typing in RUN each time. Recall the replay options explained in section 2.2. Briefly they were: (1) *Automatic Replay*. At the end of the game you inserted a GO TO statement which directed the computer back to the beginning of the game. When the user wanted to stop playing, he halted the computer by typing CONTROL C. (2) *Ask the User*. The computer asked the user at the end of each game whether he wished to play again. The string variable enabled the user to respond with the words YES and NO. (3) *Flag 99*. You built into your program a code number, 99, which the user typed in when he wanted to quit playing. (4) *How Many Games?* At the start of the program the computer asked the user how many games he wanted to play. A FOR-NEXT loop then directed the computer to play the number of games the user specified.

Even though the tables have been turned in this chapter, the program steps for three of the four replay options remain the same. Only the Flag 99 option needs to be modified. Look again at the YOU GUESS program with the Flag 99 option.

YOU GUESS/FLAG 99

PROGRAM

```
  1 RANDOM
 10 PRINT "WHENEVER YOU WISH TO STOP PLAYING"
 11 PRINT "TYPE IN THE NUMBER 99."
 15 PRINT
 20 LET R=INT(10*RND(X))+1
 35 PRINT "GUESS THE NUMBER BETWEEN 1 AND 10";
 40 INPUT G
 50 IF G=99 THEN 130
 60 IF G=R THEN 100
 70 PRINT "GUESS AGAIN";
 80 GO TO 40
100 PRINT "CORRECT."
105 PRINT
110 GO TO 20
130 PRINT "THANKS FOR PLAYING."
135 PRINT "COME BACK SOON."
140 END
```

Locate and identify the following features in the above program.

(1) The steps to instruct the user how to terminate play.
(2) The step which determines if the user typed in the flag 99.
(3) The step which directs the computer to restart the game.
(4) The steps to type out the sign-off message if the flag has been typed in by the user.

The instructions occur in lines 10 and 11. The flag test occurs in line 50. Restart occurs in line 110. Sign-off

occurs in lines 130 and 135. Where would you insert these four functions in the following COMPUTER GUESS program?

COMPUTER GUESS/1 QUESTION

PROGRAM

```
  1 RANDOM
  2 DIM N(10)
 10 FOR I=1 TO 10
 11 LET N(I)=0
 12 NEXT I
 15 PRINT "THINK OF A NUMBER FROM 1 TO 10."
 20 LET G=INT(10*RND(X))+1
 30 IF N(G)=1 THEN 20
 35 LET N(G)=1
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
140 END
```

The two steps that give instructions should go in ahead of line 10. The only space available is between lines 2 and 10. Let's use lines 5 and 6.

5 PRINT "WHENEVER YOU WISH TO STOP PLAYING"

6 PRINT "TYPE IN THE NUMBER 99."

The step to restart the game should be inserted after the computer types out the message I'M SMART. Let's use line 130. Since the game starts at line 10 with the initialization of the check list, the GO TO statement is:

130 GO TO 10

The steps to sign off occur at the end of the program. Let's use lines 140 and 145, and make line 150 the new location for the END statement.

140 PRINT "THANKS FOR PLAYING."

145 PRINT "COME BACK SOON."

150 END

Now to find a location for the flag test. Only one line in the COMPUTER GUESS program gives the user an opportunity to type in any response, including 99. It is line 60, INPUT A$. Thus the flag test must come after line 60, but before the test for YES on line 80. Let's use line 70. Which of the following IF-THEN statements would you use?

(1) 70 IF A$ = 99 THEN 140

(2) 70 IF A$ = "99" THEN 140

The correct choice is (2). Here is the reason. The string variable A$ appears in line 60. The word INPUT also appears in line 60. Any set of characters typed in by the user in response to the question mark generated by the word INPUT on line 60 will be assigned to A$. You must immediately instruct the computer to determine if the set of characters assigned to A$ is the same as the string of characters in the flag, that is 99. To test if the string of characters typed in by the user is 99, the 99 you

insert in line 70 must be in quotes. Here is COMPUTER GUESS with the Flag 99 option.

COMPUTER GUESS/FLAG "99"

PROGRAM

```
 1 RANDOM
 2 DIM N(10)
 5 PRINT "WHENEVER YOU WISH TO STOP PLAYING"
 6 PRINT "TYPE IN THE NUMBER 99."
 7 PRINT
10 FOR I=1 TO 10
11 LET N(I)=0
12 NEXT I
15 PRINT "THINK OF A NUMBER FROM 1 TO 10."
20 LET G=INT(10*RND(X))+1
30 IF N(G)=1 THEN 20
35 LET N(G)=1
40 PRINT "I GUESS" G;
60 INPUT A$
70 IF A$="99" THEN 140
80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
125 PRINT
130 GO TO 10
140 PRINT "THANKS FOR PLAYING."
145 PRINT "COME BACK SOON."
150 END
```

FLAG INSTRUCTION (lines 5–6)

FLAG "99" (line 70)

RESTART GAME (line 130)

SIGN OFF (lines 140–145)

Here is a typical game played using the above program:

OUTPUT

```
WHENEVER YOU WISH TO STOP PLAYING
TYPE IN THE NUMBER 99.

THINK OF A NUMBER FROM 1 TO 10.
I GUESS 5 ? NO
I GUESS 10 ? NO
I GUESS 4 ? NO
I GUESS 6 ? YES
I'M SMART.

THINK OF A NUMBER FROM 1 TO 10.
I GUESS 8 ? 99
THANKS FOR PLAYING.
COME BACK SOON.
```

When used as a flag to terminate the game, the number 99 does not mean much to the human player. DONE or QUIT would be better, but we couldn't use these words in the program YOU GUESS in Chapter 2 because the computer was looking for numbers in response to the question mark generated by the statement, INPUT G. Now that we have an input statement, INPUT A$, which looks at all inputs as a string of alphanumeric characters, we can use DONE or any other word. To use DONE as a flag, we change the instruction to the user in line 6, and insert DONE between quotes in line 70.

COMPUTER GUESS/FLAG "DONE"

PROGRAM

```
  1 RANDOM
  2 DIM N(10)
  5 PRINT "WHENEVER YOU WISH TO STOP PLAYING"
  6 PRINT "TYPE IN THE WORD DONE."          <-- SPECIFY FLAG
  7 PRINT
 10 FOR I=1 TO 10
 11 LET N(I)=0
 12 NEXT I
 15 PRINT "THINK OF A NUMBER FROM 1 TO 10."
 20 LET G=INT(10*RND(X))+1
 30 IF N(G)=1 THEN 20
 35 LET N(G)=1
 40 PRINT "I GUESS" G;
 60 INPUT A$
 70 IF A$="DONE" THEN 140                  <-- FLAG "DONE"
 80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
125 PRINT
130 GO TO 10
140 PRINT "THANKS FOR PLAYING."
145 PRINT "COME BACK SOON."
150 END
```

OUTPUT

```
WHENEVER YOU WISH TO STOP PLAYING
TYPE IN THE WORD DONE.

THINK OF A NUMBER FROM 1 TO 10.
I GUESS 1 ? NO
I GUESS 7 ? NO
I GUESS 8 ? NO
I GUESS 3 ? YES
I'M SMART.

THINK OF A NUMBER FROM 1 TO 10.
I GUESS 2 ? DONE
THANKS FOR PLAYING.
COME BACK SOON.
```

EXERCISE SET 3.2

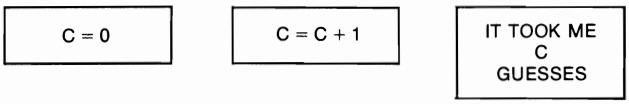
On-line:

1. Take your COMPUTER GUESS program, COM/G1, which you saved in exercise set 3.1.3. Add steps to this program to permit the user to signal the end of play by typing in a flag which is either a word or a number. Call this new program COM/G2 and save for further modification in the next exercise set.

3.3 COUNTING THE NUMBER OF GUESSES

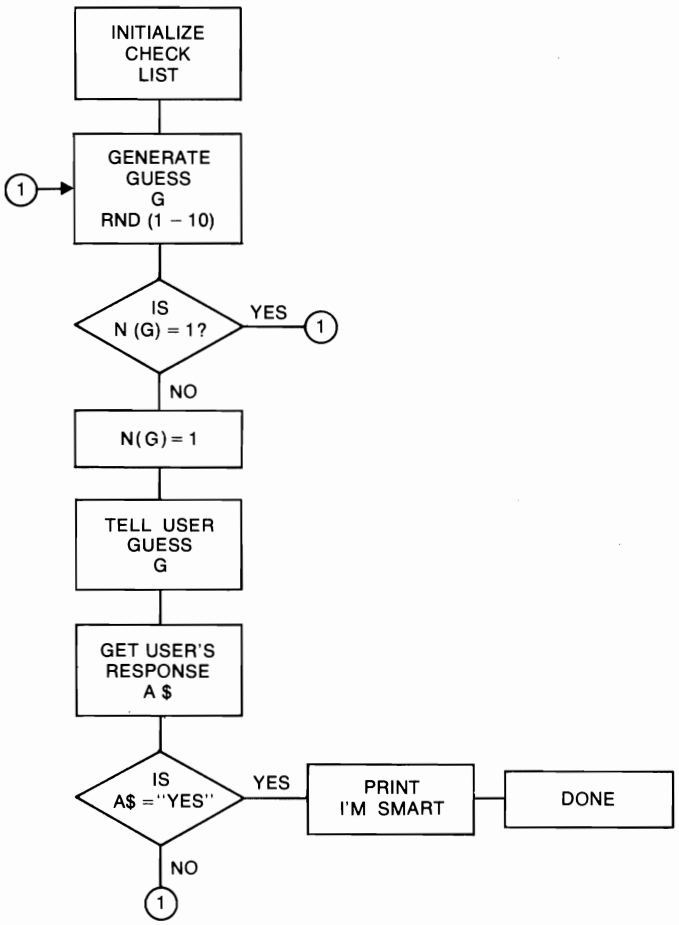
The next feature we will add to our COMPUTER GUESS game is a counter to keep track of the number of guesses made by the computer. Remember, in order to count the number of guesses in a game, three steps are

required. (1) A step which sets a counter at zero. (2) A step which increases the counter by 1 each time the computer makes a guess. (3) A step which prints out the number of guesses. They're shown here in three boxes.

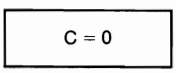


If C represents the counter, where in the following flow chart of COMPUTER GUESS would you put the three boxes?

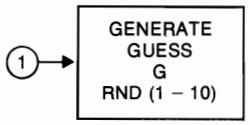
COMPUTER GUESS/1 QUESTION FLOW CHART



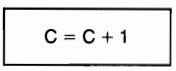
First, let's set the counter at zero by inserting



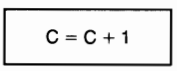
before the box



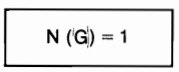
Now you must decide if you want to count the number of times the computer generates a number between 1 and 10, or the number of guesses finally typed out at the terminal. There is a difference; the first number, the count before screening for duplication, is larger than the second. Eventually we'll count both activities, but first we'll count only the number of guesses which are typed out at the terminal. The box



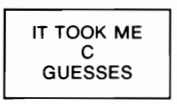
must, therefore, be inserted in the branch that departs from the NO exit of the first diamond. In short, it goes between the diamond in which you ask, "Is this number a duplicate?" and the diamond in which you essentially ask, "Is this a correct guess?" The box



is inserted after the box

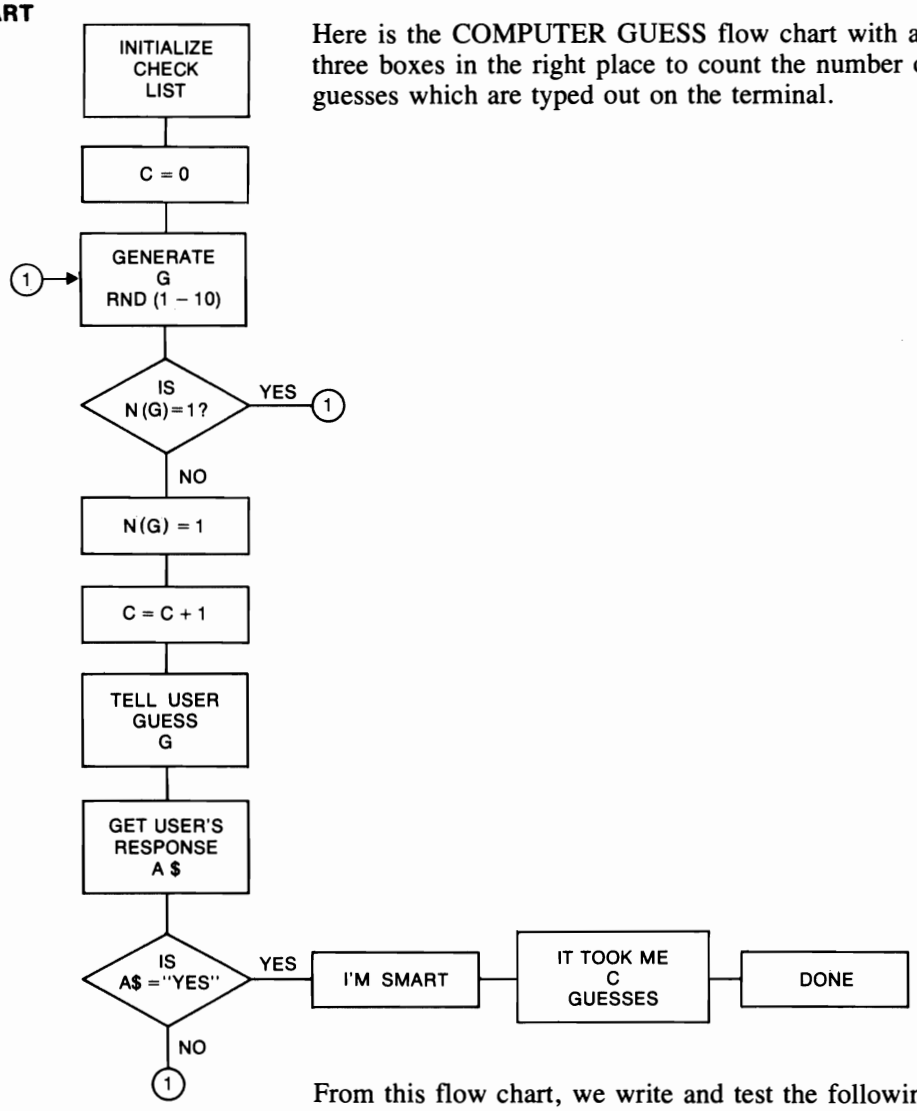


The third box,



should be inserted after

COMPUTER GUESS/COUNT GUESSES
FLOW CHART



Here is the COMPUTER GUESS flow chart with all three boxes in the right place to count the number of guesses which are typed out on the terminal.

From this flow chart, we write and test the following program.

PROGRAM

```
 1 RANDOM
 2 DIM N(10)
10 FOR I=1 TO 10
11 LET N(I)=0
12 NEXT I
14 LET C=0                                   <- INITIALIZE COUNTER
15 PRINT "THINK OF A NUMBER FROM 1 TO 10."
20 LET G=INT(10*RND(X))+1
30 IF N(G)=1 THEN 20
35 LET N(G)=1
36 LET C=C+1                                 <- INCREASE COUNTER
40 PRINT "I GUESS" G;
60 INPUT A$
80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
121 PRINT "IT TOOK ME ONLY" C "GUESSES."     <- PRINT OUT COUNTER
140 END
```

OUTPUT

```
THINK OF A NUMBER FROM 1 TO 10.
I GUESS 10 ? NO
I GUESS 5 ? NO
I GUESS 3 ? NO
I GUESS 8 ? NO
I GUESS 6 ? NO
I GUESS 9 ? NO
I GUESS 7 ? NO
I GUESS 2 ? YES
I'M SMART.
IT TOOK ME ONLY 8 GUESSES.
```

An interesting aspect of the way the computer plays the game, but one which we never see, is the number of times the computer must generate a number between 1 and 10 in order to come up with a number which passes the "no duplicates" test. In the last game, COMPUTER GUESS/COUNT GUESSES, the computer typed out eight guesses.Was that the number of random numbers generated by the statement: 20 LET G = INT(10*RND(X)) + 1? Probably not.

To count the number of times the computer generates a number between 1 and 10, three steps should be added to the program COMPUTER GUESS/COUNT GUESSES. They are:

```
 13 LET T = 0
 21 LET T = T + 1
123 PRINT "BUT I HAD TO GENERATE"
T"RANDOM NUMBERS."
```

Insert the new counter, T, immediately after the statement which generates the random number, that is line 20. Here is the complete program which counts the number of whole numbers randomly generated and the number of guesses typed out.

COMPUTER GUESS/COUNT G

PROGRAM

```
  1 RANDOM
  2 DIM N(10)
 10 FOR I=1 TO 10
 11 LET N(I)=0
 12 NEXT I
 13 LET T=0 <-------------------------- INITIALIZE NEW COUNTER
 14 LET C=0
 15 PRINT "THINK OF A NUMBER FROM 1 TO 10."
 20 LET G=INT(10*RND(X))+1
 21 LET T=T+1 <-------------------------- INCREASE NEW COUNTER
 30 IF N(G)=1 THEN 20
 35 LET N(G)=1
 36 LET C=C+1
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="YES" THEN 120
100 GO TO 20
120 PRINT "I'M SMART."
121 PRINT "IT TOOK ME ONLY" C "GUESSES."
122 PRINT
123 PRINT "BUT I HAD TO GENERATE" T "RANDOM NUMBERS." <- PRINT OUT NEW COUNTER
140 END
```

OUTPUT

```
THINK OF A NUMBER FROM 1 TO 10.
I GUESS 8 ? NO
I GUESS 7 ? NO
I GUESS 9 ? NO
I GUESS 4 ? NO
I GUESS 2 ? NO
I GUESS 1 ? NO
I GUESS 3 ? YES
I'M SMART.
IT TOOK ME ONLY 7 GUESSES.
BUT I HAD TO GENERATE 10 RANDOM NUMBERS.
```

EXERCISE SET 3.3

On-line:

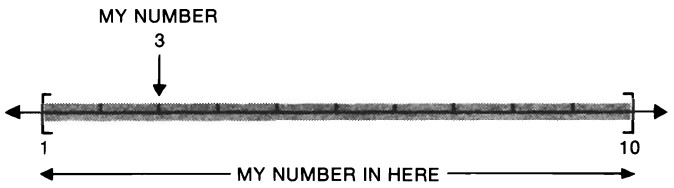
1. Take your COMPUTER GUESS program, COM/G2, which you saved in exercise set 3.2 and add steps to count the number of guesses typed out by the computer in order to hit upon the user's number. Add a second counter to count the number of whole numbers randomly generated by the computer in guessing the user's number. This new program will not be used again.
2. Draw a flow chart and write a program that will list in random order all of the whole numbers between 1 and 52 inclusive. Each number between 1 and 52 should appear only once in the list so that the list contains 52 numbers. After all the numbers have been listed, have the program type out the number of whole numbers randomly generated by the computer. On the average, how many whole numbers do you think the computer must randomly generate in order to produce a list without duplicates? Your program should produce an output like this:

```
47          13          15          51          14
48          6           18          19          9
5           11          35          17          41
8           52          4           24          36
42          21          39          30          29
37          49          10          40          43
33          23          32          34          7
44          2           38          31          27
28          1           20          16          46
3           25          45          26          12
22          50
BUT I HAD TO GENERATE 182 RANDOM NUMBERS.
```

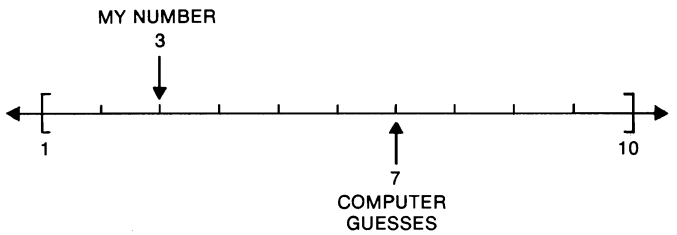
3.4 "TOO HIGH TOO LOW": THE RANDOM METHOD

In Chapter 2 we instructed the computer to tell the user, after each guess, whether his guess was too high, too low, or right on. This information enabled the user to make his next guess smaller or larger and to develop two guessing strategies. One strategy was called the Random Method and the other the Midpoint Method. Both methods were described in detail in section 2.6.

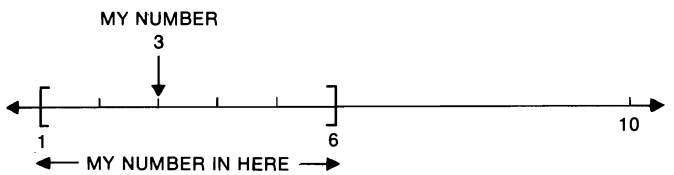
Now that we have turned the tables, we would like to teach the computer the Random Method for guessing the mystery number the human user was thinking of. In developing our new program, COMPUTER GUESS/RANDOM METHOD, it is unnecessary to have the computer keep a check list of previous guesses. As long as it operates on the feedback information typed in by the human player, it will be impossible for the computer to guess the same number again. For example, suppose I mentally pick 3 as the mystery number to be guessed by the computer. For its first guess the computer is to select a whole number at random from the interval 1 to 10 inclusive. Here is a graphic representation of the problem confronting the computer.



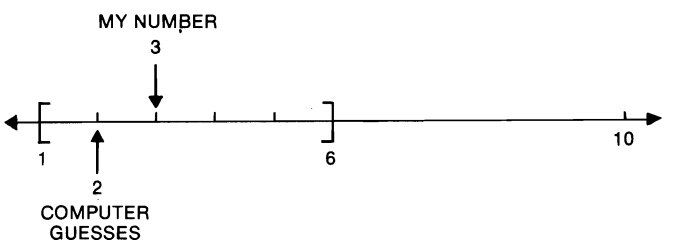
Say 7 is the first number generated by the computer.



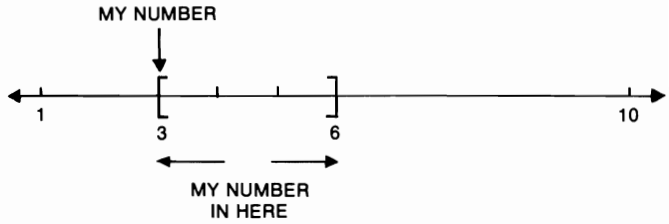
I respond by typing in H for high. The computer must now generate a whole number which belongs to the interval 1-6.



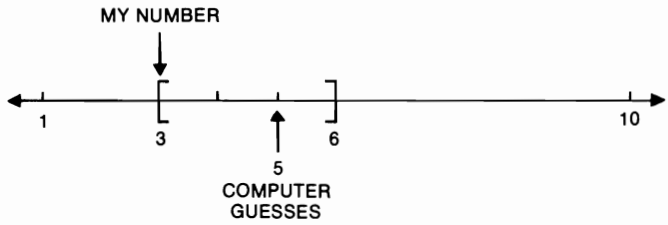
Say 2 is the next number it generated.



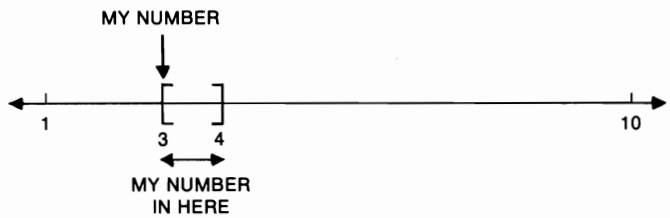
I respond by typing L for low. The computer must now generate a number which belongs to the interval 3-6.



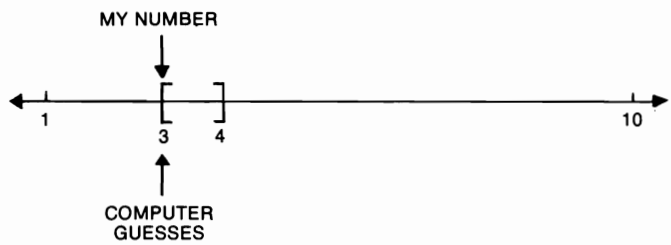
Say 5 is the third number generated.



I respond by typing H for high. Computer must now generate a number which belongs to the interval 3-4.

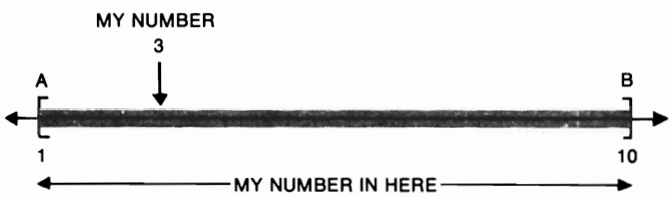


Say 3 is the next number selected.

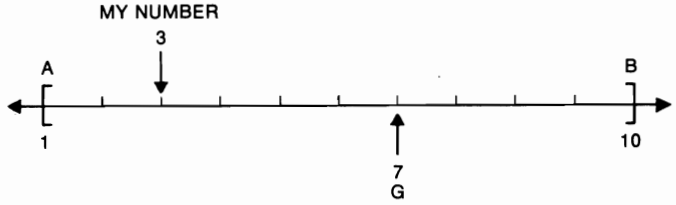


I respond by typing R for right and the game is over.

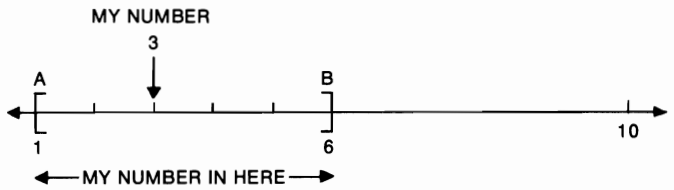
Let me repeat the foregoing description using variable names for the left boundary (A) and right boundary (B) of the interval and for the computer's guesses (G). Thus, the first guess is a number belonging to the interval A to B inclusive where A is 1 and B is 10.



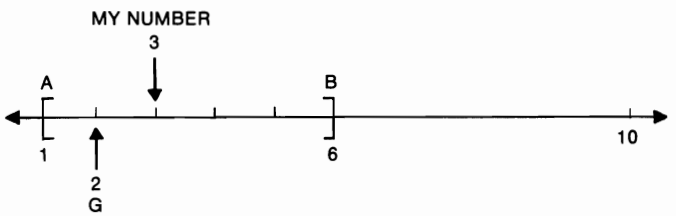
The computer generates 7 for its first guess, denoted by G.



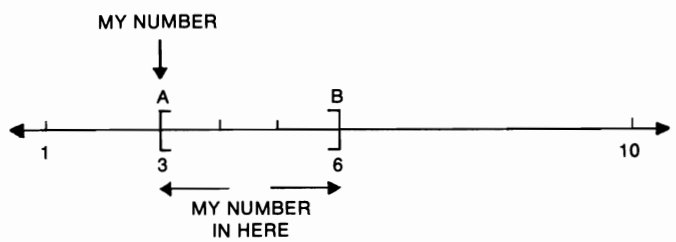
This is too high, so B is redefined as 6, or 1 less than G. That is, B = G − 1. Thus, the next guess is from:



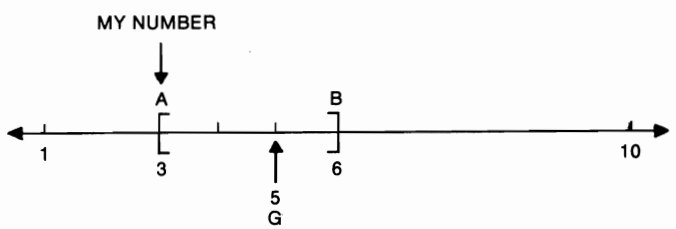
The next guess is 2.



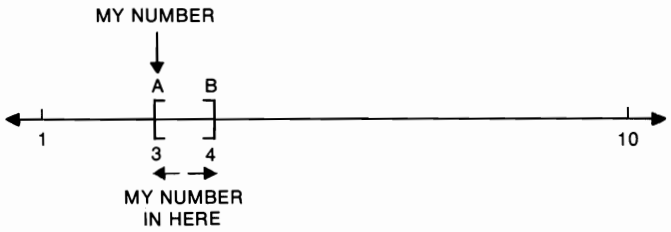
Since 2 is too low, the left boundary, A, must be moved towards the right, or 1 beyond the last guess. That is, A = G + 1 or 3. The next guess is from:



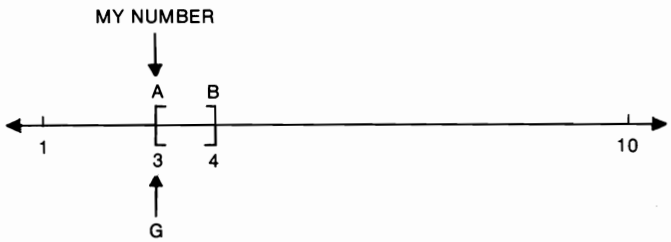
The next guess is 5.



Since 5 is too high, B is moved to the left. Thus, B = G − 1 or 4. The next guess is from:

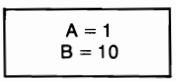


The next guess is 3.

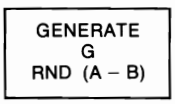


The computer has hit upon my mystery number and the game is over.

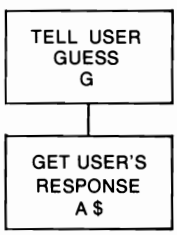
Now that the routine for our new COMPUTER GUESS game has been laid out in detail, let's draw a flow chart which specifies the sequence of steps to be followed. The first step is to define the end points of the interval from which the computer is to make its initial guess.



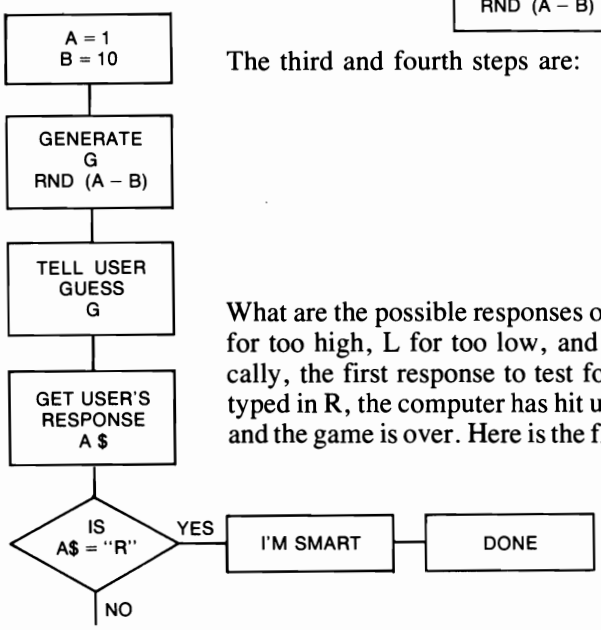
The next step is to have the computer generate a random whole number between A and B inclusive.



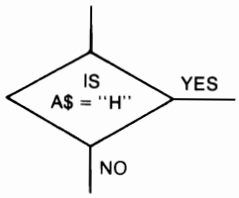
The third and fourth steps are:



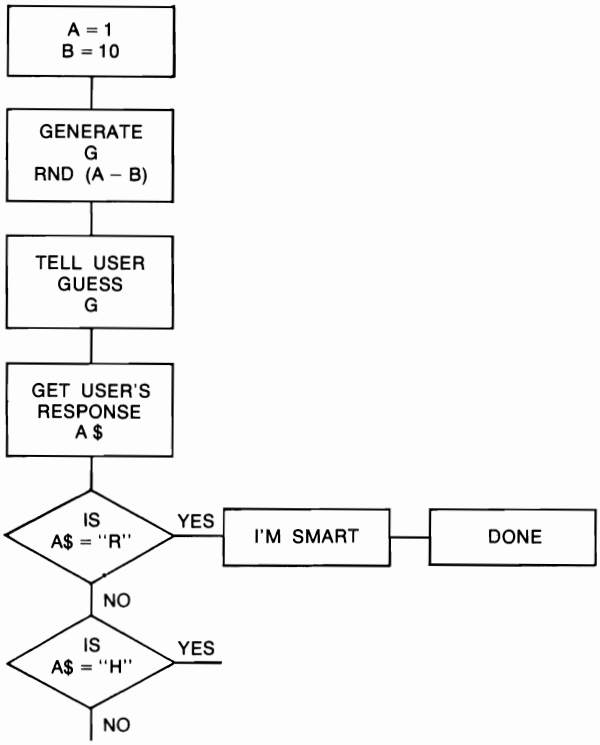
What are the possible responses of the user? They are H for too high, L for too low, and R for right on. Logically, the first response to test for is R. If the user has typed in R, the computer has hit upon the user's number and the game is over. Here is the flow chart to this point.



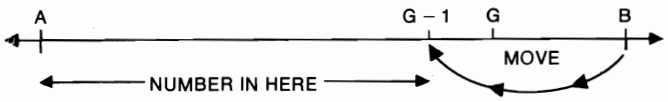
If the user did not type in R, then the computer must determine if he typed H or L. Let's program it first to look for H.



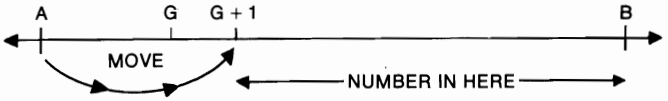
What if the user has typed in neither R nor H? Do we need a third diamond to test for an L? No, if the user has cooperated by typing in H, R, or L, then the NO exit from the above diamond leaves L as the only alternative. Here is the flow chart to this point.



There are two branches to be completed, one YES and one NO. Take the YES branch first. That is, the computer's guess was too large and its next guess must be a smaller number. To insure that the next guess is smaller, the right end point of the interval, B, must be made one less than the last guess. Thus B = G − 1.

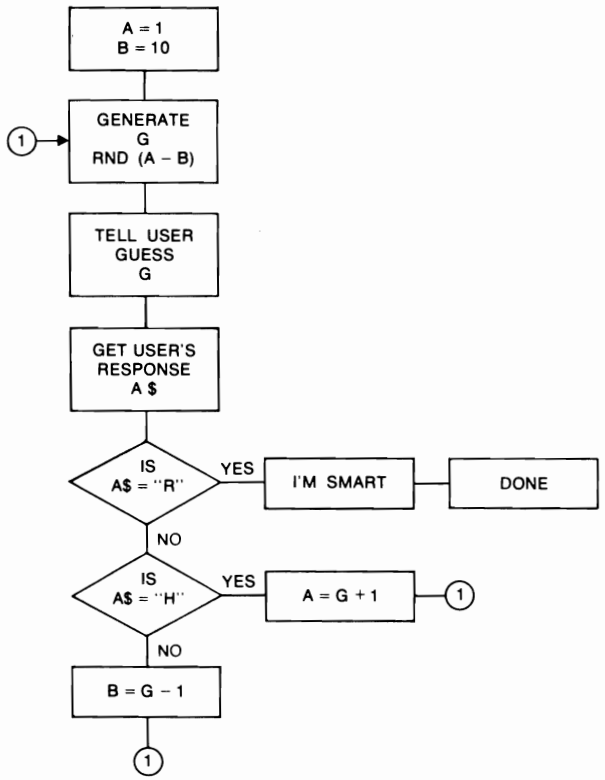


Now for the NO branch. That is, the computer's guess was too small and its next guess must be a larger number. To insure that the next guess is larger, the left end point of the interval, A, must be made one more than the last guess. Thus A = G + 1.



After either of these events, B = G − 1 or A = G + 1, the next step is to loop back to the box where the computer generates a number between A and B. Here is the complete flow chart.

COMPUTER GUESS/RANDOM METHOD FLOW CHART



From this flow chart the following program is written and tested.

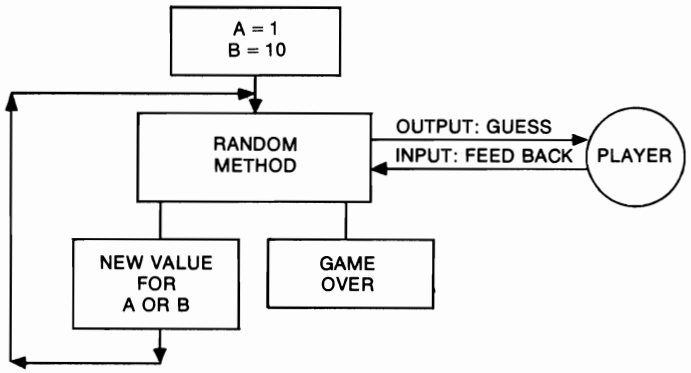
PROGRAM

```
  1 RANDOM
 10 LET A=1                                  <- INITIAL INTERVAL
 15 LET B=10                                 <-
 16 PRINT "THINK OF A NUMBER FROM 1 TO 10."
 20 LET G=INT((B-A+1)*RND(X))+A              <- GENERATE GUESS
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="R" THEN 200
100 IF A$="H" THEN 160
120 LET A=G+1                               <- NEW LEFT END POINT
140 GO TO 20
160 LET B=G-1                               <- NEW RIGHT END POINT
180 GO TO 20
200 PRINT "I'M SMART!"
210 END
```

OUTPUT

```
THINK OF A NUMBER FROM 1 TO 10.
I GUESS 6 ? H
I GUESS 1 ? L
I GUESS 2 ? R
I'M SMART!
```

To sum up, here is a simplified block diagram of COMPUTER GUESS/RANDOM METHOD, showing the role of the human player.



At the start the computer is told the initial values of A and B. After that the computer generates a random number, outputs the number, and gathers feedback. Depending on the feedback, the game is either over or a new value for A or B is computed. If a new value is computed for one of the end points, the computer then loops back to the start of the procedure. This procedure is repeated until the computer hits upon the user's number. This will occur after a finite number of repetitions, or iterations.

Our new COMPUTER GUESS/RANDOM METHOD game will be even more fun if the player can choose as the mystery number a number which belongs to a larger interval. Suppose the user wants to play COMPUTER GUESS/RANDOM METHOD between 1 and 1000; what changes need to be made in the following COMPUTER GUESS/RANDOM METHOD program?

PROGRAM **COMPUTER GUESS/ RANDOM METHOD**

```
  1 RANDOM
 10 LET A=1
 15 LET B=10
 20 LET G=INT((B-A+1)*RND(X))+A
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="R" THEN 200
100 IF A$="H" THEN 160
120 LET A=G+1
140 GO TO 20
160 LET B=G-1
180 GO TO 20
200 PRINT "I'M SMART!"
210 END
```

All that we need to do is to let the user specify the left end point of the interval, A, and the right end point of the interval, B. Thus we substitute the following for lines 10 and 15 above.

```
10 PRINT "GIVE ME LOWER AND UPPER
   NUMBERS";
15 INPUT A, B
```

Here is the complete program with a sample run. I picked 400 as the number to be guessed in the interval 123 to 432.

COMPUTER GUESS/RANDOM METHOD/ EXPAND INTERVAL

PROGRAM

```
 1 RANDOM
10 PRINT "GIVE ME LOWER AND UPPER NUMBERS";
15 INPUT A,B
20 LET G=INT((B-A+1)*RND(X))+A
40 PRINT "I GUESS" G;
60 INPUT A$
80 IF A$="R" THEN 200
100 IF A$="H" THEN 160
120 LET A=G+1
140 GO TO 20
160 LET B=G-1
180 GO TO 20
200 PRINT "I'M SMART!"
210 END
```

OUTPUT

```
GIVE ME LOWER AND UPPER NUMBERS? 123,432
I GUESS 351 ? L
I GUESS 415 ? H
I GUESS 410 ? H
I GUESS 404 ? H
I GUESS 356 ? L
I GUESS 374 ? L
I GUESS 388 ? L
I GUESS 395 ? L
I GUESS 400 ? R
I'M SMART!
```

EXERCISE SET 3.4

On-line:

1. Incorporate all of the following features into our new COMPUTER GUESS/RANDOM METHOD program. Load and run this new program and save as COM/G4.
   - A. Add steps which tell the user how to play the game; also add any other layout features and computer personality that you desire.
   - B. Add steps to handle inputs other than H, R, or L in response to the computer's guess.
   - C. Add steps that employ one of the replay options described in sections 2.2 and 3.2.
   - D. Add steps that count the number of guesses required by the computer to hit upon the user's number.
2. Take your program COM/G4 and change it so that you will be able to choose as the mystery number any number between 1 and 1000. This means that A should be set at the value 1, and B at the value 1000. Play five games with your program and insert in the following table the number of guesses needed by the computer for each game, and the average number of guesses needed for the five games. Save this new program as COM/G5 for use in the next section.

GAME #	# GUESSES
1	
2	
3	
4	
5	
AVERAGE	

3.5 "TOO HIGH TOO LOW": THE MIDPOINT METHOD

We leave the Random Method to focus on a second strategy, the Midpoint Method, which was outlined in section 2.6.2. Both methods depend on feedback from the user. Let's review the Midpoint Method as you used it when playing YOU GUESS games in the exercise set at the end of Chapter 2. After all, if you don't know the method in detail, you can't teach it to the computer.

Suppose in one of the games the computer picked a mystery number between 1 and 1000. Your first guess should have been 500, "halfway" between 1 and 1000. Then, if 500 was not correct, your next guess should have been either 250, "halfway" between 1 and 500; or 750, "halfway" between 500 and 1000. Each time you guessed, your new guess (if needed) should have come from the midpoint of a new, smaller interval.

In the Midpoint Method, you are repeatedly finding the midpoint of an interval by averaging the two numbers marking the end points. In short, you add up the two numbers and divide by 2. Thus, if the original interval is 1 to 1000, the average of 1 and 1000 is

$$\frac{1 + 1000}{2} \quad \text{or} \quad \frac{1001}{2} \quad \text{or} \quad 500.5.$$

To avoid obtaining mixed numbers, round off the number by using the next lower whole number, that is, by "rounding down." Thus, 500.5. becomes 500. This method has one limitation: you'll never guess the number 1000. Likewise, if you "round up," you'll never guess the number 1. I'll leave it to you to discover why. You can get around this difficulty by making one end point of the interval the number 0, and the other end point the number 1001. Check this out yourself if you wish.

Now let's teach the computer the Midpoint Method. Let A and B mark the end points of the interval which contain the number. Let the initial value of A and B be 0 and 1001 respectively. Let G be the computer's guess, the average of A and B:

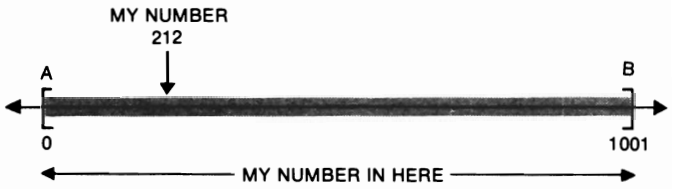
$$G = \frac{A + B}{2}$$

To round G down, use INT:

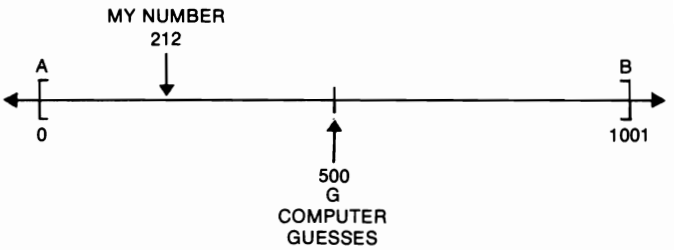
$$G = \text{INT}\left[\frac{A + B}{2}\right].$$

If necessary, refer to section 2.1.1 for the operation of INT.

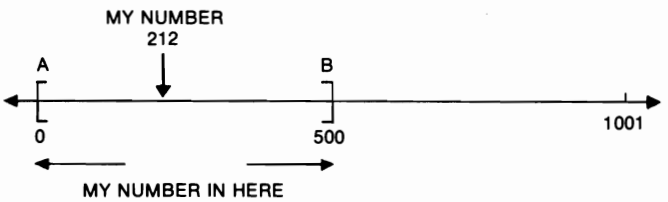
Here is a graphic demonstration of the procedure. Suppose I pick 212 as the mystery number to be guessed by the computer.



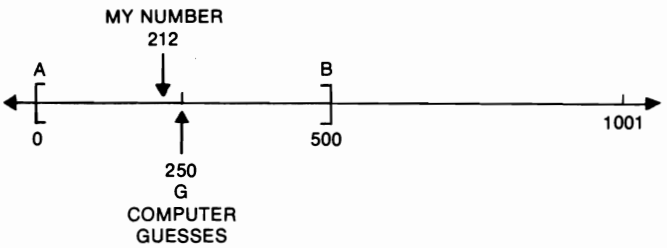
The computer's first guess is
INT((A + B)/2) = INT((0 + 1001)/2) =
INT(1001/2) = INT(500.5) = 500.



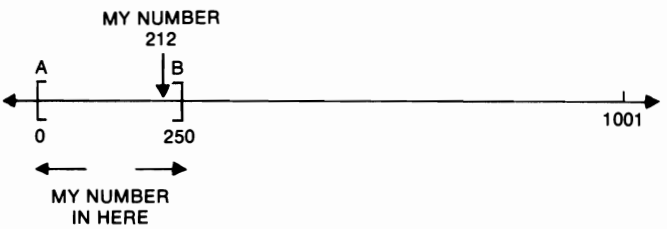
This guess is too high. The computer must move the right end point B to the left by setting B equal to G.



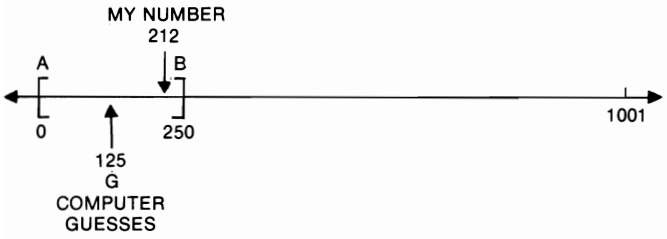
The computer's second guess is
INT((0 + 500)/2) = INT(500/2) = INT(250) = 250.



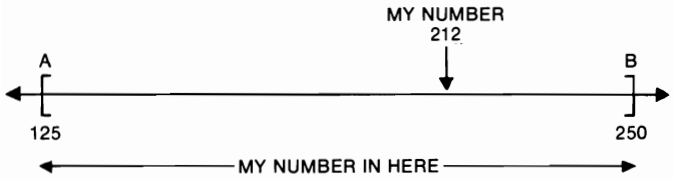
This guess is still too high. The computer again sets B equal to G.



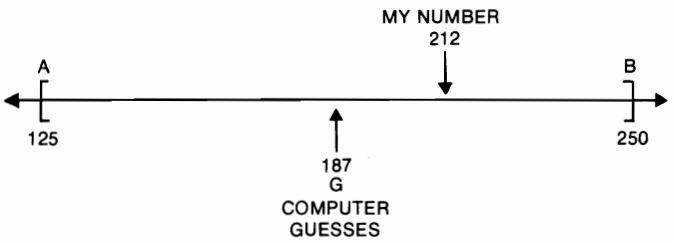
The computer's next guess is
INT((0 + 250)/2) = INT(250/2) = INT(125) = 125.



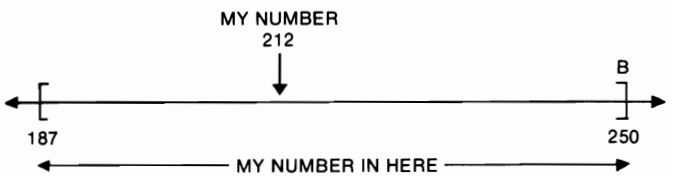
This time G is too low. The computer must move A to the right, setting A equal to G. The target area is getting smaller, so I'll change the scale of my drawing, showing only the interval of interest.



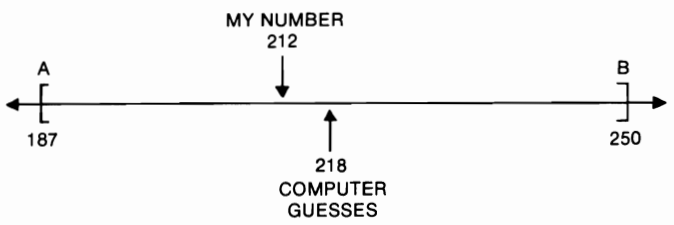
The computer's next guess is
INT((125 + 250)/2) = INT(375/2) = INT(187.5) = 187.



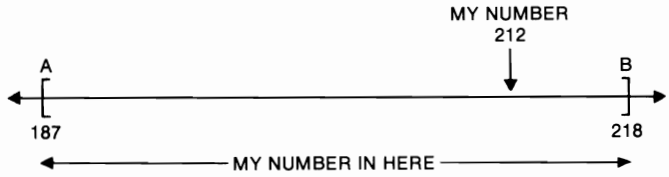
G is still too low. The computer again moves A to the right, setting A equal to G.



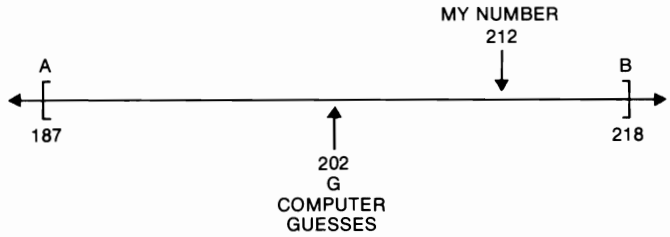
The computer's next guess is
INT((187 + 250)/2) = INT(437/2) = INT(218.5) = 218.



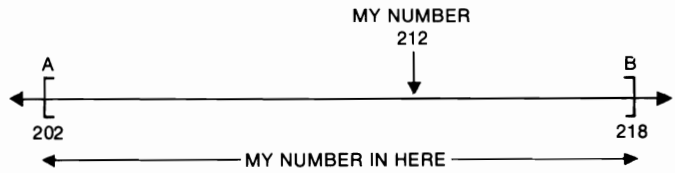
This is too high, so it sets B = G.



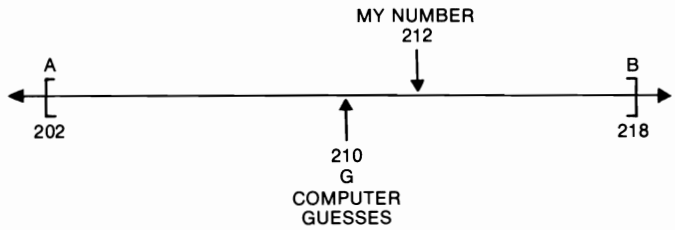
The computer's next guess is
INT ((187 + 218)/2) = INT(405/2) = INT(202.5) = 202.



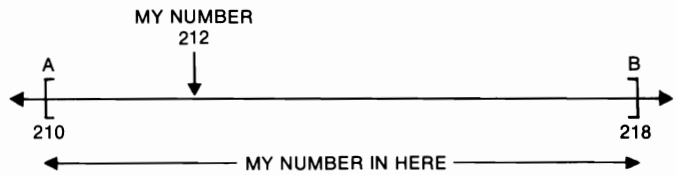
This is too low, so it sets A = G.



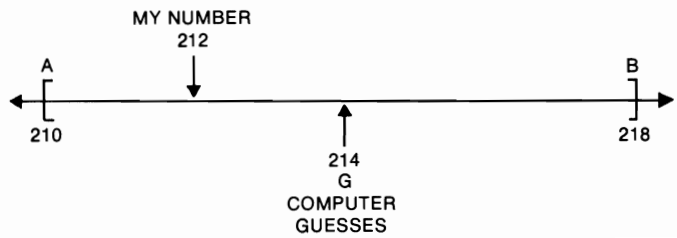
The computer's next guess is
INT((202 + 218)/2) = INT(420/2) = INT(210) = 210.



G is too low, so it sets A = G.

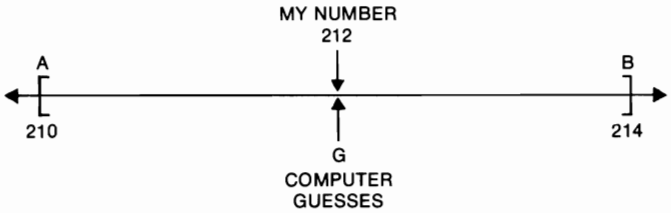


The computer's next guess is
INT((210 + 218)/2) = INT(428/2) = INT(214) = 214.



G is too high, so it sets B = G.

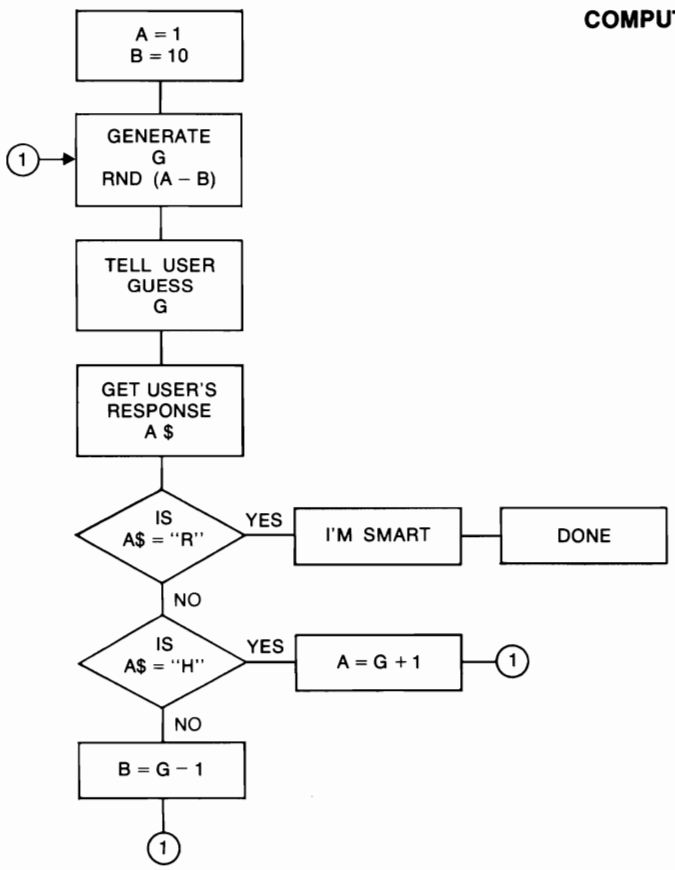
The computer's next guess is
INT((210 + 214)/2) = INT(424/2) = INT(212) = 212.



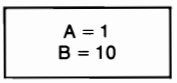
The guess is correct! It took nine guesses for the computer, following the Midpoint Method, to trap the mystery number in the middle of an interval two units in length.

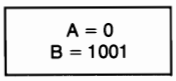
Now that we know the method, let's teach it to the computer. An easy way to do this is to look at the flow chart for COMPUTER GUESS/RANDOM METHOD and ask yourself what changes are necessary.

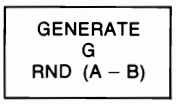
COMPUTER GUESS/RANDOM METHOD FLOW CHART

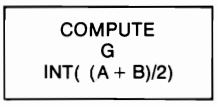


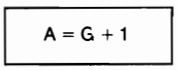
Four changes are required to transform this flow chart into COMPUTER GUESS/MIDPOINT METHOD.

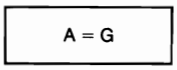
First, change 

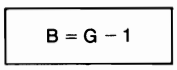
to 

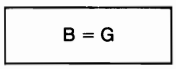
Second, change 

to 

Next, change 

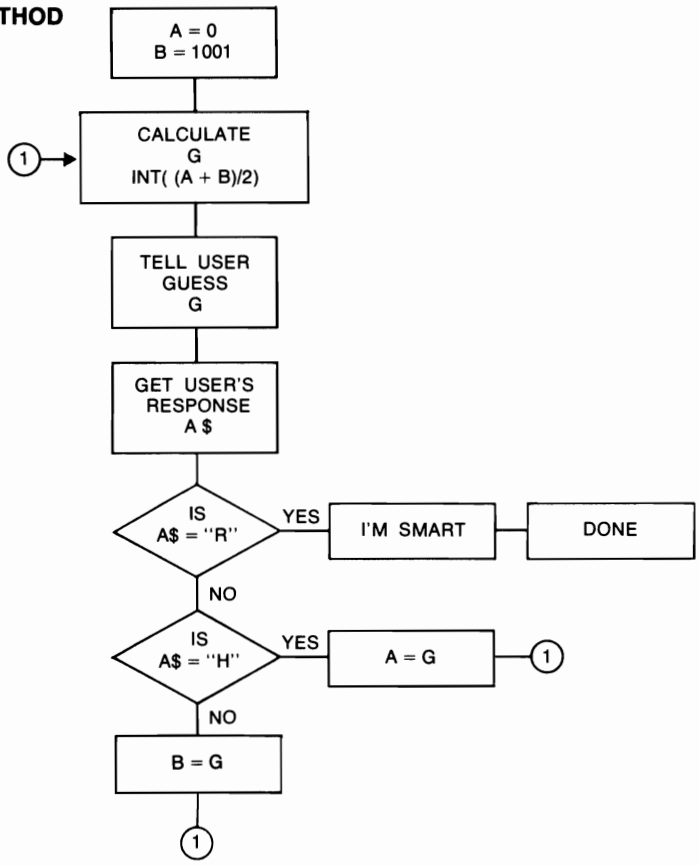
to 

and 

to 

Here is the flow chart for COMPUTER GUESS/ MIDPOINT METHOD.

COMPUTER GUESS/MIDPOINT METHOD FLOW CHART



From this flow chart the following program is written and tested.

PROGRAM

```
 10 LET A=0
 15 LET B=1001
 16 PRINT "THINK OF A NUMBER FROM 1 TO 1000."
 20 LET G=INT((A+B)/2)
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="R" THEN 200
100 IF A$="H" THEN 160
120 LET A=G
140 GO TO 20
160 LET B=G
180 GO TO 20
200 PRINT "I'M SMART!"
210 END
```

INITIAL INTERVAL

CALCULATE GUESS

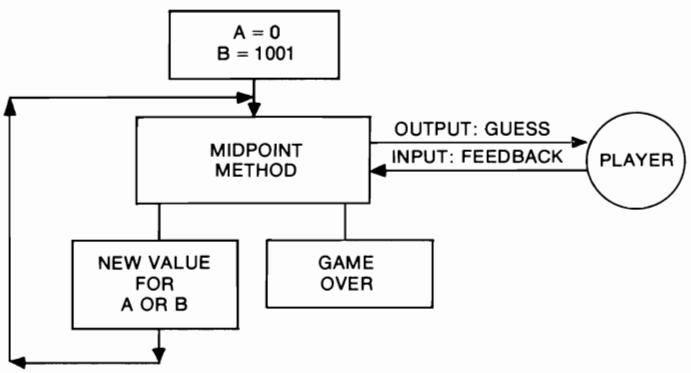
NEW LEFT END POINT

NEW RIGHT END POINT

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 500 ? H
I GUESS 250 ? H
I GUESS 125 ? L
I GUESS 187 ? L
I GUESS 218 ? H
I GUESS 202 ? L
I GUESS 210 ? L
I GUESS 214 ? H
I GUESS 212 ? R
I'M SMART!
```

OUTPUT

Here is a block diagram of COMPUTER GUESS/ MIDPOINT METHOD.



Compare this diagram with the block diagram following the program COMPUTER GUESS/RANDOM METHOD. The diagrams are the same. The programs are essentially the same. The differences occur in (1) the initialization of the end points of the interval, (2) the method of guessing a number, and (3) the assignment of new values to A or B:

	Random Method	Midpoint Method
Initialization of the End Points	A = 1 B = 1000	A = 0 B = 1001
Method of Guessing	G = INT((B − A + 1) * RND(X)) + 1	G = INT((A + B)/2)
Assignment of New End Point Values	A = G + 1 B = G − 1	A = G B = G
Random Statement	Necessary	Unnecessary

EXERCISE SET 3.5

On-line:

1. Take your program COM/G5, and modify it so that it uses the Midpoint Method for guessing a number. Play five games with your program. Insert in the following table the number of guesses needed by the computer for each game, and the average number of guesses needed for the five games. This new program will not be used again.

GAME#	#GUESSES
1	
2	
3	
4	
5	
AVERAGE	

3.6 WHICH IS THE BETTER STRATEGY: RANDOM OR MIDPOINT?

In exercise set 3.4 you played five games with the computer using your COMPUTER GUESS/RANDOM METHOD program and inserted the results in a table. In exercise set 3.5 you played another five games with the computer using your COMPUTER GUESS/MIDPOINT METHOD program. Which strategy is better? In this case, the better strategy is the one which allows you or the computer to hit upon the mystery number with the smaller number of guesses, on the average. Of course, a strategy which works better for you most of the time may occasionally require a very large number of guesses. We want to take a look at that characteristic of each strategy also.

Look at the results you obtained in the five games you played using each method. Which method required the smaller number of guesses, on the *average*? Which method required the largest number of guesses in any one game? The smallest number of guesses in any one game?

Unfortunately I can't look over your shoulder at your results, so let us both look at the results obtained by my

computer. When it used the Random Strategy, it followed a program just like the one I asked you to write in exercise set 3.4. When it used the Midpoint Strategy, it followed a program just like the one I asked you to write in exercise set 3.5. Here is my COMPUTER GUESS/RANDOM METHOD program.

COMPUTER GUESS/RANDOM METHOD

```
  1 RANDOM
 10 LET A=1
 15 LET B=1000
 16 PRINT "THINK OF A NUMBER FROM 1 TO 1000."
 17 LET C=0
 20 LET G=INT((B-A+1)*RND(X))+A
 21 LET C=C+1
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="R" THEN 200
100 IF A$="H" THEN 160
120 LET A=G+1
140 GO TO 20
160 LET B=G-1
180 GO TO 20
200 PRINT "I'M SMART!"
201 PRINT "I TOOK" C "GUESSES."
210 END
```

Here are five games played with the above program.

GAME NO. 1

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 9 ? L
I GUESS 600 ? H
I GUESS 375 ? H
I GUESS 369 ? H
I GUESS 126 ? H
I GUESS 20 ? L
I GUESS 86 ? H
I GUESS 80 ? H
I GUESS 21 ? L
I GUESS 56 ? L
I GUESS 77 ? H
I GUESS 66 ? H
I GUESS 63 ? H
I GUESS 59 ? H
I GUESS 58 ? R
I'M SMART!
I TOOK 15 GUESSES.
```

GAME NO. 2

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 601 ? H
I GUESS 223 ? L
I GUESS 486 ? L
I GUESS 511 ? L
I GUESS 512 ? L
I GUESS 526 ? H
I GUESS 523 ? H
I GUESS 517 ? L
I GUESS 520 ? H
I GUESS 518 ? R
I'M SMART!
I TOOK 10 GUESSES.
```

GAME NO. 3

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 307 ? L
I GUESS 647 ? H
I GUESS 325 ? L
I GUESS 414 ? L
I GUESS 461 ? H
I GUESS 448 ? H
I GUESS 431 ? L
I GUESS 441 ? H
I GUESS 440 ? H
I GUESS 435 ? H
I GUESS 433 ? H
I GUESS 432 ? R
I'M SMART!
I TOOK 12 GUESSES.
```

GAME NO. 4

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 667 ? L
I GUESS 856 ? L
I GUESS 898 ? H
I GUESS 897 ? H
I GUESS 869 ? H
I GUESS 857 ? L
I GUESS 863 ? L
I GUESS 866 ? L
I GUESS 868 ? R
I'M SMART!
I TOOK 9 GUESSES.
```

GAME NO. 5

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 757 ? H
I GUESS 635 ? H
I GUESS 224 ? H
I GUESS 41 ? L
I GUESS 208 ? L
I GUESS 222 ? H
I GUESS 221 ? H
I GUESS 219 ? H
I GUESS 218 ? H
I GUESS 210 ? L
I GUESS 214 ? H
I GUESS 212 ? L
I GUESS 213 ? R
I'M SMART!
I TOOK 13 GUESSES.
```

Here is a summary of the five games:

Game#	#Guesses
1	15
2	10
3	12
4	9
5	13
AVERAGE	$\frac{59}{5} = 11.8$

How does my table compare with the table you constructed in exercise set 3.4?

Referring to the table , what is the minimum, maximum and average number of guesses required by the computer to hit upon the mystery number? The minimum number in the table is nine. However, in theory we know the computer can hit upon the mystery number on its first guess. But this would not happen very often. The maximum number in the table is 15. However, just as it is possible for the computer to get lucky, it could also be extremely unlucky. Suppose I picked 1 as the number for the computer to guess. The computer's first guess could be 1000 which is too high. The next guess could be 999 which is again too high. This could be followed by a third guess of 998. But such a sequence of guesses which begins at 1000 and decreases by 1 each time seems highly unlikely. What is the average number of guesses per game according to the table? The total number of guesses in the five games was 59. Divide this by the number of games and the average is 11.8 guesses per game. The five sample runs of COMPUTER GUESS/RANDOM METHOD can be summarized as follows:

Minimum Number of Guesses:	9
Maximum Number of Guesses:	15
Average:	11.8

So much for the Random Method. Now look at COMPUTER GUESS/MIDPOINT METHOD. Here is my program.

COMPUTER GUESS/MIDPOINT METHOD

```
 10 LET A=0
 15 LET B=1001
 16 PRINT "THINK OF A NUMBER FROM 1 TO 1000."
 17 LET C=0
 20 LET G=INT((A+B)/2)
 21 LET C=C+1
 40 PRINT "I GUESS" G;
 60 INPUT A$
 80 IF A$="R" THEN 200
100 IF A$="H" THEN 160
120 LET A=G
140 GO TO 20
160 LET B=G
180 GO TO 20
200 PRINT "I'M SMART!"
201 PRINT "I TOOK" C "GUESSES."
210 END
```

Here are five games played with the above program. To make the results comparable, I picked the same five numbers, 58, 518, 432, 868, and 213, that I used in my five games with COMPUTER GUESS/RANDOM METHOD.

GAME NO. 1

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 500 ? H
I GUESS 250 ? H
I GUESS 125 ? H
I GUESS 62 ? H
I GUESS 31 ? L
I GUESS 46 ? L
I GUESS 54 ? L
I GUESS 58 ? R
I'M SMART!
I TOOK 8 GUESSES.
```

GAME NO. 2

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 500 ? L
I GUESS 750 ? H
I GUESS 625 ? H
I GUESS 562 ? H
I GUESS 531 ? H
I GUESS 515 ? L
I GUESS 523 ? H
I GUESS 519 ? H
I GUESS 517 ? L
I GUESS 518 ? R
I'M SMART!
I TOOK 10 GUESSES.
```

GAME NO. 3

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 500 ? H
I GUESS 250 ? L
I GUESS 375 ? L
I GUESS 437 ? H
I GUESS 406 ? L
I GUESS 421 ? L
I GUESS 429 ? L
I GUESS 433 ? H
I GUESS 431 ? L
I GUESS 432 ? R
I'M SMART!
I TOOK 10 GUESSES.
```

GAME NO. 4

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 500 ? L
I GUESS 750 ? L
I GUESS 875 ? H
I GUESS 812 ? L
I GUESS 843 ? L
I GUESS 859 ? L
I GUESS 867 ? L
I GUESS 871 ? H
I GUESS 869 ? H
I GUESS 868 ? R
I'M SMART!
I TOOK 10 GUESSES.
```

GAME NO. 5

```
THINK OF A NUMBER FROM 1 TO 1000.
I GUESS 500 ? H
I GUESS 250 ? H
I GUESS 125 ? L
I GUESS 187 ? L
I GUESS 218 ? H
I GUESS 202 ? L
I GUESS 210 ? L
I GUESS 214 ? H
I GUESS 212 ? L
I GUESS 213 ? R
I'M SMART!
I TOOK 10 GUESSES.
```

Here is a summary of the five games:

Game #	#Guesses
1	8
2	10
3	10
4	10
5	10
AVERAGE	$\frac{48}{5} = 9.6$

The minimum, maximum and average number of guesses required by the computer to hit upon the mystery number are:

Minimum Number of Guesses:	8
Maximum Number of Guesses:	10
Average:	9.6

Let's put the two summaries side by side.

	Random Method	Midpoint Method
Minimum Number of Guesses:	9	8
Maximum Number of Guesses:	15	10
Average:	11.8	9.6

Which is the better strategy? Based on the data gathered from the ten sample runs, the Midpoint Method looks like the winner. Can we be sure? Suppose I had picked numbers other than 58, 518, 432, 868, and 213 for the computer to guess. Would the results be about the same? Will the Random Method ever guess a number in one try? Will it ever take 1000 guesses? Will the Midpoint Method ever guess a number in one try? Will it ever take more than ten guesses? Are my averages reliable? To answer these questions or to get more reliable estimates, we need to play a large number of games, say 1000, using each method. You can do this one of two ways. You can sit at your terminal and play one game every two minutes. Would you believe this comes to about 33 hours for each method? On the other hand you can take advantage of the speed of the computer, by programming the computer not only to guess the mystery number, but to think of it in the first place! Starting first with the Random Method, let's give the computer this dual role.

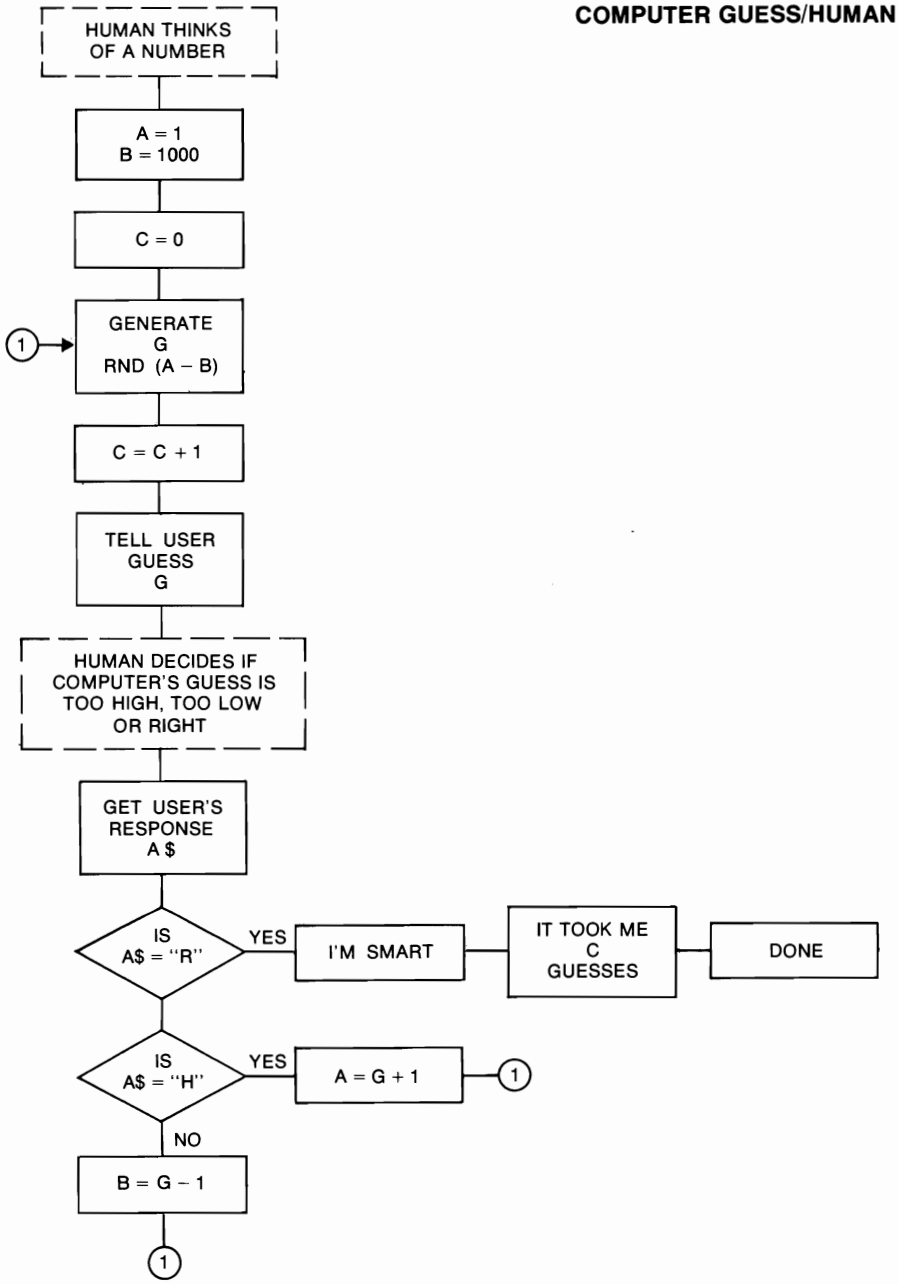
3.6.1 THE RANDOM METHOD: 1000 GAMES

The human operations in playing the COMPUTER GUESS/RANDOM METHOD game are:

1. Think of a number between 1 and 1000 and remember it.
2. Inform the computer whether its guess is too high, too low, or right on.

We can program the computer to perform both these "human" tasks for us. The following flow chart for COMPUTER GUESS/RANDOM METHOD has been modified to show the two human operations in dashed boxes.

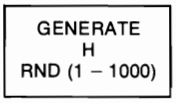
COMPUTER GUESS/HUMAN OPERATIONS FLOW CHART



Let's program the first human task:

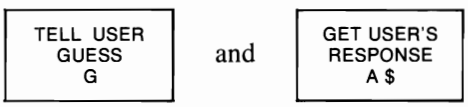
In real life the human picks his mystery number "at random" between 1 and 1000. The computer can easily

do this by generating a number at random between 1 and 1000. The dashed box can be replaced by:

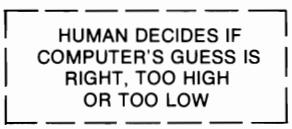


The next step is to program the computer to perform the second human task: decide if the computer's guess is too high, too low, or right on. This "human" decision is made after the computer types out its guess.

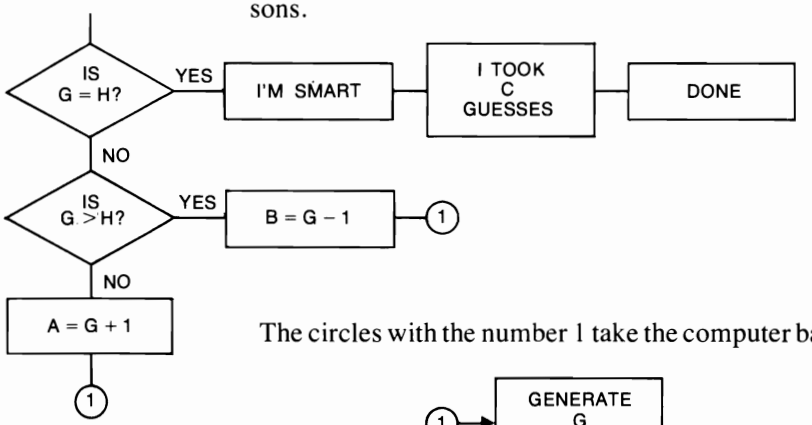
Since the computer will make this decision instead of a human, there is no need to have the guess typed out, and no need to have the computer get the "human" response: too high, too low, or right on.



can be eliminated. Having crossed out those two boxes, let's expand into a flow chart the "human" task:



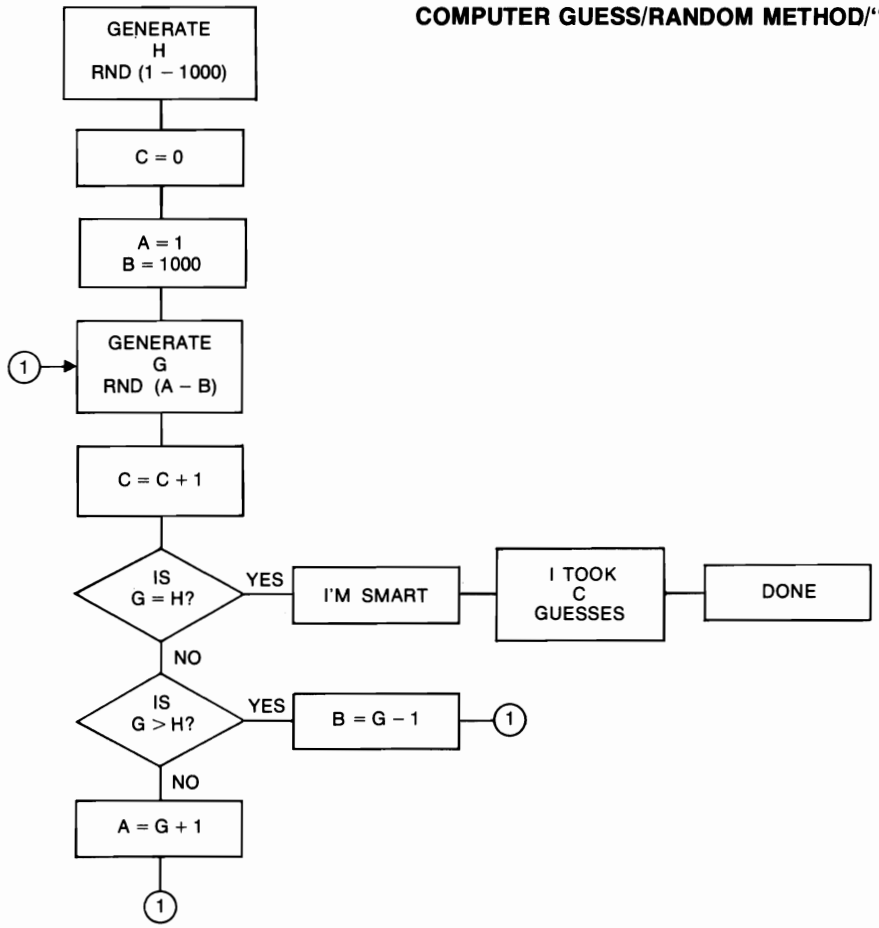
The first step is to test whether the computer hit upon the mystery number. That is, DOES G = H? If G does equal H, then the computer is on target and the game is over. If G does not equal H, then two possibilities exist. If the computer's number is greater than the human's, G > H, the computer's guess is too high and the right hand end point of the interval, B, must be moved one unit to the left of G by saying: LET B = G-1. If G is not equal to H and is not greater than M, the third possibility must follow, namely G<H. In this case, the computer's guess is too low and the left end point of the interval, A, must be moved one unit to the right by saying: LET A = G+1. Let's flowchart these comparisons.



The circles with the number 1 take the computer back to

So much for the human tasks. Here is the revised flow chart.

COMPUTER GUESS/RANDOM METHOD/"DEHUMANIZE" FLOW CHART



The following program is encoded from the above flow chart.

PROGRAM

```
  1 RANDOM
 10 LET H=INT(1000*RND(X))+1 <-- PICK NUMBER FOR HUMAN
 20 LET C=0
 30 LET A=1
 40 LET B=1000
 50 LET G=INT((B-A+1)*RND(X))+A
 60 LET C=C+1
 70 IF G=H THEN 130
 80 IF G>H THEN 110
 90 LET A=G+1 <-------------- IF G < H
100 GO TO 50
110 LET B=G-1 <-------------- IF G > H
120 GO TO 50
130 PRINT "I'M SMART!" <---------- IF G = H
140 PRINT "I TOOK" C "GUESSES."
150 END
```

Here is a typical run:

OUTPUT

```
I'M SMART!
I TOOK 10 GUESSES.
```

Does the program operate as we planned? The output doesn't tell us. To take a look behind the scenes, let's

insert two "see through" print statements. One will tell us the mystery number picked by the "human": 15 PRINT "HUMAN PICKED" H. The other will tell us the computer's guesses: 55 PRINT "I GUESS" G. Here is a typical run of the "see through" version of the program.

COMPUTER GUESS/SEE THROUGH

```
 1 RANDOM
10 LET H=INT(1000*RND(X))+1
15 PRINT "HUMAN PICKED" H
20 LET C=0
30 LET A=1
40 LET B=1000
50 LET G=INT((B-A+1)*RND(X))+A
55 PRINT "I GUESS" G
60 LET C=C+1
70 IF G=H THEN 130
80 IF G>H THEN 110
90 LET A=G+1
100 GO TO 50
110 LET B=G-1
120 GO TO 50
130 PRINT "I'M SMART!"
140 PRINT "I TOOK" C "GUESSES."
150 END
```

← PRINT OUT COMPUTER'S GUESS

← PRINT OUT HUMAN NUMBER

```
HUMAN PICKED 988
I GUESS 532
I GUESS 615
I GUESS 867
I GUESS 912
I GUESS 923
I GUESS 978
I GUESS 982
I GUESS 997
I GUESS 996
I GUESS 986
I GUESS 990
I GUESS 987
I GUESS 988
I'M SMART!
I TOOK 13 GUESSES.
```

Convinced that the program operates satisfactorily, let's have the computer play the game as many times as we wish. We will specify the number of games, N, to be played after we say RUN. To do this, we will put the program, COMPUTER GUESS/RANDOM METHOD/"DEHUMANIZED" in a loop using FOR and NEXT. P will count off the number of games played. Furthermore, since we're now interested in having the computer play many games as quickly as possible, we'll omit the PRINT statements which tell us the mystery number for each game, the computer's guesses, and the fact that the computer is smart.

COMPUTER GUESS/ RANDOM METHOD/N GAMES

N GAMES LOOP

```
 1 RANDOM
 7 PRINT "HOW MANY GAMES";
 8 INPUT N
 9 FOR P=1 TO N
10 LET H=INT(1000*RND(X))+1
20 LET C=0
30 LET A=1
40 LET B=1000
50 LET G=INT((B-A+1)*RND(X))+A
60 LET C=C+1
70 IF G=H THEN 140
80 IF G>H THEN 110
90 LET A=G+1
100 GO TO 50
110 LET B=G-1
120 GO TO 50
140 PRINT "I TOOK" C "GUESSES."
145 NEXT P
150 END
```

```
HOW MANY GAMES? 20
I TOOK 9 GUESSES.
I TOOK 7 GUESSES.
I TOOK 14 GUESSES.
I TOOK 11 GUESSES.
I TOOK 8 GUESSES.
I TOOK 11 GUESSES.
I TOOK 10 GUESSES.
I TOOK 14 GUESSES.
I TOOK 12 GUESSES.
I TOOK 9 GUESSES.
I TOOK 16 GUESSES.
I TOOK 13 GUESSES.
I TOOK 12 GUESSES.
I TOOK 10 GUESSES.
I TOOK 16 GUESSES.
I TOOK 14 GUESSES.
I TOOK 19 GUESSES.
I TOOK 13 GUESSES.
I TOOK 14 GUESSES.
I TOOK 12 GUESSES.
```

We're interested in the results, not of each game, but of many games. That is, we're interested in how many games it took one guess to hit upon the mystery number, in how many games it took two guesses, in how many games it took three guesses, etc. In short we're interested in the frequency of games that took one, two, three, four, five, etc. guesses. If you were keeping track of the frequency by hand, you would set up a tally list which would look something like this:

Number of Guesses	Frequency
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

To set up a "tally list" in the computer, we use a subscripted variable. At the beginning of this chapter we used a subscripted variable to set up a check list to avoid duplicate guesses. A DIM statement was needed to set aside a certain number of boxes. How many boxes must we now set aside for *this* tally list? Certainly the minimum number of guesses is 1 (the computer might get lucky). To this point the maximum number has been 19 (look at the last output), so 25 boxes ought to be enough. If it isn't, we can change our upper limit to 30 or 40. We'll use the variable F for frequency and subscript it as: F(1), F(2), F(3), . . . , F(25). In box form the "tally list" looks like this:

LABEL	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)	F(9)	F(10)
CONTENTS	0	0	0	0	0	0	0	0	0	0

LABEL	F(11)	F(12)	F(13)	F(14)	F(15)	F(16)	F(17)	F(18)	F(19)	F(20)
CONTENTS	0	0	0	0	0	0	0	0	0	0

F(21)	F(22)	F(23)	F(24)	F(25)
0	0	0	0	0

Each time a game is played, the computer will find the box which represents the number of guesses necessary to hit upon the mystery number and add one (1) to the contents. When 30 games have been played, the computer's "tally list" might look like this:

F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)	F(9)	F(10)
0	1	0	0	3	0	1	0	2	1

F(11)	F(12)	F(13)	F(14)	F(15)	F(16)	F(17)	F(18)	F(19)	F(20)
2	4	5	3	2	1	2	0	1	1

F(21)	F(22)	F(23)	F(24)	F(25)
0	1	0	0	0

If in the thirty-first game 13 guesses are required, the computer adds 1 to box F(13). In BASIC we write LET F(13) = F(13) + 1, which means that the new value of F(13) is equal to the old value of F(13), 5, plus 1, or 6. Thus, after each game is played, the computer executes this statement:

LET F(C) = F(C) + 1

in which C is the number of guesses needed to hit upon the mystery number.

Let's fix up the old program COMPUTER GUESS/RANDOM METHOD/N GAMES. First, let's put in the tally list. Three things are required.

(1) A DIMENSION statement to set aside space in the computer's memory for the 25 boxes: 2 DIM F(25).

(2) An initializing loop to put a zero in each box. Add the following initializing loop:

```
3 FOR I = 1 TO 25
4 LET F(I) = 0
5 NEXT I
```

(3) A tally statement LET F(C) = F(C) + 1 to be executed after each game is played. The ideal position for this statement is the line in the old program occupied by the statement 140 PRINT "I TOOK" C "GUESSES." Since we don't need to print the number of guesses anyway, remove the PRINT statement and insert 140 LET F(C) = F(C) + 1.

Finally, to print out the frequency table after the 1000

games are played, add this loop:

```
150 PRINT "GUESSES," "FREQUENCY"
155 FOR I = 1 TO 25
156 PRINT I, F(I)
157 NEXT I
```

Here is the new program COMPUTER GUESS/ RANDOM METHOD/RANDOM.

COMPUTER GUESS/ RANDOM METHOD/RANDOM

PROGRAM

```
 1 RANDOM
 2 DIM F(25)
 3 FOR I=1 TO 25
 4 LET F(I)=0
 5 NEXT I
 7 PRINT "HOW MANY GAMES";
 8 INPUT N
 9 FOR P=1 TO N
10 LET H=INT(1000*RND(X))+1
20 LET C=0
30 LET A=1
40 LET B=1000
50 LET G=INT((B-A+1)*RND(X))+A
60 LET C=C+1
70 IF G=H THEN 140
80 IF G>H THEN 110
90 LET A=G+1
100 GO TO 50
110 LET B=G-1
120 GO TO 50
140 LET F(C)=F(C)+1
145 NEXT P
150 PRINT "GUESSES","FREQUENCY"
155 FOR I=1 TO 25
156 PRINT I,F(I)
157 NEXT I
160 END
```

INITIALIZE TALLY LIST

N GAMES LOOP

TALLY STATEMENT

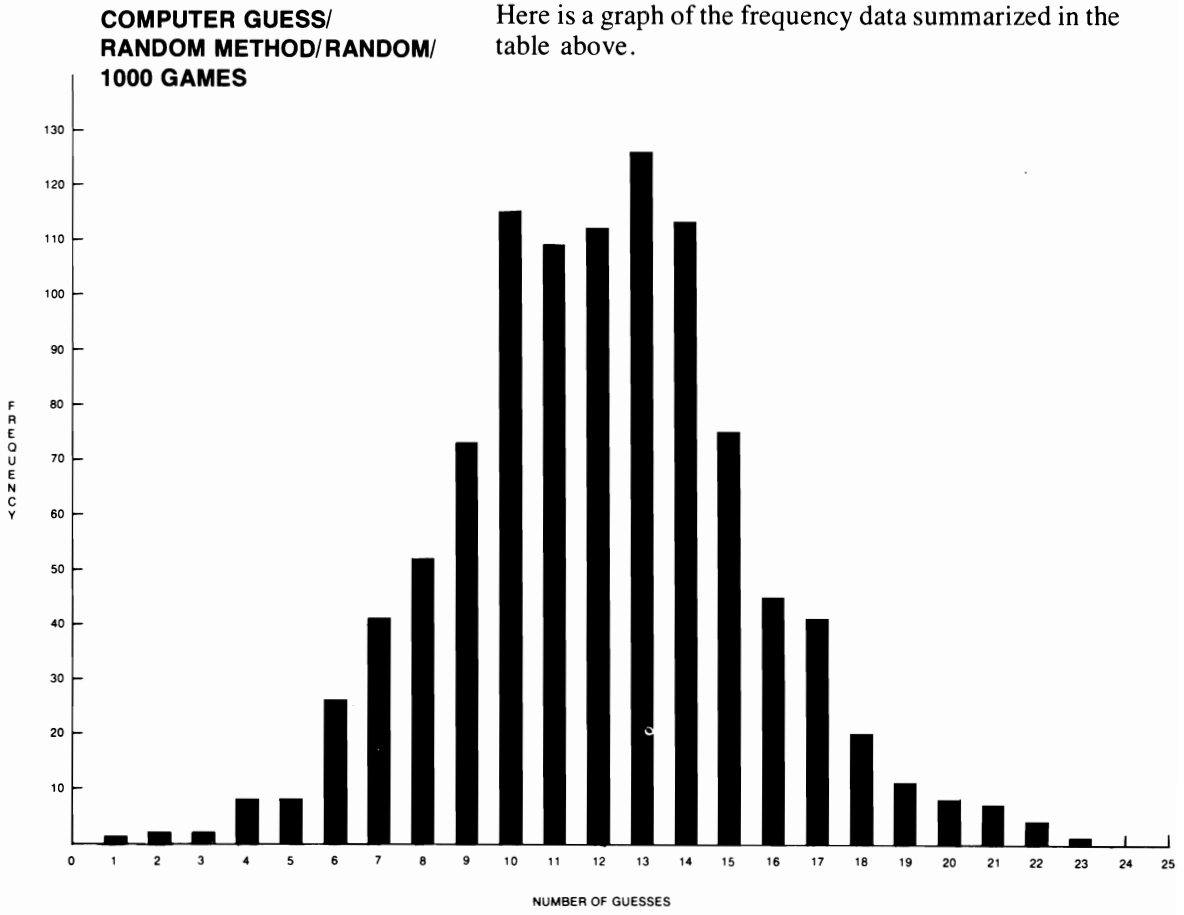
COLUMN HEADING

PRINT OUT FREQUENCY TABLE

A typical run requesting the computer to play 1000 games is to the right.

OUTPUT

```
HOW MANY GAMES? 1000
 GUESSES        FREQUENCY
 1              1
 2              2
 3              2
 4              8
 5              8
 6              26
 7              41
 8              52
 9              73
 10             115
 11             109
 12             112
 13             126
 14             113
 15             75
 16             45
 17             41
 18             20
 19             11
 20             8
 21             7
 22             4
 23             1
 24             0
 25             0
```



Here is a graph of the frequency data summarized in the table above.

The speed of the computer is fantastic. The actual run time of this program on my computer (a time-sharing system) was about 10 minutes, a marked difference from the length of time, 33 hours, it would take me to play 1000 games of COMPUTER GUESS/RANDOM METHOD one at a time. Here is what we learn from the above output:

(1) Minimum Number. One guess was enough to hit upon the number in one game out of 1000. Two games took two guesses, and two games took three guesses. In short, it's possible for the computer to be lucky, but not very often.

(2) Maximum Number. In one game the number of guesses required to hit upon the number was 23. That's the maximum. Four games took 22 guesses.

(3) Average number. If you were to multiply each number in the first column by its corresponding frequency in the second column, and divide the total by 1000, you would obtain 12.003 as the average number of guesses.

To sum up:

Minimum Number of Guesses:	1
Maximum Number of Guesses:	23
Average Number of Guesses:	12.003

In the program COMPUTER GUESS/RANDOM METHOD/RANDOM, the mystery number picked by the "human player" was generated at random from the interval 1 to 1000. As a result some numbers were picked many times while others were never picked at all. The "human" could, however, pick his mystery number in such a way as to insure that, in 1000 games, each number from 1 to 1000 was used once.

To program the computer to act similarly, replace the statement 10 LET H = INT(1000*RND(X)+1 with the new statement 10 FOR H = 1 TO 1000. As H goes from 1 to 1000, it represents not only the number picked by the "human," but also the number of games played to that point by the computer. The old FOR and NEXT statements, 9 FOR P = 1 TO N and 145 NEXT P, no longer needed, may be deleted. The statement 7 PRINT "HOW MANY GAMES"; and 8 INPUT N can also be deleted since the statement FOR H = 1 TO 1000 controls the number of games.

To sum up, we delete:

```
  7 PRINT "HOW MANY RUNS";
  8 INPUT N
  9 FOR P = 1 TO N
 10 LET H = INT(1000*RND(X))+1
145 NEXT P
```

And add:

```
 10 FOR H = 1 TO 1000
145 NEXT H
```

We also want to have the computer find the average number of guesses required in 1000 games. This is done by obtaining the total number of guesses and dividing by 1000. In each individual game the computer generates its guesses by executing line 50 one or more times. While the present counter, C, in line 60, counts the number of guesses generated in each game by line 50, it cannot keep a running total for all games because at the beginning of each game, C is set back to zero by line 20. To count the total number of times line 50 is executed in 1000 games, insert

```
55 LET T = T+1.
```

Initialize the new counter by inserting before the game loop

```
6 LET T = 0
```

Finally, insert

```
158 PRINT "THE AVERAGE IS" T/1000
```

to type out the average number of guesses after the frequency table is listed.

**COMPUTER GUESS/
RANDOM METHOD/SERIAL**

PROGRAM

```
 1 RANDOM
 2 DIM F(25)
 3 FOR I=1 TO 25
 4 LET F(I)=0
 5 NEXT I
 6 LET T=0          <-- INITIALIZE NEW COUNTER
10 FOR H=1 TO 1000
20 LET C=0
30 LET A=1
40 LET B=1000
50 LET G=INT((B-A+1)*RND(X))+A
55 LET T=T+1        <-- TOTAL NUMBER GUESSES
60 LET C=C+1
70 IF G=H THEN 140
80 IF G>H THEN 110
90 LET A=G+1
100 GO TO 50
110 LET B=G-1
120 GO TO 50
140 LET F(C)=F(C)+1
145 NEXT H
150 PRINT "GUESSES","FREQUENCY"
155 FOR I=1 TO 25
156 PRINT I,F(I)
157 NEXT I
158 PRINT "THE AVERAGE IS" T/1000   <-- PRINT OUT AVERAGE
160 END
```

H LOOP

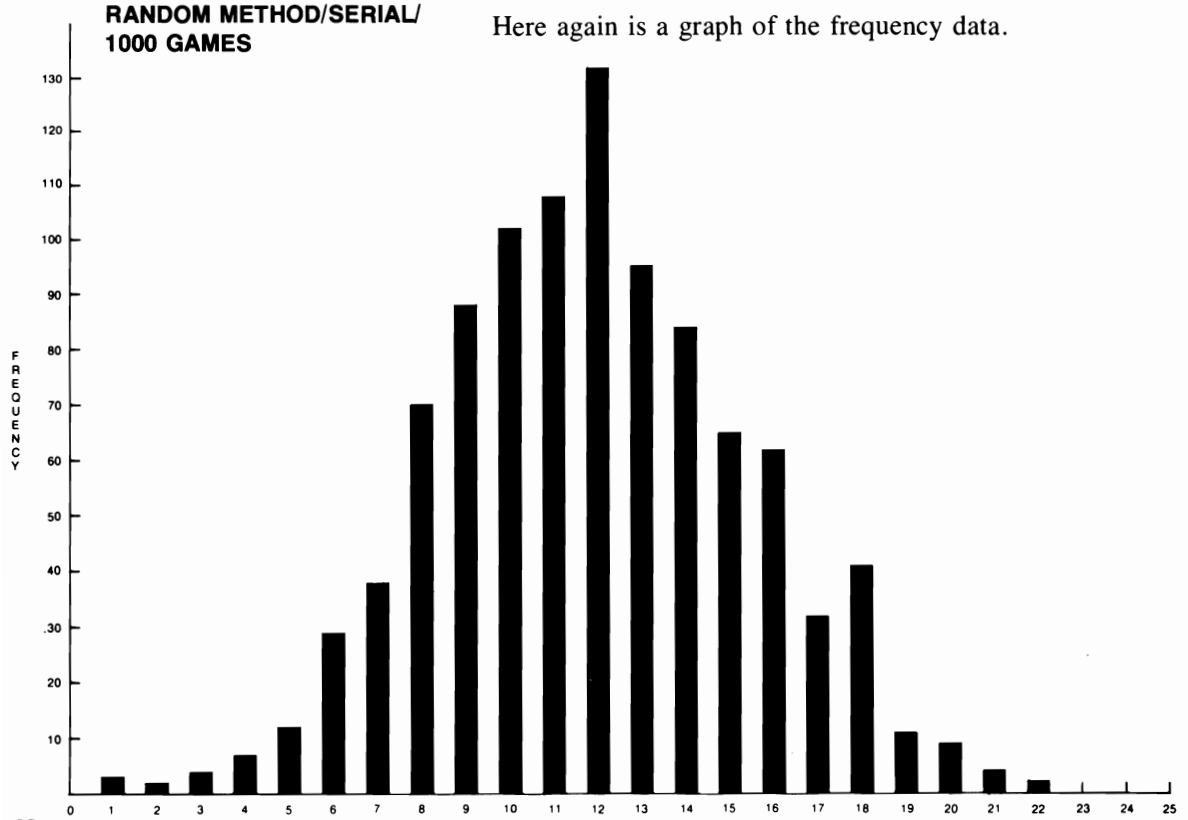
OUTPUT

GUESSES	FREQUENCY
1	3
2	2
3	4
4	7
5	12
6	29
7	38
8	70
9	88
10	102
11	108
12	132
13	95
14	84
15	65
16	62
17	32
18	41
19	11
20	9
21	4
22	2
23	0
24	0
25	0

THE AVERAGE IS 11.868

**COMPUTER GUESS/
RANDOM METHOD/SERIAL/
1000 GAMES**

Here again is a graph of the frequency data.



Now look back at the output. Here is a table comparing the results obtained when the number was picked at random with the results obtained when each number from 1 to 1000 was picked once.

	Random Method/ Random	Random Method/ Serial
Minimum Number Guesses:	1	1
Maximum Number Guesses:	23	22
Average Number Guesses:	12.003	11.868

It would appear that the method by which the computer picks a number for the "human" makes little difference in the overall results.

EXERCISE SET 3.6.1

On-line:

1. Assume the human thinks of a mystery number between 1 and 200. What is the minimum, the maximum, and the average number of guesses needed by the computer to hit upon the mystery number using the Random Method? To do this,

   A) Take the program COMPUTER GUESS/ RANDOM METHOD/RANDOM and

      (1) change the interval from which the "human" picks a number from 1 to 1000 to 1 to 200, and
      (2) add steps to find the total number of guesses and the average. Now run this program. Save the program as COM/G6 for modification in section 3.6.2.

   B) Graph the output produced by the above program.
   C) Modify the above program so the computer picks each and every number from 1 to 200. Run this program.
   D) Graph the output produced by the above program.
   E) Fill in the following table and save it for the next section.

	Random Method/ Random	Random Method/ Serial
Minimum Number Guesses:		
Maximum Number Guesses:		
Average Number Guesses:		

3.6.2 THE MIDPOINT METHOD: 1000 GAMES

So much for the minimum, maximum, and average number of guesses per game using the Random Method. Let's gather the same statistics for the Midpoint

Method. To do this we need to write a program like COMPUTER GUESS/RANDOM METHOD/ RANDOM, substituting the Midpoint Method for the Random Method. Look at the block diagrams of the Random Method and the Midpoint Method in sections 3.4 and 3.5. Here is a summary of the differences between the two methods.

	Random Method	Midpoint Method
Initialization of the End Points	A = 1 B = 1000	A= 0 B = 1001
Method of Guessing	G=INT((B−A+1)*RND(X))+1	G=INT((A+B)/2)
Assignment of New End Point Values	A = G + 1 B = G − 1	A = G B = G
RANDOM statement	Necessary	Unnecessary

Since the programs for the Random and Midpoint Methods are identical in structure, we need to make only six changes in the program COMPUTER GUESS/RANDOM METHOD/RANDOM to obtain the new program COMPUTER GUESS/MIDPOINT METHOD/RANDOM. Here is a copy of COMPUTER GUESS/RANDOM METHOD/RANDOM showing the changes to be made.

COMPUTER GUESS/ RANDOM METHOD/RANDOM

```
  1 RANDOM ◄──────────────────────── DELETE
  2 DIM F(25)
  3 FOR I=1 TO 25
  4 LET F(I)=0
  5 NEXT I
  7 PRINT "HOW MANY GAMES";
  8 INPUT N
  9 FOR P=1 TO N
 10 LET H=INT(1000*RND(X))+1
 20 LET C=0
 30 LET A=1 ◄──────────────────── REPLACE WITH: 30 LET A = 0
 40 LET B=1000 ◄───────────────── REPLACE WITH: 40 LET B = 1001
 50 LET G=INT((B-A+1)*RND(X))+A ◄ REPLACE WITH: 50 LET G = INT
                                                ((A + B)/2)
 60 LET C=C+1
 70 IF G=H THEN 140
 80 IF G>H THEN 110
 90 LET A=G+1 ◄────────────────── REPLACE WITH: 90 LET A = G
100 GO TO 50
110 LET B=G-1 ◄────────────────── REPLACE WITH: 110 LET B = G
120 GO TO 50
140 LET F(C)=F(C)+1
145 NEXT P
150 PRINT "GUESSES","FREQUENCY"
155 FOR I=1 TO 25
156 PRINT I,F(I)
157 NEXT I
160 END
```

As in COMPUTER GUESS/RANDOM METHOD/ SERIAL, we want the computer to calculate the average number of guesses. To count the total number of guesses, add to the above program the new counter, T:

6 LET T = 0
55 LET T = T+1

To divide T by N, the number of games played, add:

158 PRINT "THE AVERAGE IS" T/N

Here is the new program.

COMPUTER GUESS/ MIDPOINT METHOD/RANDOM

PROGRAM

```
  2 DIM F(25)
  3 FOR I=1 TO 25
  4 LET F(I)=0
  5 NEXT I
→ 6 LET T=0
  7 PRINT "HOW MANY GAMES";
  8 INPUT N
  9 FOR P=1 TO N
 10 LET H=INT(1000*RND(X))+1
 20 LET C=0
 30 LET A=0 ┐
 40 LET B=1001 ┘ ← INITIAL INTERVAL
 50 LET G=INT((A+B)/2) ← CALCULATE GUESS
→55 LET T=T+1
 60 LET C=C+1
 70 IF G=H THEN 140
 80 IF G>H THEN 110
 90 LET A=G ← NEW LEFT END POINT
100 GO TO 50
110 LET B=G ← NEW RIGHT END POINT
120 GO TO 50
140 LET F(C)=F(C)+1
145 NEXT P
150 PRINT "GUESSES","FREQUENCY"
155 FOR I=1 TO 25
156 PRINT I,F(I)
157 NEXT I
→158 PRINT "THE AVERAGE IS" T/N
160 END
```

A typical run of this program is given to the right.

OUTPUT

```
HOW MANY GAMES? 1000
GUESSES        FREQUENCY
 1              1
 2              1
 3              2
 4              11
 5              15
 6              27
 7              69
 8              125
 9              258
 10             491
 11             0
 12             0
 13             0
 14             0
 15             0
 16             0
 17             0
 18             0
 19             0
 20             0
 21             0
 22             0
 23             0
 24             0
 25             0
THE AVERAGE IS 9.005
```

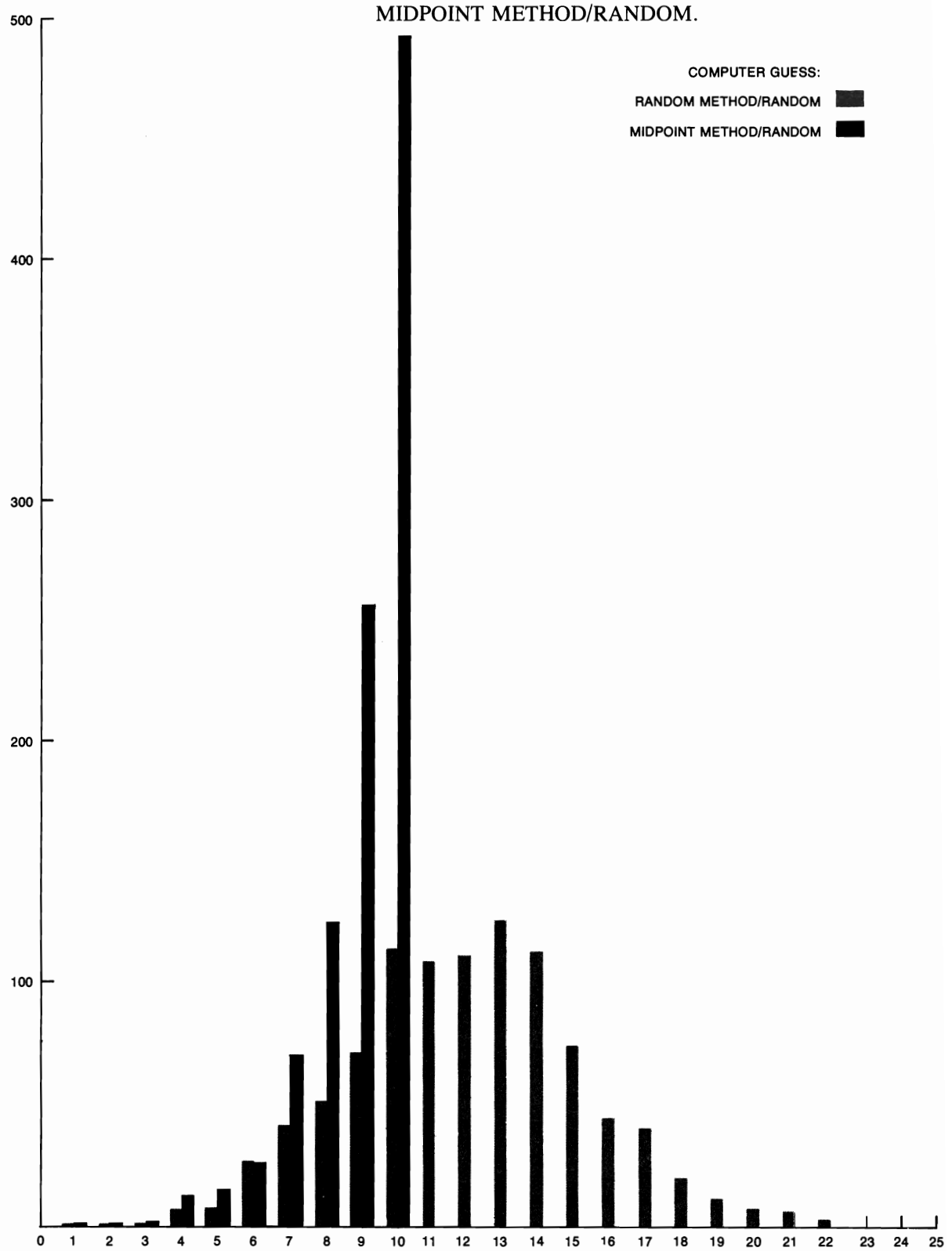
Look at the output. What is the minimum, maximum, and average number of guesses per game needed by the computer to hit upon the user's number? How do these statistics compare with those obtained with COMPUTER GUESS/RANDOM METHOD/RANDOM? Here is a table which brings these results together.

	Random Method/ Random	Midpoint Method/ Random
Minimum Number Guesses:	1	1
Average Number Guesses:	12.003	9.005
Maximum Number Guesses:	23	10

Which method is more efficient? The Midpoint Method

COMPUTER GUESS:
RANDOM METHOD/RANDOM/1000 GAMES
MIDPOINT METHOD/RANDOM/1000 GAMES

is the better strategy. Here is a graph comparing the outputs generated by COMPUTER GUESS/RANDOM METHOD/RANDOM and COMPUTER GUESS/ MIDPOINT METHOD/RANDOM.



Look at the last output again. With the Midpoint Method, the maximum number of guesses was 10. But in that game, the computer generated the mystery number at random. It did not pick *all* the numbers between 1 and 1000. Can we be sure that, when using the Midpoint Method, the computer will never need more than ten guesses to hit upon any number the "human" thinks of between 1 and 1000? To find out, let's revise the game so that the "human" picks each number from 1 to 1000. The easy way to do this is to make six changes in the old program COMPUTER GUESS/RANDOM METHOD/SERIAL to obtain a new program COMPUTER GUESS/MIDPOINT METHOD/SERIAL. Here is a copy of COMPUTER GUESS/RANDOM METHOD/SERIAL showing the changes to be made:

COMPUTER GUESS/RANDOM METHOD/SERIAL

```
  1 RANDOM <------------------------------ DELETE
  2 DIM F(25)
  3 FOR I=1 TO 25
  4 LET F(I)=0
  5 NEXT I
  9 LET T=0
 10 FOR H=1 TO 1000
 20 LET C=0
 30 LET A=1 <---------------------------- REPLACE WITH: 30 LET A = 0
 40 LET B=1000 <------------------------- REPLACE WITH: 40 LET B = 1001
 50 LET G=INT((B-A+1)*RND(X))+A <-------- REPLACE WITH: 50 LET G = INT ( (A + B)/2)
 55 LET T=T+1
 60 LET C=C+1
 70 IF G=H THEN 140
 80 IF G>H THEN 110
 90 LET A=G+1 <-------------------------- REPLACE WITH: 90 LET A = G
100 GO TO 50
110 LET B=G-1 <------------------------- REPLACE WITH: 110 LET B = G
120 GO TO 50
140 LET F(C)=F(C)+1
145 NEXT H
150 PRINT "GUESSES","FREQUENCY"
155 FOR I=1 TO 25
156 PRINT I,F(I)
157 NEXT I
158 PRINT "THE AVERAGE IS" T/1000
160 END
```

Here is the new program COMPUTER GUESS/ MIDPOINT METHOD/SERIAL.

COMPUTER GUESS/ MIDPOINT METHOD/SERIAL

PROGRAM

```
  2 DIM F(25)
  3 FOR I=1 TO 25
  4 LET F(I)=0
  5 NEXT I
  6 LET T=0
 10 FOR H=1 TO 1000
 20 LET C=0
 30 LET A=0 ┐
 40 LET B=1001 ┘ ← INITIAL INTERVAL
 50 LET G=INT((A+B)/2) ← CALCULATE GUESS
 55 LET T=T+1
 60 LET C=C+1
 70 IF G=H THEN 140
 80 IF G>H THEN 110
 90 LET A=G ← NEW LEFT END POINT
100 GO TO 50
110 LET B=G ← NEW RIGHT END POINT
120 GO TO 50
140 LET F(C)=F(C)+1
145 NEXT H
150 PRINT "GUESSES","FREQUENCY"
155 FOR I=1 TO 25
156 PRINT I,F(I)
157 NEXT I
158 PRINT "THE AVERAGE IS" T/1000
160 END
```

OUTPUT

```
 GUESSES          FREQUENCY
  1                1
  2                2
  3                4
  4                8
  5                16
  6                32
  7                64
  8                128
  9                256
  10               489
  11               0
  12               0
  13               0
  14               0
  15               0
  16               0
  17               0
  18               0
  19               0
  20               0
  21               0
  22               0
  23               0
  24               0
  25               0
THE AVERAGE IS 8.987
```

No game took more than 10 guesses. Thus we can assert that the Midpoint Method will always hit upon any number the user is thinking of between 1 and 1000 in ten or fewer guesses.

EXERCISE SET 3.6.2

On-line:

1. Assume the human thinks of a number between 1 and 200. What is the minimum, maximum, and average number of guesses needed by the computer to hit upon the human's number using the Midpoint Method? To do this,
   A) Take the program, COM/G6, which you saved in the last exercise set and make the six changes necessary to transform it into a program which uses the Midpoint Method. Run this program. Graph the output.
   B) Modify the above program so the computer picks each and every number from 1 to 200. Run this program. Graph the output.

C) Summarize the results obtained in A and B in this table:

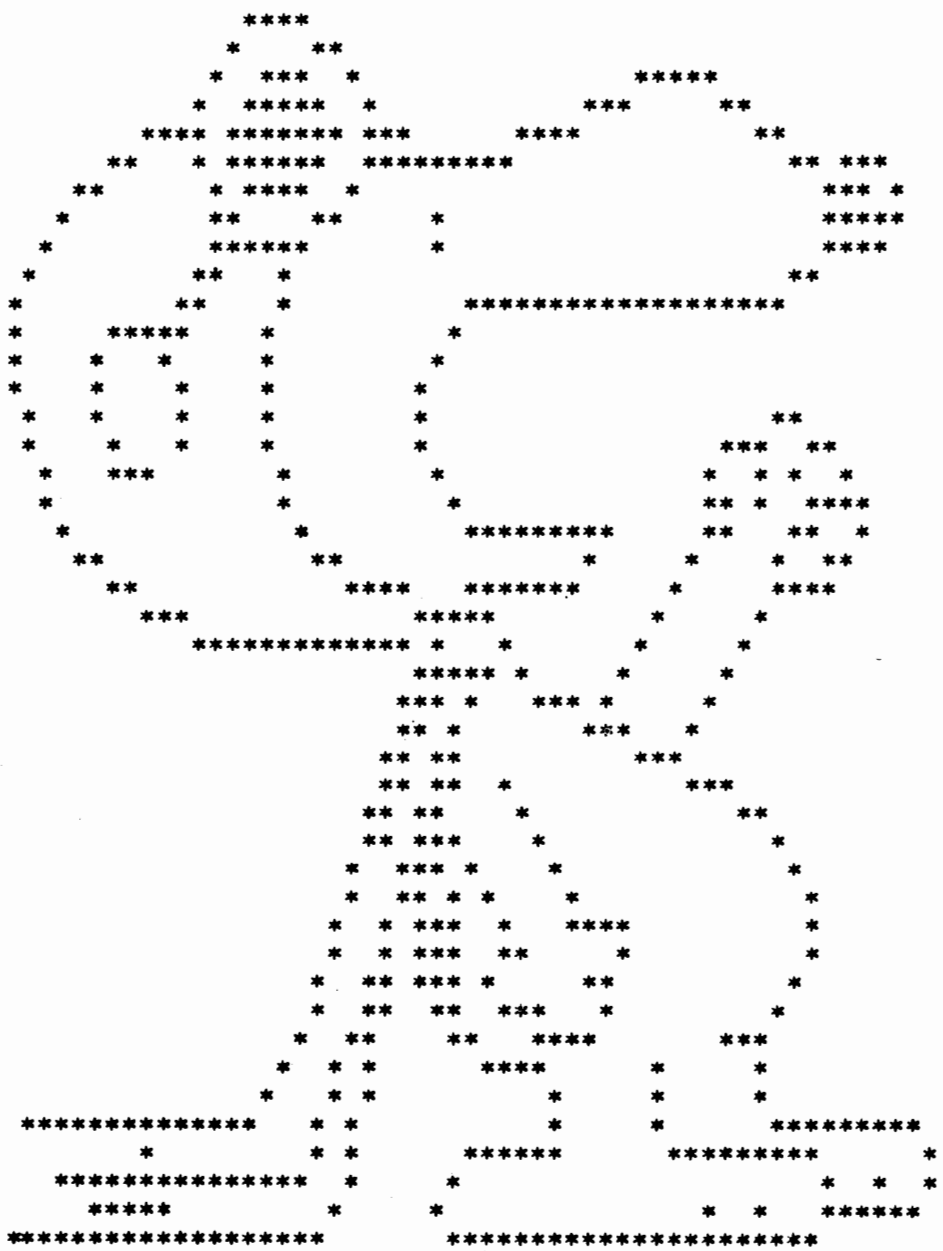
	Midpoint Method/ Random	Midpoint Method/ Serial
Minimum Number Guesses:		
Maximum Number Guesses:		
Average Number Guesses:		

D) Use the table below to compare the results summarized in the above table with the results listed in a similar table which you completed in set 3.6.1

	Random Method/ Random	Midpoint Method/ Random
Minimum Number Gueses:		
Maximum Number Guesses:		
Average Number Guesses:		

Which strategy, the Midpoint Method or the Random Method, is the smart one? Why?

2. (Optional) You may wish to write a Computer Guess game program which will produce an output similar to the one shown at the beginning of this chapter. Use the Random Method, the Midpoint Method, or a combination of both. This combination, or Mixed Method, would use the Midpoint Method on the first guess, the Random Method on the second, the Midpoint Method on the third, etc. Is a mixed strategy better than the Midpoint Method above?



CURSE YOU, COMPUTER!

4

PROGRAMMING THE COMPUTER TO DRAW

Our guess-a-number games have served to introduce you to the following operations in BASIC: LET, PRINT, GO TO, IF-THEN, STOP, END, RANDOM, DIM, FOR and NEXT, and INT. You have also been introduced to branching, looping and the use of a subscripted variable to construct check lists and tally lists. In this chapter, by teaching the computer to draw, I'll introduce new operations: TAB, READ and DATA, GO SUB and RETURN, as well as the important concept of the subroutine.

4.1 SQUARES AND RECTANGLES: LOOP BUILDING

Write a program to draw the following design:

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

You probably would call that design a rectangle, but the computer thinks of it as a square since there is an equal number of asterisks, or stars, on each side. There are many possible programs which can be written to draw that square. A very easy program might use a loop to print a line of five stars five times in succession. Here is such a program.

SQUARE/ELEMENTARY

PROGRAM

```
10 FOR C=1 TO 5 STEP 1
20 PRINT "*****"
30 NEXT C
40 END
```

OUTPUT

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

This program gives us an opportunity to review the construction of loops. As you know, the first statement of a loop begins with FOR and the last statement with NEXT. In between the FOR and NEXT statements we sandwich one or more statements in BASIC. The statement

FOR C = 1 TO 5 STEP 1

at the beginning of this loop tells the computer that the variable C is to start at 1 (C = 1), that the next value of C is to be one more than the last value (STEP 1), and that the maximum value of C is to be 5 (C = 1 TO 5). If the

step size is 1, as above, it is not necessary to include the modifier STEP 1. On the other hand, if the difference or step between successive values of the variable is not 1, the word STEP is included, followed by a number representing the step size. The programs we have written in previous chapters all called for a step size of 1, so the modifier was omitted. Here are some examples of the use of the word STEP.

Write a program to print out a list of the whole numbers from 1 to 13.

STEP 1

PROGRAM

```
10 FOR N=1 TO 13
20 PRINT N
30 NEXT N
40 END
```

OUTPUT

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

The modifier STEP 1 was omitted in line 10.

Write a program to print out a list of the whole numbers from 0 to 50 in steps of 5.

STEP 5

PROGRAM

```
10 FOR N=0 TO 50 STEP 5
20 PRINT N
30 NEXT N
40 END
```

OUTPUT

```
 0
 5
10
15
20
25
30
35
40
45
50
```

Write a program to print out a list of the whole numbers 12, 14, 16, 18, 20, 22, 24, 26, 28.

STEP 2

PROGRAM

```
10 FOR N=12 TO 28 STEP 2
20 PRINT N
30 NEXT N
40 END
```

OUTPUT

```
12
14
16
18
20
22
24
26
28
```

Write a program to print out a list of the whole numbers 1 to 13 in reverse order.

```
10 FOR N=13 TO 1 STEP -1
20 PRINT N
30 NEXT N
40 END
```

PROGRAM

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

OUTPUT

STEP −1

Write a program to print out a list of the whole numbers 50, 45, 40, 35, 30, 25, 20, 15, 10, 5, 0.

```
10 FOR N=50 TO 0 STEP -5
20 PRINT N
30 NEXT N
40 END
```

PROGRAM

```
50
45
40
35
30
25
20
15
10
5
0
```

OUTPUT

STEP −5

As you see from these examples, the FOR statement has the form

FOR C = C1 to C2 STEP D

where C is the name of the control variable,
C1 is the initial value of the control variable,
C2 is the final value of the control variable, and
D is the difference (positive or negative) between successive values of C. C is called the control variable because it controls the number of times the statements between FOR and NEXT are executed.

Now let's look at the statements between the FOR and NEXT statements. The statement PRINT "*****" causes five stars to be printed on the same line. This task can also be thought of as printing on one line, one star at a time, five times. To do it this way, however, it's necessary to build a loop within a loop. Thus, 20 PRINT "*****" could be replaced by

```
19 FOR S = 1 TO 5
20 PRINT "*";
21 NEXT S
```

Inserting these steps in the program SQUARE/ ELEMENTARY, we obtain:

```
10 FOR C=1 TO 5
19 FOR S=1 TO 5
20 PRINT "*";
21 NEXT S
30 NEXT C
40 END
```

PROGRAM

SQUARE/2 LOOP

When run, however, this program draws the following design:

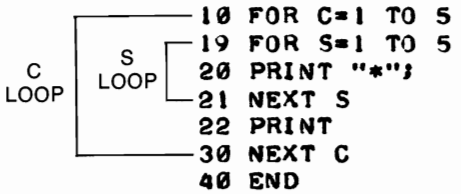
OUTPUT
```
**************************
```

which is not what we expected (certainly I didn't). After five stars have been printed and C goes to its next value, which is 2, the semi-colon at the end of line 20 inhibits the line feed and causes the computer to continue printing on the same line. To signal the computer to move to the next line after the inner loop has been satisfied, we must insert a blank PRINT statement after the NEXT S statement, but before the NEXT C statement. Here is the desired program:

SQUARE *PROGRAM*
```
10 FOR C=1 TO 5
19 FOR S=1 TO 5
20 PRINT "*";
21 NEXT S
22 PRINT
30 NEXT C
40 END
```

OUTPUT
```
*****
*****
*****
*****
*****
```

You have just seen an example of a nested loop, that is a loop within a loop. Look at it this way:



To make nested loops obvious to the reader of your program (including yourself), you can indent the statements comprising the inner loop as follows:

```
10 FOR C=1 TO 5
19     FOR S=1 TO 5
20     PRINT "*";
21     NEXT S
22 PRINT
30 NEXT C
40 END
```

In a simple design like a square, it's quicker to use the PRINT "*****" statement instead of the nested loop, but nested loops have their value, as you shall see shortly.

EXERCISE SET 4.1

On-line:

In each of these on-line exercises, use the pair of words FOR and NEXT as much as possible to produce the desired output.

1. Write a program to generate the following output:

```
25    24    23    22    21
20    19    18    17    16
15    14    13    12    11
10    9     8     7     6
5     4     3     2     1
```

2. Write a program to generate the following output:

```
5     10    15    20    25
30    35    40    45    50
55    60    65    70    75
80    85    90    95    100
105   110   115   120   125
```

3. Write a program to generate the following output:

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

4. Write a program to generate the following output:

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

5. Write a program to generate a rectangular design with the dimensions to be supplied by the user. Your program should produce the following:

```
I CAN DRAW ANY SIZE RECTANGLE AS LONG
AS YOU TELL ME ITS WIDTH AND LENGTH.

WHENEVER YOU WISH ME TO STOP DRAWING
TYPE IN 99 IN RESPONSE TO HOW WIDE.

I'M READY TO DRAW A RECTANGLE.
HOW WIDE? 5
HOW LONG? 3
*****
*****
*****


I'M READY TO DRAW A RECTANGLE.
HOW WIDE? 100
SORRY, I CAN'T DRAW A RECTANGLE THAT WIDE.
TRY AGAIN.


I'M READY TO DRAW A RECTANGLE.
HOW WIDE? 1
HOW LONG? 1
*


I'M READY TO DRAW A RECTANGLE.
HOW WIDE? 99
GOOD BYE---YOUR ELECTRONIC ARTIST.
```

4.2 PARALLELOGRAMS: THE TAB COMMAND

Write a program to produce the following design:

```
XXXXX
 XXXXX
  XXXXX
   XXXXX
    XXXXX
     XXXXX
      XXXXX
```

As in the last section, we'll begin with an obvious program and work towards a version containing a loop.

PARALLELOGRAM/ELEMENTARY

PROGRAM

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

OUTPUT

```
XXXXX
 XXXXX
  XXXXX
   XXXXX
    XXXXX
     XXXXX
      XXXXX
```

To write the loop program, you need to know the maximum number of characters, alphabetic or numeric, that your terminal can print on one line. My terminal, the most common variety, is the Model 33 Teletype. Across the page there are 72 printing positions, or spaces, in which a character may be printed. You might say it looks like this, each dash indicating a space in which a character may be printed.

```
    5    10   15   20   25   30   35   40   45   50   55   60   65   70
    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓    ↓
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
```

To draw a picture such as this:

```
XXXXX
 XXXXX
  XXXXX
   XXXXX
    XXXXX
     XXXXX
      XXXXX
```

you must specify the printing position to be occupied by each X. You can use graph paper. Or, as I've done here, you can hold down the REPEAT key to generate several lines of dashes. In either case, mark the positions to be occupied by an X.

```
-XXXXX---------
--XXXXX--------
---XXXXX-------
----XXXXX------
-----XXXXX-----
------XXXXX----
-------XXXXX---
```

1. The first line has 1 blank space followed by 5 X's.
2. The second line has 2 blank spaces followed by 5 X's.
3. The third line has 3 blank spaces followed by 5 X's.
4. The fourth line has 4 blank spaces followed by 5 X's.
5. The fifth line has 5 blank spaces followed by 5 X's.
6. The sixth line has 6 blank spaces followed by 5 X's.
7. The seventh line has 7 blank spaces followed by 5 X's.

Two tasks are involved in drawing the parallelogram. The first is to make the Teletype printing head count out the appropriate number of blank spaces on each line. The second task is to print the correct number of X's, which in this case is always five. To cause the carriage to count off a specified number of blank spaces, we will use the TAB command. For instance, to move the carriage five spaces to the right, placing it in position to print in the sixth space, we write:

PRINT TAB(5);

We end the statement with a semi-colon, signalling the computer to print the next character in the sixth position. In the general case, to move the carriage N spaces in from the left margin you write:

PRINT TAB(N);

leaving the carriage ready to print in space N + 1. With this new BASIC command, TAB, we can rewrite our program PARALLELOGRAM/ELEMENTARY to produce the following intermediate version.

```
10 PRINT TAB(1);"XXXXX"  PROGRAM  XXXXX  OUTPUT   PARALLELOGRAM/TAB
20 PRINT TAB(2);"XXXXX"             XXXXX
30 PRINT TAB(3);"XXXXX"              XXXXX
40 PRINT TAB(4);"XXXXX"               XXXXX
50 PRINT TAB(5);"XXXXX"                XXXXX
60 PRINT TAB(6);"XXXXX"                 XXXXX
70 PRINT TAB(7);"XXXXX"                  XXXXX
80 END
```

Look back at the elementary program; all we've done is eliminate the blank spaces in quotes.

Now that I've shown you how to use the TAB command in drawing a parallelogram, let me show you a program which uses a loop in drawing the same parallelogram. Look at the above program, particularly the sequence of numbers from 1 to 7 after the TAB commands. This sequence suggests a loop to me. To build

this program with a loop involves two tasks. This two-part procedure will be used not only to build a program to draw parallelograms, but also to build loop programs to draw triangles, diamonds and trapezoids.

The two tasks are (1) determine how many lines of printing are needed to draw the picture, and (2) determine the composition of each line of the design. The first task is usually easy. You look at the graph paper on which you laid out the figure and count the number of lines in the design. Write FOR and NEXT statements which will control the number of lines to be printed. In this parallelogram there are seven lines. Thus the first and last statements of the loop are:

```
FOR L = 1 TO 7

NEXT L
```

Now let's fill in the middle of the loop. What is the composition of each line in the design? Each line in this parallelogram is composed of a number of blanks followed by five X's. Let us first find a formula which specifies the number of blanks to be counted off from the left margin on each line. To find the formula, make a table which lists the line number of the design and the number of blanks in that line.

Line Number	Number of Blanks
1	1
2	2
3	3
4	4
5	5
6	6
7	7

In this case the number of blanks necessary for any line is the same as the line number. Thus, to draw line 3, we must leave three blanks before the five X's, and to draw line 7, we must leave seven blanks before the five X's. To draw any line L in this parallelogram, we count off L blanks before printing five X's. As you learned in writing PARALLELOGRAM/TAB, the BASIC word most helpful to us here is TAB. If we're drawing line L, we can count off L blanks by writing PRINT TAB(L);. The semi-colon is required after TAB(L) for there is more printing to be done on the same line, in this case five X's. We now insert PRINT TAB(L); in the loop immediately after the FOR statement.

```
FOR L = 1 TO 7
PRINT TAB(L);

NEXT L
```

After counting off L blanks from the left margin, the five X's can be drawn by writing: PRINT "XXXXX".

We insert this new statement after PRINT TAB(L);.

```
FOR L = 1 TO 7
PRINT TAB(L);
PRINT "XXXXX"
NEXT L
```

Now we add line numbers and an END statement and have the following program to draw a parallelogram which slopes "downhill."

PARALLELOGRAM/DOWNHILL

```
10 FOR L=1 TO 7 PROGRAM
20 PRINT TAB(L);
30 PRINT "XXXXX"
40 NEXT L
50 END
```

```
XXXXX OUTPUT
 XXXXX
  XXXXX
   XXXXX
    XXXXX
     XXXXX
      XXXXX
```

To reduce the number of lines in the program, you can combine lines 20 and 30 into one line, as shown below.

PARALLELOGRAM/DOWNHILL/ELEGANT

```
10 FOR L=1 TO 7          PROGRAM
20 PRINT TAB(L);"XXXXX"
30 NEXT L
40 END
```

```
XXXXX OUTPUT
 XXXXX
  XXXXX
   XXXXX
    XXXXX
     XXXXX
      XXXXX
```

Now let's write a program to produce a parallelogram which slopes "uphill."

Write a program to draw the following design. The dashes and column numbers are not part of the design. They are to help you count spaces.

```
    5    10   15   20
    ↓    ↓    ↓    ↓
----------------------
        XXXXX
       XXXXX
      XXXXX
     XXXXX
    XXXXX
   XXXXX
  XXXXX
 XXXXX
```

As in the development of the program PARALLELOGRAM/DOWNHILL, two operations are required. (1) Determine how many lines of printing are needed to draw the picture, and (2) determine the composition of each line of the design. There are seven lines in the new "uphill" parallelogram. Thus, the first and last statements of the loop are:

```
FOR L = 1 TO 7
NEXT L
```

The next job is to determine the composition of each line. As in PARALLELOGRAM/DOWNHILL, each line is composed of a number of blanks followed by five

X's. Here is a table listing the number of blanks on each line.

Line Number	Number of Blanks
1	7
2	6
3	5
4	4
5	3
6	2
7	1

The number of blanks is no longer the same as the line number. What we need is an expression which will produce the value 7 when L is 1, 6 when L is 2, 5 when L is 3, . . . , 1 when L is 7. Here's a way to find the expression. For each line in the above table, find the sum of the line number and the number of blanks.

Line Number	Number of Blanks	Sum
1	7	8
2	6	8
3	5	8
4	4	8
5	3	8
6	2	8
7	1	8

Notice in the new table the number of blanks is always equal to 8, minus the line number. In short, the expression is 8-L. Try it.

Line Number	Number of Blanks
1	8 − 1 = 7
2	8 − 2 = 6
3	8 − 3 = 5
4	8 − 4 = 4
5	8 − 5 = 3
6	8 − 6 = 2
7	8 − 7 = 1

As above, you will frequently discover a constant or some other "pattern" by adding or subtracting the line number and the desired number of blanks. The pattern is then useful in building the expression to generate the number of blanks. In general, if the pattern is a constant, try addition or subtraction in your expression. If the pattern is not a constant, try multiplication.

Now that we have the expression 8 − L for the "uphill" parallelogram, we insert after the FOR statement, PRINT TAB (8 − L);.

```
FOR L = 1 TO 7
PRINT TAB (8 - L);
NEXT L
```

To print out five X's on each line, we now add PRINT "XXXXX" after PRINT TAB (8 − L);.

```
FOR L = 1 TO 7
PRINT TAB (8 − L);
PRINT "XXXXX"
NEXT L
```

Adding line numbers and an END statement, we have the following program.

PARALLELOGRAM/UPHILL

```
10 FOR L=1 TO 7
20 PRINT TAB(8-L);
30 PRINT "XXXXX"
40 NEXT L
50 END
```

PROGRAM

```
 XXXXX
 XXXXX
 XXXXX
 XXXXX
 XXXXX
 XXXXX
 XXXXX
```

OUTPUT

Lines 20 and 30 can be combined in one line as follows:

PARALLELOGRAM/ UPHILL/ELEGANT

```
10 FOR L=1 TO 7
20 PRINT TAB(8-L);"XXXXX"
30 NEXT L
40 END
```

PROGRAM

```
 XXXXX
 XXXXX
 XXXXX
 XXXXX
 XXXXX
 XXXXX
```

OUTPUT

There is a second program which will draw the parallelogram. To construct this new program we will follow the same two-part procedure, but will number the lines of the design in the reverse order. Up to this point we have always designated the top line of the design as line 1, the next line as line 2, etc. This time we'll number them as follows:

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

If we adopt this new numbering scheme, the table which lists the line numbers and the number of blanks looks like this.

Line Number	Number of Blanks
7	7
6	6
5	5
4	4
3	3
2	2
1	1

Now let's follow our two-part procedure to write our new program. First, we write a loop to define the line

numbers in the sequence 7, 6, 5, 4, 3, 2, 1. Earlier in this chapter you learned how to use a FOR statement to generate a sequence of numbers, in either ascending or descending order, by placing the modifier STEP after the word FOR. STEP is then followed by a number denoting the difference between successive numbers in the sequence. The FOR statement to generate 7, 6, 5, 4, 3, 2, 1 is FOR L = 7 to 1 STEP − 1. We can now write the two loop control statements.

```
FOR L = 7 TO 1 STEP −1
NEXT L
```

The second task is to determine the composition of each line of the design. Look at the above table. In each case, the number of blanks is the same as the line number. Thus, when drawing line L, we count off L blanks and print five X's. To count off L blanks, we add to our program PRINT TAB (L);.

```
FOR L = 7 TO 1 STEP −1
PRINT TAB(L);
NEXT L
```

To print the five X's, we add "XXXXX" after TAB(L);. After inserting an END statement and line numbers we have:

PARALLELOGRAM/ UPHILL/STEP −1

PROGRAM

```
10 FOR L=7 TO 1 STEP -1
20 PRINT TAB(L);"XXXXX"
30 NEXT L
40 END
```

OUTPUT

```
XXXXX
 XXXXX
  XXXXX
   XXXXX
    XXXXX
     XXXXX
      XXXXX
```

Numbering the lines of the design in reverse order, or in some other order, often makes it easier to find an expression to be used with TAB.

EXERCISE SET 4.2

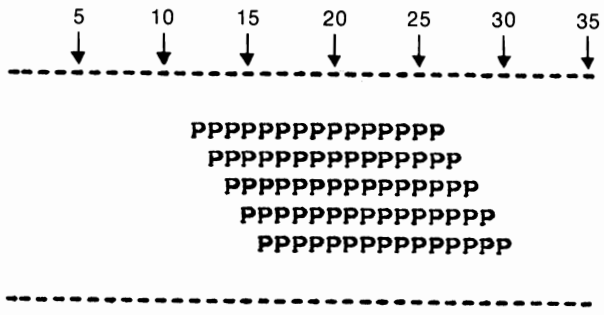
On-line:

In each of these on-line exercises use the TAB command and FOR and NEXT. Also, if appropriate, use nested loops. Dashes with column numbers have been given at the top and bottom of each design to aid in determining the exact location of blanks and characters.

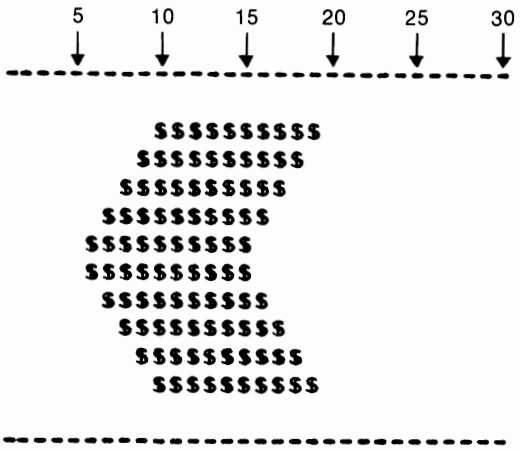
1. Write a program to draw this design:

```
    5    10   15
    ↓    ↓    ↓
----+----+----+----

HELLO
 HELLO
  HELLO
   HELLO
    HELLO

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

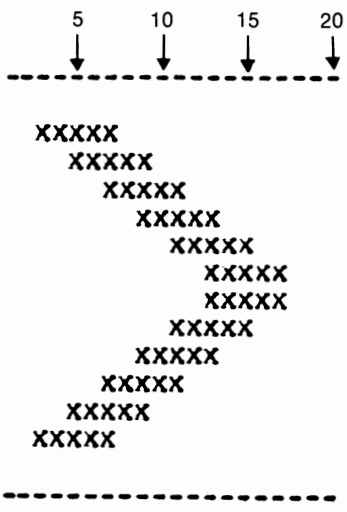
2. Write a program to draw this design:



3. Write a program to draw this design:



4. Write a program to draw this design:



4.3 TRIANGLES, TRAPEZOIDS, AND DIAMONDS: TABBING FORMULAS

In the first section of this chapter, you wrote programs to draw squares and rectangles. These were easy to construct, since each line of the design had the same number of blanks from the left margin, followed by the same number of non-blank characters. In the last section you learned how to use a TAB expression to vary the number of blanks from the left margin for each line of

the design. In this section we'll continue to use TAB to vary the number of blanks from the left margin, and, in addition, learn how to vary the number of non-blank characters to be printed on each line.

Write a program to draw the following triangle. Dashes and column numbers have been provided to aid in identifying the exact location of the blanks and stars.

TRIANGLE/ELEMENTARY

An elementary program to draw this triangle is:

PROGRAM

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

OUTPUT

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

TRIANGLE/TAB

Using the TAB command, the program looks like this.

PROGRAM

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

OUTPUT

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

Let's proceed to a more elegant program to draw the desired triangle. As in developing the routine to produce the uphill and downhill parallelograms in the last section, two tasks are involved. (1) Determine how many lines of printing are needed to draw the design, and (2) determine the composition of each line of the design. Part one is easy. How many lines of printing are needed to draw the triangle? Six. Thus, the FOR and NEXT statements to define the loop are:

```
FOR L = 1 TO 6
NEXT L
```

Now on to part two, the determination of the format of each line of the design. Here in table form is the structure of each line of the triangle.

Line Number	Number of Blanks	Number of Stars
1	5	1
2	4	3
3	3	5
4	2	7
5	1	9
6	0	11

Look at the table. Since the number of blanks varies from line to line and the number of stars varies from line to line, two expressions must be derived, one for the blanks and one for the stars. First the expression for the number of blanks. As in finding the expression for the number of blanks required to draw the uphill parallelogram, the first step is to try to find a constant, or some other pattern, by adding or subtracting each line number and the corresponding number of blanks for that line. When we subtract, the result is not a constant. When we add, the sum is always 6. Now can you find the expression? Is the expression 6 − L or L − 6? The correct choice is 6 − L. Let's try it.

Line Number L	Number of Blanks	6 − L
1	5	6 − 1=5
2	4	6 − 2=4
3	3	6 − 3=3
4	2	6 − 4=2
5	1	6 − 5=1
6	0	6 − 6=0

We can now insert after FOR L = 1 TO 6 the statement PRINT TAB(6 − L);.

```
FOR L = 1 TO 6
PRINT TAB(6 − L);
NEXT L
```

Now for an expression that specifies the number of stars to be printed on each line. Here is a table listing just the line number and the number of stars.

Line Number	Number of Stars
1	1
2	3
3	5
4	7
5	9
6	11

Look at the numbers in the Stars column. They are all odd numbers. We need an expression to generate odd numbers when the original sequence of numbers is 1, 2, 3, 4, 5, 6. An expression which produces the odd

numbers is: 2*L − 1. Let's try it.

Line Number, L	Number of Stars	2*L − 1
1	1	2*1 − 1=2 − 1=1
2	3	2*2 − 1=4 − 1=3
3	5	2*3 − 1=6 − 1=5
4	7	2*4 − 1=8 − 1=7
5	9	2*5 − 1=10 − 1=9
6	11	2*6 − 1=12 − 1=11

To print out 2*L − 1 stars on each line of the design, a loop must be inserted after PRINT TAB(6 − L);. Here is the S loop, nested in the L loop.

```
FOR L = 1 TO 6
PRINT TAB(6 − L);
    FOR S = 1 TO 2*L − 1
    PRINT "*";
    NEXT S
NEXT L
```

Adding the END statement and line numbers, we obtain:

TRIANGLE/BUG

PROGRAM

```
10 FOR L=1 TO 6
20 PRINT TAB(6-L);
30    FOR S=1 TO 2*L-1
40    PRINT "*";
50    NEXT S
60 NEXT L
70 END
```

OUTPUT

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

Look at the output. What happened to the triangle? Can you find the bug in the program? If you load and run the above program and watch the output carefully, you will observe:

(1) Five blanks are counted off followed by one star.
(2) The carriage is returned to the left margin without a line feed.
(3) Four blanks are counted off followed by three stars.
(4) The carriage is returned to the left margin without a line feed.
(5) Three blanks are counted off followed by five stars, etc.

What's the problem? The difficulty stems from the semi-colon after "*" in line 40. After the inner loop is satisfied and the NEXT L is generated, printing continues on the same line. To eliminate this bug, insert a blank PRINT statement after NEXT S. Here is our final program for the triangle.

TRIANGLE

PROGRAM

```
10 FOR L=1 TO 6
20 PRINT TAB(6-L);
30    FOR S=1 TO 2*L-1
40    PRINT "*";
50    NEXT S
55 PRINT
60 NEXT L
70 END
```

OUTPUT

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

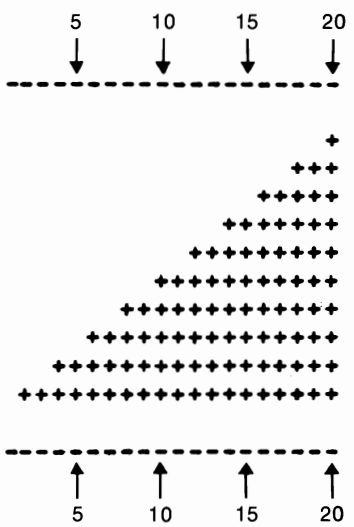
EXERCISE SET 4.3

On-line:

In each of these on-line exercises use the TAB command and FOR and NEXT. Also use nested loops.

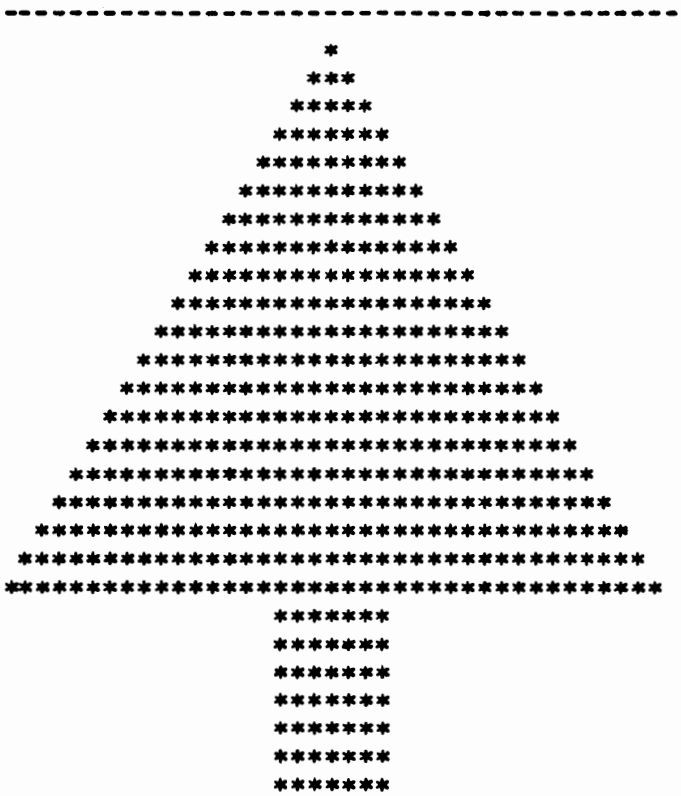
1. Write a program to draw:

2. Write a program to draw:

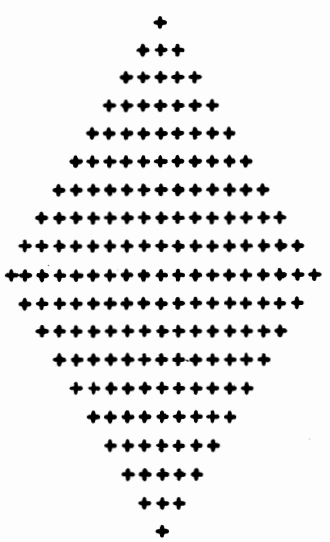


3. Combining the methods you have learned for drawing triangles and rectangles, write a program to

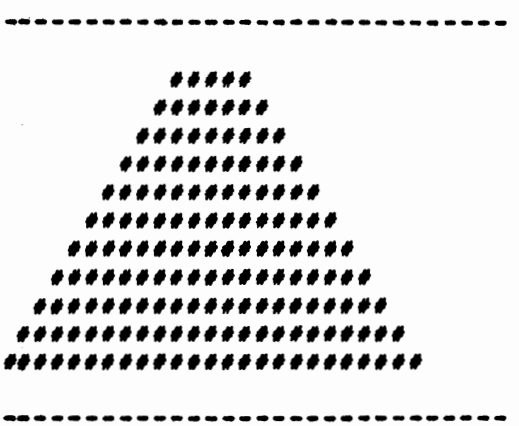
draw a tree such as the following:



4. Write a program to produce the following diamond or one of your own design.



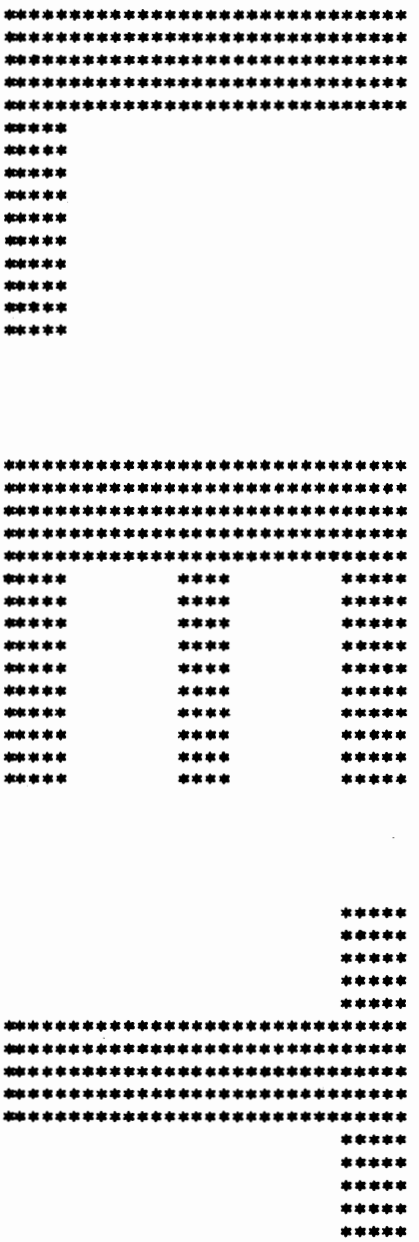
5. Write a program to produce the following trapezoid or one of your own design.

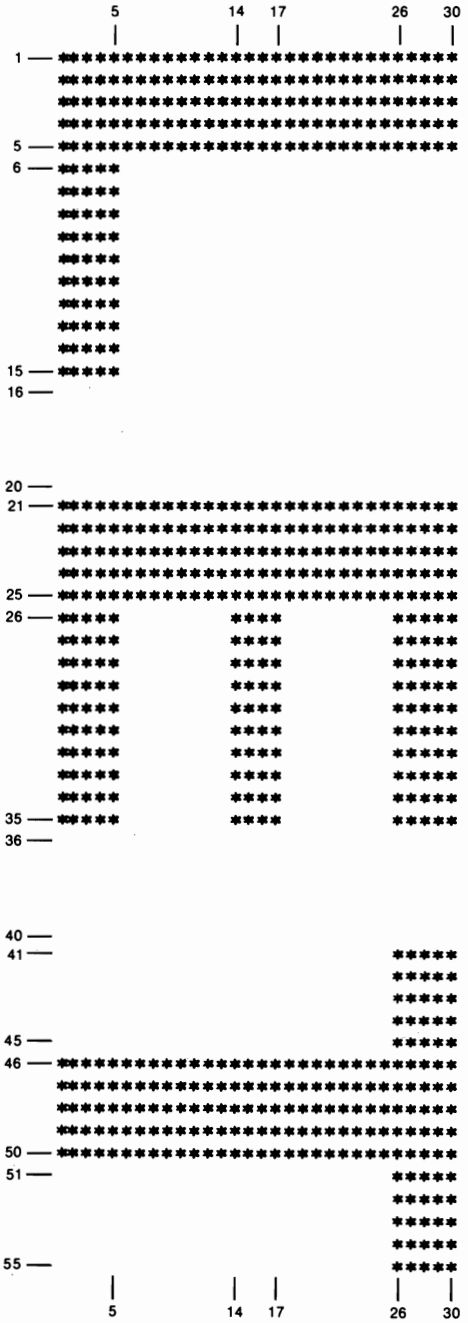


4.4 BLOCK LETTERS: SUBROUTINES

You'll have a lot of fun with a poster program which types out in large block letters any message you specify. Happily, you don't need a different program for each of the 26 letters of the alphabet! In this section you will learn how to put subroutines together to build a new and complex program with a minimum amount of work.

Write a program to type out the word LET as it appears at right.





Here is another copy of LET to which I have added column and line numbers for reference purposes.

Let's start in an elementary way, as we've done many times before. One way to draw LET is to write a program which has 55 PRINT statements, one for each line in the design. For example, to create line 1 you would write:

```
1 PRINT ''******************************''.
```

Or you can construct a loop for each line, such as this one for line 1.

```
1 FOR S = 1 TO 30
2 PRINT ''*'';
3 NEXT S
4 PRINT
```

Either approach works, but is time-consuming and results in an unwieldy program.

The next best approach is to combine adjacent lines in units which require the same programming steps.

1. Lines 1-5 consist of 5 lines with 30 stars each.
2. Lines 6-15 consist of 10 lines with 5 stars each.
3. Lines 16-20 consist of 5 blank lines.
4. Lines 21-25 consist of 5 lines with 30 stars each.
5. Lines 26-35 consist of 10 lines each with 5 stars followed by 8 blanks, 4 stars, 8 blanks, and 5 stars.
6. Lines 36-40 consist of 5 blank lines.
7. Lines 41-45 consist of 5 lines with 25 blanks followed by 5 stars.
8. Lines 46-50 consist of 5 lines with 30 stars each.
9. Lines 51-55 consist of 5 lines with 25 blanks followed by 5 stars.

A program which follows this analysis is:

True, there are more lines (59) in this program than in a program in which there is one PRINT statement for each line (56 lines), but less typing is required, especially of stars.

A further step in the same direction is to find elements of the design which are repeated. Look at the word LET. Find a rectangle which is repeated in each letter of the word. It is the rectangle formed by design lines 1-5, 21-25, and 46-50. Look at the program LET/ELEMENTARY and find the corresponding sets of program steps which draw this rectangle. The repeated set of program steps to draw this rectangle is:

```
FOR L = 1 TO 5
FOR S = 1 TO 30
PRINT "*";
NEXT S
PRINT
NEXT L
```

These six steps are repeated in LET/ELEMENTARY at program lines 5-30, 80-105, and 230-255.

Find a second element of the design which is repeated. The five blank lines between the letters L and E, and E and T are repeated. Can you find the corresponding set of program steps which generate these five blank lines? The set of steps is:

```
FOR L = 1 TO 5
PRINT
NEXT L
```

These three steps are repeated at program lines 65-75 and 180-190.

There is another rectangle which appears twice in the design. It is five stars wide and five stars long and forms the left and right branches of the letter T. Can you find the corresponding program steps in LET/ELEMENTARY? They are:

```
FOR L = 1 TO 5
PRINT TAB(25);
FOR S = 1 TO 5
PRINT "*";
NEXT S
PRINT
NEXT L
```

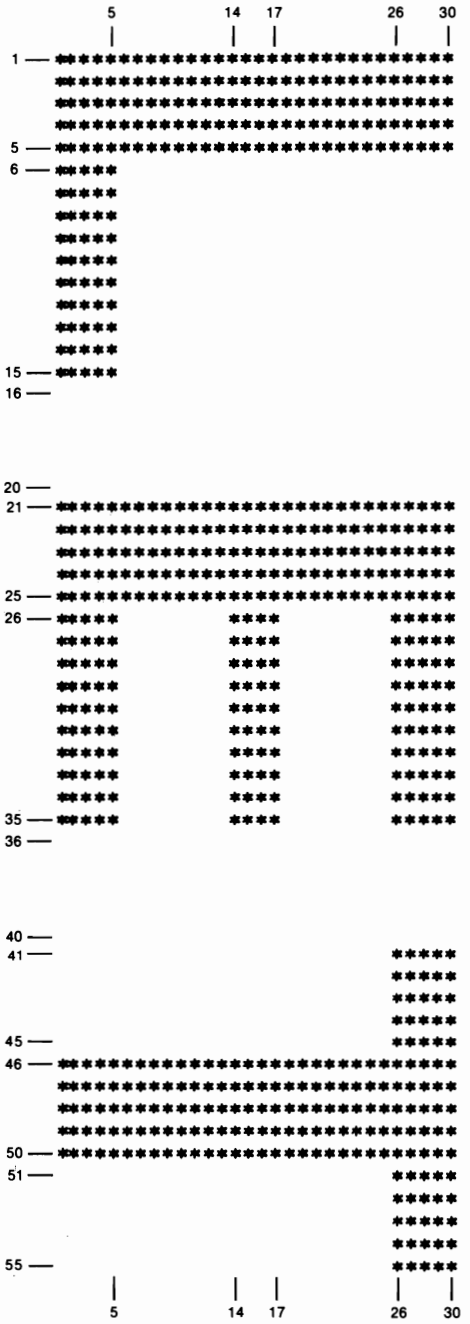
and are located at program lines 195-225 and 260-290. TAB(25); is needed to have the rectangle located 25 spaces from the left margin.

There is yet another rectangle which is repeated in the design. It occurs in design lines 6-15 and in lines 26-35 as the top and bottom bars of the letter E. This rectangle is five stars wide and ten stars long. Although this rectangle is repeated in the design, design lines 6-15 are

PROGRAM **LET/ELEMENTARY**

```
 5 FOR L=1 TO 5
10 FOR S=1 TO 30
15 PRINT "*";
20 NEXT S
25 PRINT
30 NEXT L
35 FOR L=1 TO 10
40 FOR S=1 TO 5
45 PRINT "*";
50 NEXT S
55 PRINT
60 NEXT L
65 FOR L=1 TO 5
70 PRINT
75 NEXT L
80 FOR L=1 TO 5
85 FOR S=1 TO 30
90 PRINT "*";
95 NEXT S
100 PRINT
105 NEXT L
110 FOR L=1 TO 10
115 FOR S=1 TO 5
120 PRINT "*";
125 NEXT S
130 PRINT TAB(13);
135 FOR S=1 TO 4
140 PRINT "*";
145 NEXT S
150 PRINT TAB(25);
155 FOR S=1 TO 5
160 PRINT "*";
165 NEXT S
170 PRINT
175 NEXT L
180 FOR L=1 TO 5
185 PRINT
190 NEXT L
195 FOR L=1 TO 5
200 PRINT TAB(25)
205 FOR S=1 TO 5
210 PRINT "*";
215 NEXT S
220 PRINT
225 NEXT L
230 FOR L=1 TO 5
235 FOR S=1 TO 30
240 PRINT "*";
245 NEXT S
250 PRINT
255 NEXT L
260 FOR L=1 TO 5
265 PRINT TAB(25);
270 FOR S=1 TO 5
275 PRINT "*";
280 NEXT S
285 PRINT
290 NEXT L
295 END
```

LINES 1-5
LINES 6-15
LINES 16-20
LINES 21-25
LINES 26-35
LINES 36-40
LINES 41-45
LINES 46-50
LINES 51-55



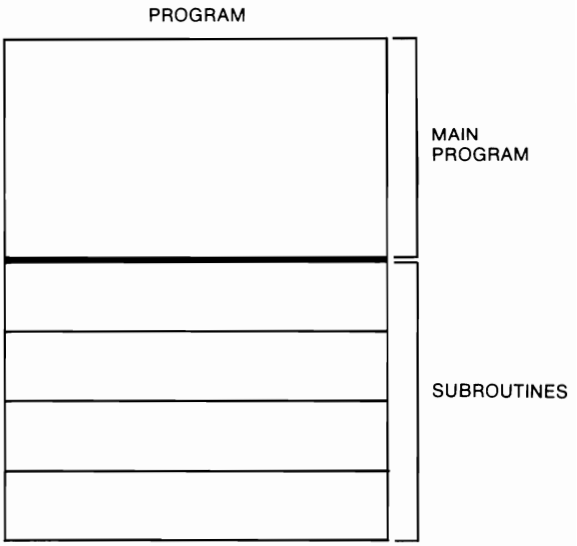
not entirely the same as design lines 26-35, and the corresponding set of program steps to draw lines 6-15 is not the same as the set of steps to draw design lines 26-35. Look at the program LET/ELEMENTARY and see for yourself.

To sum up, three patterns are repeated in the design, and three corresponding sets of instructions — or routines — are repeated in the program. Here are the repeated patterns:

(1) The rectangle 30 stars wide and 5 stars long occurs three times.
(2) The five blank lines occur twice.
(3) The rectangle 5 stars wide and 5 stars long occurs twice.

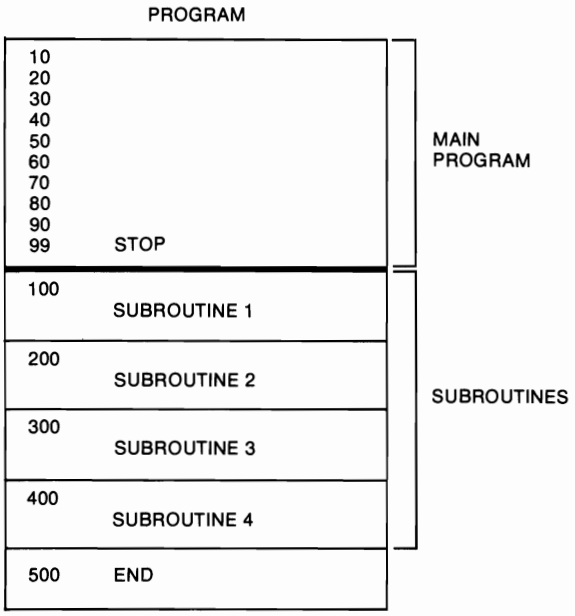
It must be possible to gain some advantage from the fact that the routines corresponding to the patterns are repeated in the program. We can reduce the overall number of steps in the program by entering into the program only once any set of steps that is repeated. These steps will be set aside in the program and identified as *subroutines*. Any subroutine can be activated as required by the main program.

Here is the architecture of a program which uses subroutines. The first section contains the main program and the second section all the subroutines.

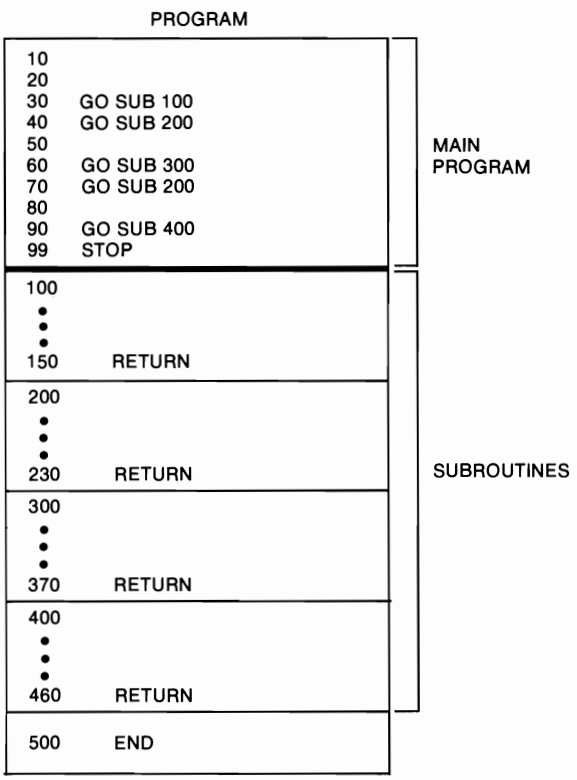


The command STOP marks the end of the main program or first section. As always, the last statement in the program is the END statement. Adding typical line numbers, the architecture of a program employing four

subroutines looks like this:



To signal the computer to jump from the main program to a particular subroutine, you write in the main program a statement which begins with GO SUB followed by the beginning line number of the subroutine you wish to activate. Thus, in the above example, if, in line 30, you wish to call for subroutine 1, you write in line 30 GO SUB 100. At the end of the subroutine, you put in a RETURN command. Thus, if the last line in subroutine 1 is 140, you write 150 RETURN. RETURN signals the computer to return to the main program and pick up where it left off. More precisely, in this case the computer would return to the next line after line 30, which is line 40. GO SUB is similar to the GO TO command, except that the GO SUB command allows the computer to reenter the main program and pick up where it left off when it encounters the RETURN statement at the end of the subroutine. Look at our example above. If we wish to activate subroutine 1 at line 30, subroutine 2 at line 40, subroutine 3 at line 60, subroutine 2 at line 70, and subroutine 4 at line 90, our program looks like this:



Let's construct a program using subroutines to draw LET. The program consists of two sections, either of which can be written first. Obviously the two parts are interrelated, however. If you write the main program first, you must keep in mind the various subroutines you'll be writing in the second section. If you write the subroutines first, you know that, when you write the main program, you'll be calling for them as needed. We'll write the subroutines first and then the main program.

The first subroutine will draw a rectangle 30 stars wide and 5 stars long. What shall we use for line numbers? When writing subroutines before writing the main program, you must make an educated guess as to the range of line numbers you should reserve for the statements in the main program. Let's reserve lines 1-499 for the main program and begin our first subroutine at line 500. Here is subroutine 1 which draws the rectangle 30 stars wide and 5 stars long.

```
Subroutine 1:   500 FOR L = 1 TO 5
                505 FOR S = 1 TO 30
                510 PRINT "*";
                515 NEXT S
                520 PRINT
                525 NEXT L
                530 RETURN
```

Note the RETURN in line 530.

The second subroutine will generate five blank lines. To separate this second subroutine from the first, begin with line number 600. Changing the hundreds digit from 5 to 6 will aid in locating the two routines in the completed program.

Subroutine 2:
```
600 FOR L = 1 TO 5
605 PRINT
610 NEXT L
615 RETURN
```

Don't forget the RETURN statement to terminate the routine.

The third subroutine will draw a 5 by 5 rectangle 25 spaces from the left margin. We begin this third routine at line 700.

Subroutine 3:
```
700 FOR L = 1 TO 5
705 PRINT TAB(25);
710 FOR S = 1 TO 5
715 PRINT "*";
720 NEXT S
725 PRINT
730 NEXT L
735 RETURN
```

If we put these three subroutines together, the second section of the program looks like this:

```
500 FOR L=1 TO 5
505 FOR S=1 TO 30
510 PRINT "*";
515 NEXT S
520 PRINT
525 NEXT L
530 RETURN
```
SUBROUTINE 1

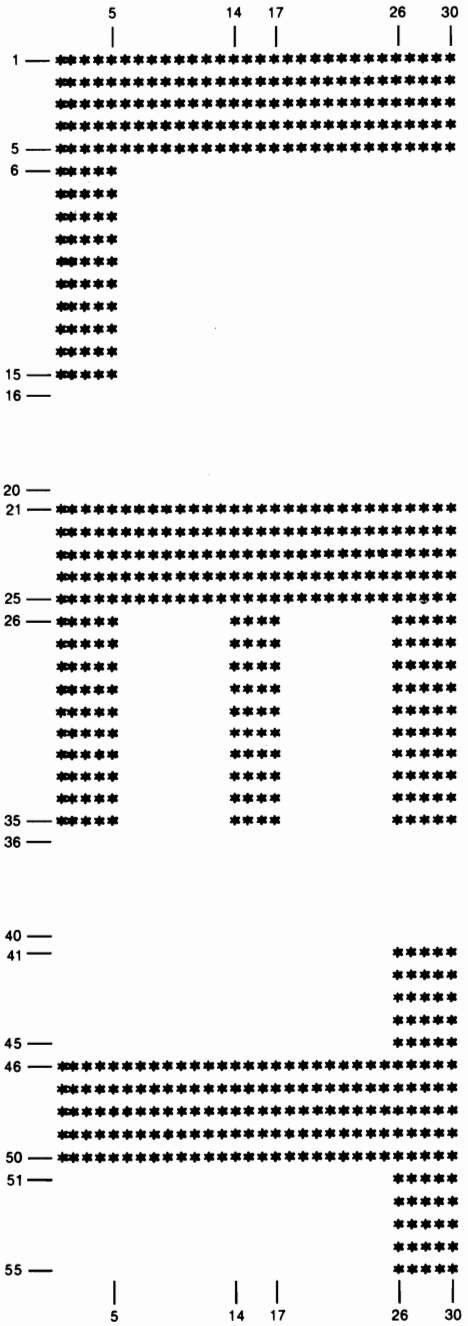
```
600 FOR L=1 TO 5
605 PRINT
610 NEXT L
615 RETURN
```
SUBROUTINE 2

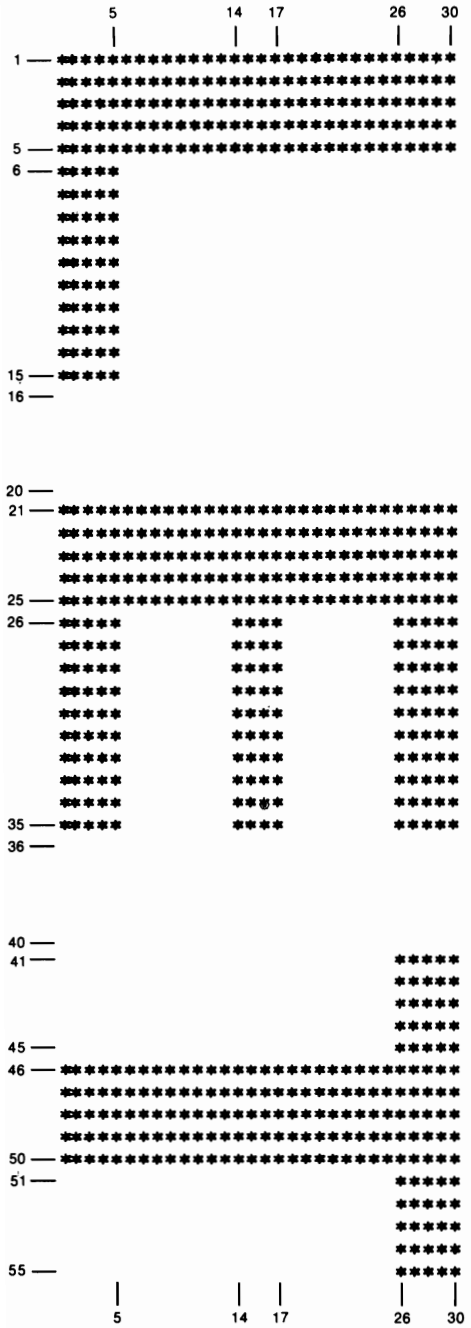
```
700 FOR L=1 TO 5
705 PRINT TAB(25);
710 FOR S=1 TO 5
715 PRINT "*";
720 NEXT S
725 PRINT
730 NEXT L
735 RETURN
```
SUBROUTINE 3

```
800 END
```

In constructing the main program, use the design of LET as a guide. The first design element you want to draw is a rectangle 30 stars wide and 5 stars long in design lines 1-5. The steps to draw this rectangle already exist in subroutine 1 beginning at line 500. So the first step in the main program is:

```
10 GO SUB 500
```





The second design element is a rectangle five stars wide and ten stars long in design lines 6-15. Since this element occurs only once, we have not put it in a subroutine. It is programmed directly in the main program by adding the following lines.

```
15 FOR L = 1 TO 10
20 FOR S = 1 TO 5
25 PRINT "*";
30 NEXT S
35 PRINT
40 NEXT L
```

The third design element consists of 5 blank lines in design lines 16-20. Program steps to do this have already been written as subroutine 2 beginning at line 600. Add to the main program:

```
45 GO SUB 600
```

The next element is a rectangle 30 stars wide and 5 stars long in design lines 21-25. To draw this rectangle again, we recall subroutine 1:

```
50 GO SUB 500
```

Design lines 26-35, the three bars of the letter E, occur only once in the design. The steps to draw them go directly in the main program. Read them carefully, particularly the commands TAB(13) and TAB(25).

```
 55 FOR L = 1 TO 10
 60 FOR S = 1 TO 5
 65 PRINT "*";
 70 NEXT S
 75 PRINT TAB(13);
 80 FOR S = 1 TO 4
 85 PRINT "*";
 90 NEXT S
 95 PRINT TAB(25);
100 FOR S = 1 TO 5
105 PRINT "*";
110 NEXT S
115 PRINT
120 NEXT L
```

To draw design lines 36-40, the five blank lines, we call for subroutine 2 again.

```
125 GO SUB 600
```

To draw design lines 41-45, the 5 by 5 rectangle, we call for subroutine 3, beginning at program line 700.

```
130 GO SUB 700
```

To draw design lines 46-50 we call for the first subroutine again.

```
135 GO SUB 500
```

To draw design lines 51-55 we recall subroutine 3.

```
140 GO SUB 700
```

These are all the steps required to draw the design. A STOP statement must be written to keep the main program from running on into the second section. Be careful; the STOP statement is often forgotten in the first draft of the program.

```
145 STOP
```

Here is the complete main program.

```
 10 GO SUB 500
 15 FOR L=1 TO 10
 20 FOR S=1 TO 5
 25 PRINT "*";
 30 NEXT S
 35 PRINT
 40 NEXT L
 45 GO SUB 600
 50 GO SUB 500
 55 FOR L=1 TO 10
 60 FOR S=1 TO 5
 65 PRINT "*";
 70 NEXT S
 75 PRINT TAB(13);
 80 FOR S=1 TO 4
 85 PRINT "*";
 90 NEXT S
 95 PRINT TAB(25);
100 FOR S=1 TO 5
105 PRINT "*";
110 NEXT S
115 PRINT
120 NEXT L
125 GO SUB 600
130 GO SUB 700
135 GO SUB 500
140 GO SUB 700
145 STOP
```

Combining the two sections we obtain the program LET/3 SUB.

LET/3 SUB ***PROGRAM***

MAIN PROGRAM

```
10 GO SUB 500
15 FOR L=1 TO 10
20 FOR S=1 TO 5
25 PRINT "*";
30 NEXT S
35 PRINT
40 NEXT L
45 GO SUB 600
50 GO SUB 500
55 FOR L=1 TO 10
60 FOR S=1 TO 5
65 PRINT "*";
70 NEXT S
75 PRINT TAB(13);
80 FOR S=1 TO 4
85 PRINT "*";
90 NEXT S
95 PRINT TAB(25);
100 FOR S=1 TO 5
105 PRINT "*";
110 NEXT S
115 PRINT
120 NEXT L
125 GO SUB 600
130 GO SUB 700
135 GO SUB 500
140 GO SUB 700
145 STOP
```

DESIGN LINES 1-5

DESIGN LINES 6-15

DESIGN LINES 16-20

DESIGN LINES 21-25

DESIGN LINES 26-35

DESIGN LINES 36-40

DESIGN LINES 41-45

DESIGN LINES 46-50

DESIGN LINES 51-55

SUBROUTINES

```
500 FOR L=1 TO 5
505 FOR S=1 TO 30
510 PRINT "*";
515 NEXT S
520 PRINT
525 NEXT L
530 RETURN
600 FOR L=1 TO 5
605 PRINT
610 NEXT L
615 RETURN
700 FOR L=1 TO 5
705 PRINT TAB(25);
710 FOR S=1 TO 5
715 PRINT "*";
720 NEXT S
725 PRINT
730 NEXT L
735 RETURN
800 END
```

SUBROUTINE 1

SUBROUTINE 2

SUBROUTINE 3

The advantage gained by using subroutines in this case is relatively small, reducing the number of steps from 59 to 48. But the saving can be greater in more complex programs.

It is possible to further shorten the program LET/3 SUB. There is a set of three steps which is repeated four times, three times in the main program and once in the subroutines:

```
FOR S = 1 TO 5
PRINT "*";
NEXT S
```

These steps occur at: (1) program lines 20, 25, 30 (2) program lines 60, 65, 70, (3) program lines 100, 105, 110, and at (4) program lines 710, 715, 720. This repetition of the same three steps suggests another subroutine. Let's write a fourth subroutine beginning at line 800 and move the END statement to 900.

Subroutine 4:

```
800 FOR S = 1 TO 5
805 PRINT "*";
810 NEXT S
815 RETURN
```

Here is a copy of our new program, LET/4 SUB, to the right of LET/3 SUB. I've indicated where the three old steps have been replaced by GO SUB 800, further reducing the program from 48 to 44 steps. Note that subroutine 3, beginning at line 700, calls, in line 715, for subroutine 4. In other words, subroutines may be "nested" in much the same way as loops.

PROGRAM **LET/3 SUB**

```
 10 GO SUB 500
 15 FOR L=1 TO 10
 20 FOR S=1 TO 5
 25 PRINT "*";
 30 NEXT S
 35 PRINT
 40 NEXT L
 45 GO SUB 600
 50 GO SUB 500
 55 FOR L=1 TO 10
 60 FOR S=1 TO 5
 65 PRINT "*";
 70 NEXT S
 75 PRINT TAB(13);
 80 FOR S=1 TO 4
 85 PRINT "*";
 90 NEXT S
 95 PRINT TAB(25);
100 FOR S=1 TO 5
105 PRINT "*";
110 NEXT S
115 PRINT
120 NEXT L
125 GO SUB 600
130 GO SUB 700
135 GO SUB 500
140 GO SUB 700
145 STOP
500 FOR L=1 TO 5
505 FOR S=1 TO 30
510 PRINT "*";
515 NEXT S
520 PRINT
525 NEXT L
530 RETURN
600 FOR L=1 TO 5
605 PRINT
610 NEXT L
615 RETURN
700 FOR L=1 TO 5
705 PRINT TAB(25);
710 FOR S=1 TO 5
715 PRINT "*";
720 NEXT S
725 PRINT
730 NEXT L
735 RETURN
800 END
```

PROGRAM **LET/4 SUB**

```
 10 GO SUB 500 <-- ACTIVATE SUB 1
 15 FOR L=1 TO 10
 20
 25 GO SUB 800 <-- ACTIVATE SUB 4
 30
 35 PRINT
 40 NEXT L
 45 GO SUB 600 <-- ACTIVATE SUB 2
 50 GO SUB 500 <-- ACTIVATE SUB 1
 55 FOR L=1 TO 10
 60
 65 GO SUB 800 <-- ACTIVATE SUB 4
 70
 75 PRINT TAB(13);
 80 FOR S=1 TO 4
 85 PRINT "*";
 90 NEXT S
 95 PRINT TAB(25);
100
105 GO SUB 800 <-- ACTIVATE SUB 4
110
115 PRINT
120 NEXT L
125 GO SUB 600 <-- ACTIVATE SUB 2
130 GO SUB 700 <-- ACTIVATE SUB 3
135 GO SUB 500 <-- ACTIVATE SUB 1
140 GO SUB 700 <-- ACTIVATE SUB 3
145 STOP
500 FOR L=1 TO 5
505 FOR S=1 TO 30
510 PRINT "*";
515 NEXT S
520 PRINT
525 NEXT L
530 RETURN
600 FOR L=1 TO 5
605 PRINT
610 NEXT L
615 RETURN
700 FOR L=1 TO 5
705 PRINT TAB(25);
710
715 GO SUB 800 <-- ACTIVATE SUB 4
720
725 PRINT
730 NEXT L
735 RETURN
800 FOR S=1 TO 5
805 PRINT "*";
810 NEXT S
815 RETURN
900 END
```

SUBROUTINE 1

SUBROUTINE 2

SUBROUTINE 3

SUBROUTINE 4

EXERCISE SET 4.4

On-line:

1. Write a program to print out the word HELLO or any other word you choose. Your program should

demonstrate that you understand subroutines by using GO SUB and RETURN as often as possible.

4.5 SNOOPY: AN IRREGULAR DESIGN

Look at the picture of Snoopy at the beginning of this chapter. It contains few if any repetitive patterns. There are no expressions for the number of blanks and stars on each line. No subroutines can be established. You must construct the design line by line. For example, to draw line 2 you write:

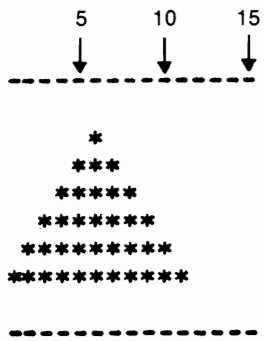
```
2 PRINT "          *       **"
```

Taking this line for line approach, your program would consist of 45 lines: 44 PRINT statements, one for each of the 44 design lines, plus an END statement. One mistake in locating a star and you must retype the entire PRINT statement.

There is, however, another approach to drawing Snoopy or any other picture. This new approach requires two new BASIC words: READ and DATA. You still have to count the number of blanks and stars to be printed on each design line, but correcting for mistakes is easier. Furthermore, once your drawing program is in the computer, you just input new data for each line of a new design. Moreover, you can store on paper tape or on cassette data for a variety of designs and simply load the data for the particular design you want drawn.

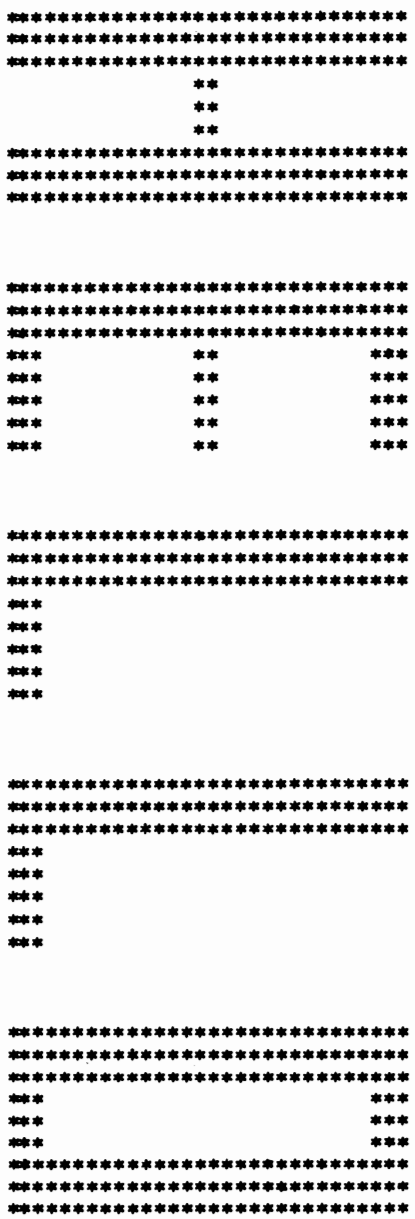
4.5.1 DRAWING USING READ AND DATA

For the sake of simplicity, let us look again at a regular design, the triangle.



As shown earlier, we begin by making a table showing the number of blanks and stars required on each line.

Line Number	Number of Blanks	Number of Stars
1	5	1
2	4	3
3	3	5
4	2	7
5	1	9
6	0	11



At this point we depart from the method used earlier and introduce two new variables: B representing the number of blanks in a line, and S representing the number of stars in a line. Thus, for design line 1, B = 5 and S = 1; for line 2, B = 4 and S = 3; for line 3, B = 3 and S = 5; etc. We'll now write a set of steps to draw design line 1. In the following routine, the C loop prints the number of stars represented by S.

```
10 LET B=5
20 LET S=1
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*";
60 NEXT C
70 PRINT
```

Look at line 30. The command TAB(B); instructs the computer to leave B blanks from the left margin. In the case of design line 1, the computer leaves five blanks. The semicolon indicates there is more to come after the blanks. Look at the C loop, lines 40, 50 and 60. This loop will print S stars. In the case of design line 1, the computer prints one star. Again, the semicolon in line 50 signals the computer to continue printing stars, if there are any, on the same line. In this case, however, there is only one star. When the C loop is satisfied, that is when S stars have been printed, the computer goes to the next statement in the program, line 70, the blank PRINT statement. The blank PRINT statement signals the computer to terminate printing on design line 1 and to advance to design line 2. To draw design line 2, we redefine B as 4 and S as 3.

```
 80 LET B=4
 90 LET S=3
100 PRINT TAB(B);
110 FOR C=1 TO S
120 PRINT "*";
130 NEXT C
140 PRINT
```

To draw design line 3, we redefine B as 3 and S as 5.

```
150 LET B=3
160 LET S=5
170 PRINT TAB(B);
180 FOR C=1 TO S
190 PRINT "*";
200 NEXT C
210 PRINT
```

We could continue, writing a similar set of seven steps to draw each of the remaining three lines of the design. But that would make a long program of 42 lines, seven program lines for each of the six design lines. Fortunately, the program can be shortened by putting five of each set of seven program lines in a subroutine:

```
PRINT TAB(B);
FOR C = 1 TO S
PRINT "*";
NEXT C
PRINT
```

Here is the program to draw the triangle. The subroutine begins at line 500.

```
 10 LET B=5      ← DATA FOR DESIGN LINE 1
 20 LET S=1
 30 GO SUB 500
 40 LET B=4      ← DATA FOR DESIGN LINE 2
 50 LET S=3
 60 GO SUB 500
 70 LET B=3      ← DATA FOR DESIGN LINE 3
 80 LET S=5
 90 GO SUB 500
100 LET B=2      ← DATA FOR DESIGN LINE 4
110 LET S=7
120 GO SUB 500
130 LET B=1      ← DATA FOR DESIGN LINE 5
140 LET S=9
150 GO SUB 500
160 LET B=0      ← DATA FOR DESIGN LINE 6
170 LET S=11
180 GO SUB 500
190 STOP
500 PRINT TAB(B);          DRAW
510 FOR C=1 TO S           LINE
520 PRINT "*";
530 NEXT C
540 PRINT
550 RETURN
600 END
```

TRIANGLE/SUB

PROGRAM ***OUTPUT***

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

The above program requires twelve LET statements for the assignment of data to B and S. Let me show you a more concise way to assign data to B and S by using the new BASIC words READ and DATA. We begin the new program by building a FOR-NEXT loop. Since there are six design lines to be drawn we write:

```
FOR L = 1 TO 6
NEXT L
```

Next we make a list of the values of B and S in the order in which they occurred in TRIANGLE/SUB.

5, 1, 4, 3, 3, 5, 2, 7, 1, 9, 0, 11

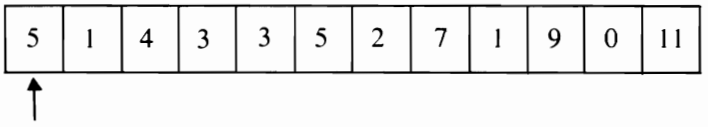
To put this list of numbers in the computer's memory, we use a statement which begins with the command DATA, placing the data line at the beginning of the program.

```
DATA 5,1,4,3,5,2,7,1,9,0,11
FOR L = 1 TO 6
NEXT L
```

When the computer runs the program, it takes the numbers in the data line and stores them in a data bank which looks like this.

5	1	4	3	3	5	2	7	1	9	0	11

The computer uses a "pointer" to indicate the next number in the data bank to be assigned to B or S. Needless to say, at the beginning of the program the pointer points to the first box, like this:



The pointer is implemented in the program by a READ statement. We write a READ statement which instructs the computer to go to the data bank, find the first number and assign it to B, and then find the second number and assign it to S. The statement, READ B, S, is put in the loop after FOR L = 1 TO 6.

```
DATA 5,1,4,3,3,5,2,7,1,9,0,11
FOR L = 1 TO 6
READ B,S
NEXT L
```

The subroutine which *draws* each line in the triangle is the same subroutine used in TRIANGLE/SUB, beginning at line 500. The complete new program looks like this:

TRIANGLE/READ/SUB

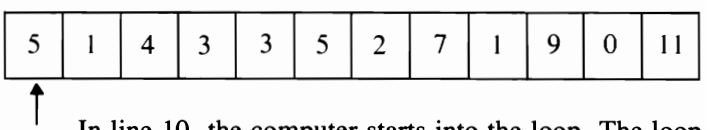
```
PROGRAM   1 DATA 5,1,4,3,3,5,2,7,1,9,0,11
         10 FOR L=1 TO 6
         20 READ B,S
         30 GO SUB 500 <-- ACTIVATE DRAW LINE
         40 NEXT L
         50 STOP
        500 PRINT TAB(B);
        510 FOR C=1 TO S
        520 PRINT "*";
        530 NEXT C
        540 PRINT
        550 RETURN
        600 END
```

6 LINES

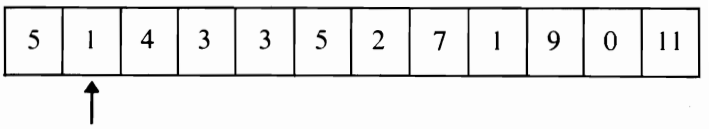
DRAW LINE

```
OUTPUT   *
        ***
       *****
      *******
     *********
    ***********
```

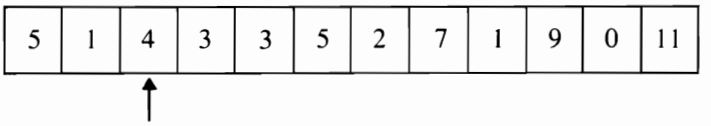
Let's play computer. In line 1, we enter the data into the computer's memory.



In line 10, the computer starts into the loop. The loop counter, L, is 1. Then the computer executes 20 READ B, S. This command instructs the computer to go to the data bank, locate the pointer, and assign to the variable B the number in the box above the pointer, B = 5. The pointer is then moved one box to the right.



So much for B. The comma after B in the statement, READ B,S, signals the computer there is another variable in the READ statement. In this case the next variable is S. The computer goes again to the data bank and locates the number in the box above the pointer and assigns this to S. The pointer is then moved one box to the right.

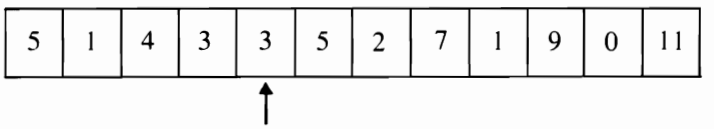


After the READ statement has been executed the first time, the values of L, B, and S are:

L	B	S
1	5	1

The next step in the program is 30 GO SUB 500. This directs the computer to go to the subroutine and draw a line using the above values of B and S. When the subroutine is completed, the computer returns to the main program and the NEXT L is executed, L = 2.

The computer starts around the loop again. On the second execution of READ B, S, the computer goes to the data bank, locates the pointer and assigns the value in the box above the pointer to B. Again the pointer is moved one box to the right. The value in this box is assigned to S and the pointer is moved to the right once more.

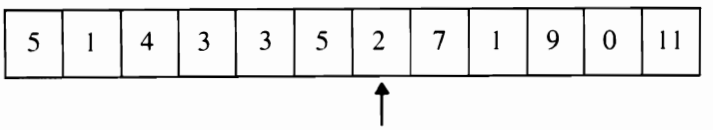


Having executed the READ statement for the second time, the values of L, B, and S are:

L	B	S
2	4	3

The subroutine is activated and design line 2 is drawn.

The computer returns to the main program and the NEXT L is executed, starting the computer around the loop for the third time. READ B, S is executed, after which the pointer is located at:

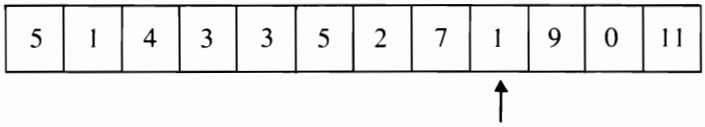


And the values of L, B, and S are:

L	B	S
3	3	5

The subroutine is activated again and line 3 is drawn.

The computer returns to the main program and the NEXT L is generated. READ B, S is executed. The pointer now has been moved to:

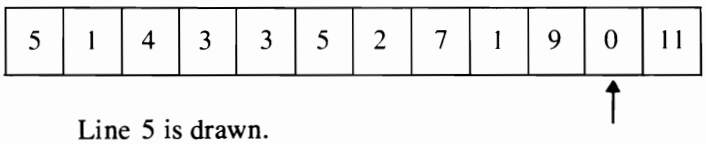


And the values of L, B, and S are:

L	B	S
4	2	7

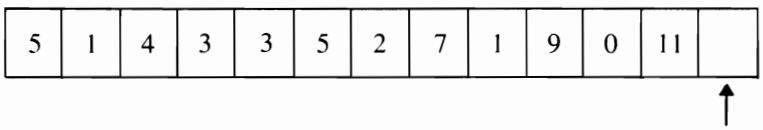
Line 4 is drawn.

The NEXT L is generated, and the computer starts around the loop again. B and S are assigned the values 1 and 9, and line 5 is drawn. While executing READ B, S, the pointer is moved to:



Line 5 is drawn.

The NEXT L is generated. The last two values in the data bank, 0 and 11, are assigned to B and S. There are no more numbers in the bank. The pointer now points to an empty box.

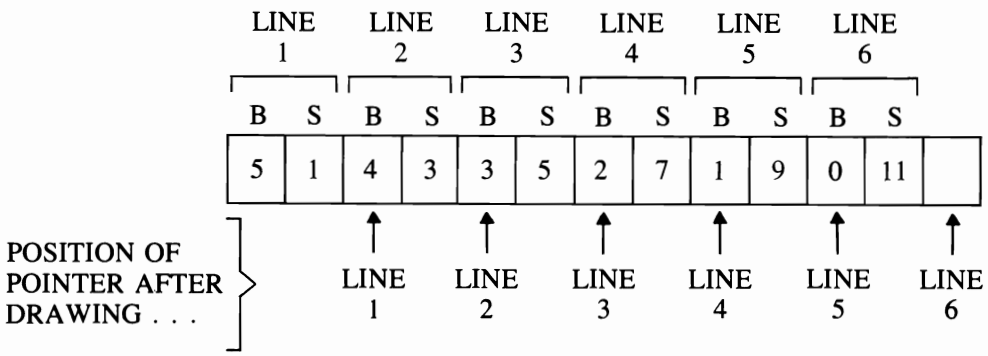


The values of L, B, and S are:

L	B	S
6	0	11

Line 6 is drawn.

There is no next L. The loop has been satisfied and the program goes to the next step, 50 STOP. To sum up, the pointer has travelled from left to right across the data bank as shown here.



So much for playing computer with TRIANGLE/READ/SUB. Since the "draw" subroutine is used only once, why not put it in the main program, eliminating the GO SUB, RETURN and STOP statements?

TRIANGLE/READ/NO SUB

PROGRAM

```
  1 DATA 5,1,4,3,3,5,2,7,1,9,0,11
 10 FOR L=1 TO 6
 20 READ B,S
 30 PRINT TAB(B);
 40 FOR C=1 TO S
 50 PRINT "*";
 60 NEXT C
 70 PRINT
 80 NEXT L
 90 END
```

6 LINES — DRAW LINE

The data line may be placed anywhere in the program before the END statement. The DATA statement is not executed in sequence with the other statements, like LET or PRINT. The line just holds the numbers which are to be stored in the computer's memory. For example, in the program above, the DATA line is placed in line 1, outside the loop. In the program below it is placed in the loop.

TRIANGLE/READ/IN-LOOP

PROGRAM

```
 10 FOR L=1 TO 6
 11 DATA 5,1,4,3,3,5,2,7,1,9,0,11
 20 READ B,S
 30 PRINT TAB(B);
 40 FOR C=1 TO S
 50 PRINT "*";
 60 NEXT C
 70 PRINT
 80 NEXT L
 90 END
```

Here it is again placed outside the loop, directly before the END statement.

TRIANGLE/READ/OUT-LOOP

PROGRAM

```
 10 FOR L=1 TO 6
 20 READ B,S
 30 PRINT TAB(B);
 40 FOR C=1 TO S
 50 PRINT "*";
 60 NEXT C
 70 PRINT
 80 NEXT L
 81 DATA 5,1,4,3,3,5,2,7,1,9,0,11
 90 END
```

In general, it's desirable to place the DATA line outside the loop to keep the loop uncluttered.

In the above programs all the data are listed on one line. In the case of a more complicated design, like SNOOPY, imagine the difficulty you would have in quickly finding an incorrect value for B or S, should an error occur in the location of a blank or star. To make it easier to correct the data, it is sometimes desirable to put

the data for each design line on a separate program line with a corresponding line number, like this:

TRIANGLE/READ ***PROGRAM***

```
1 DATA 5,1
2 DATA 4,3
3 DATA 3,5
4 DATA 2,7
5 DATA 1,9
6 DATA 0,11
10 FOR L=1 TO 6
20 READ B,S
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*";
60 NEXT C
70 PRINT
80 NEXT L
90 END
```

EXERCISE SET 4.5.1

Off-line:

1. Here is the program TRIANGLE/READ with the data lines omitted.

```
10 FOR L=1 TO 6
20 READ B,S
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*";
60 NEXT C
70 PRINT
80 NEXT L
90 END
```

A. Add the following DATA line to the above program, then play computer to determine the design that would be drawn.

85 DATA 5,4,5,4,5,4,5,4,5,4,5,4

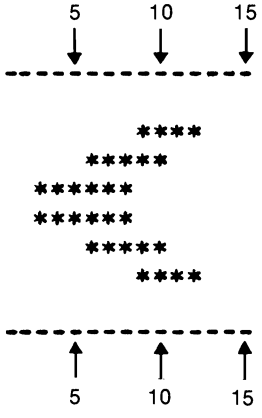
B. Substitute the following DATA line in the above program. Play computer to determine the design that would be drawn.

85 DATA 1,5,2,5,3,5,4,5,5,5,6,5

C. Substitute the following DATA line in the above program and play computer.

90 DATA 0,11,1,9,2,7,3,5,4,3,5,1

D. Substitute in the above program a DATA line which will cause the computer to draw this design at left.



4.5.2 DRAWING USING FLAG 100

Let's try to develop a general purpose drawing routine in which *all* of the design characteristics, including the number of design lines, are specified in the DATA lines. Look at this listing of TRIANGLE/READ.

PROGRAM TRIANGLE/READ

```
 1 DATA 5,1
 2 DATA 4,3
 3 DATA 3,5
 4 DATA 2,7
 5 DATA 1,9
 6 DATA 0,11
10 FOR L=1 TO 6
20 READ B,S
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*";
60 NEXT C
70 PRINT
80 NEXT L
90 END
```

The above program uses a FOR-NEXT loop to control the number of design lines. We can eliminate the FOR-NEXT loop by using instead the number flag 100 to signal the computer when to stop drawing. Make the following modifications in the above program:

(1) Insert: 7 DATA 100
(2) Delete: 10 FOR L = 1 TO 6
(3) Replace 80 with: 80 GO TO 20
(4) Insert: 25 IF B = 100 THEN 90

Here is the above program, TRIANGLE/READ, with the modifications. A typical run follows.

TRIANGLE/OUT OF DATA

```
 1 DATA 5,1
 2 DATA 4,3
 3 DATA 3,5
 4 DATA 2,7
 5 DATA 1,9
 6 DATA 0,11
 7 DATA 100
20 READ B,S
25 IF B=100 THEN 90
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*;
60 NEXT C
70 PRINT
80 GO TO 20
90 END
```

PROGRAM

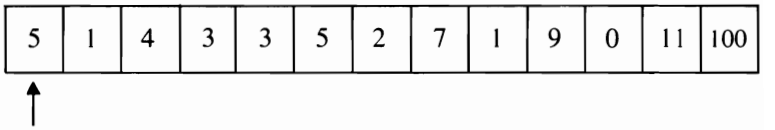
```
   *
  ***
 *****
*******
*********
***********
```

OUTPUT

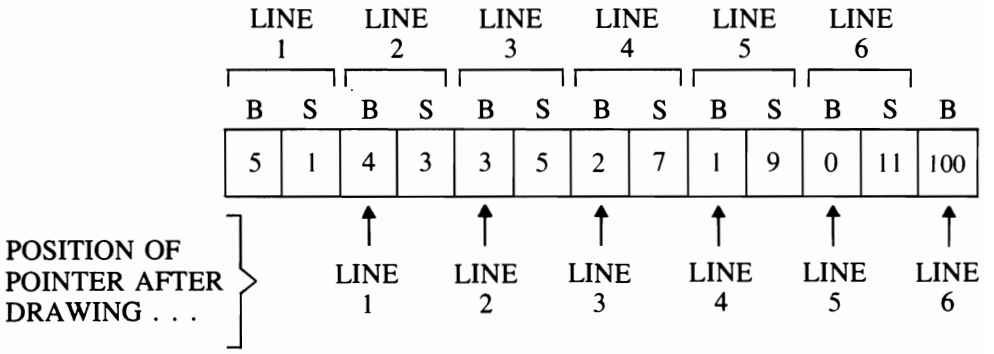
```
OUT OF DATA IN LINE 20
```

Look at the output. The triangle is fine, but we certainly don't want the message OUT OF DATA IN LINE 20 to be part of the design. Let's play computer to find out why that line appeared. When scanned by the computer,

the seven DATA lines establish the following data bank for the design.



The following diagram shows the progress of the pointer from left to right as each design line is drawn. This is how it looks to the computer.



Look at the last position of the pointer in the above diagram. It is pointing to the number 100. Why doesn't the computer print 100 blanks? It doesn't print 100 blanks because the statement 20 READ B, S can only be satisfied if there are *two* numbers available to read in the data bank. There's only one number left. The computer is stymied! It sends out a message for help, "OUT OF DATA IN LINE 20." To avoid this difficulty, instruct the computer to read only one number at a time from the data bank instead of a pair. Separate the message READ B, S, into two READ statements like this:

```
20 READ B
26 READ S
```

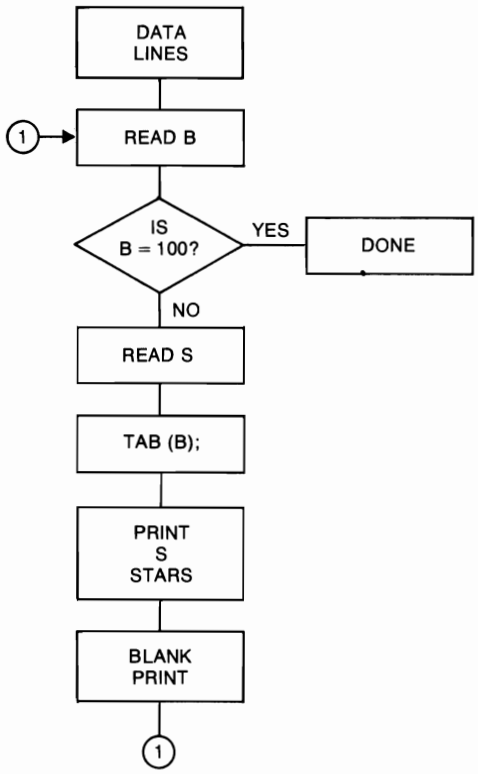
Here is the revised program.

TRIANGLE/FLAG 100 ***PROGRAM***

```
 1 DATA 5,1
 2 DATA 4,3
 3 DATA 3,5
 4 DATA 2,7
 5 DATA 1,9
 6 DATA 0,11
 7 DATA 100
20 READ B
25 IF B=100 THEN 90 ← TEST FOR FLAG 100
26 READ S
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*";
60 NEXT C
70 PRINT
80 GO TO 20
90 END
```

DRAW LINE

Here is a flow chart that shows the function of the IF-THEN statement in testing for FLAG 100.



DRAW FLAG/100 FLOW CHART

EXERCISE SET 4.5.2

Off-line:

1. Play computer with this program and draw the output.

```
 1 DATA 0,10,8,1,7,1,6,1
 2 DATA 5,1,4,1,3,1,2,1
 3 DATA 1,1,0,10,100
20 READ B
25 IF B=100 THEN 90
26 READ S
30 PRINT TAB(B);
40 FOR C=1 TO S
50 PRINT "*";
60 NEXT C
70 PRINT
80 GO TO 20
90 END
```

4.5.3 DRAWING USING FLAG 99 AND TAB COUNTER

Compare the designs at right. Each design line in the triangle consists of one set of blanks, followed by one set of stars. On the other hand, each line in the design HI is composed of two or three sets of blanks, separated by one or more stars.

TRIANGLE

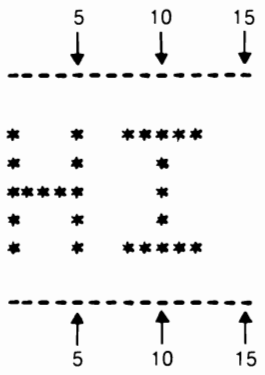
```
*
***
*****
*******
*********
***********
```

HI

```
          5         10        15
          |         |         |
----------------------------------
*   *   *****
*   *     *
*****     *
*   *     *
*   *   *****
----------------------------------
          |         |         |
          5         10        15
```

DRAW/FLAG 100

HI



We can't use the old routine TRIANGLE/FLAG 100 to draw the new design because each time the "star printing" loop, lines 40, 50 and 60, is satisfied, the blank PRINT statement in line 70 switches the output to the next line. Let's fix up the old routine so that the computer will not go to a new design line until it completes the one it is drawing. To signal the computer that it is at the end of a design line, let's insert a new flag, 99.

We think of each design line as composed of pairs of blanks (B) and stars (S). For example, the first pair in the first design line is B = 0, S = 1. The second pair is B = 3, S = 1. The third and final pair is B = 2, S = 5. These data are placed in program line 1, with 99 at the end.

1 DATA 0,1,3,1,2,5,99

The next design line is made up of the following pairs: 0,1; 3,1; 4,1, and is represented by the following DATA line.

2 DATA 0,1,3,1,4,1,99

Design line 3 consists of the following pairs: 0,5; 4,1, and is represented by the following DATA line:

3 DATA 0,5,4,1,99

Count the blanks and stars in the design lines 4 and 5 and complete the corresponding DATA lines below:

4 DATA

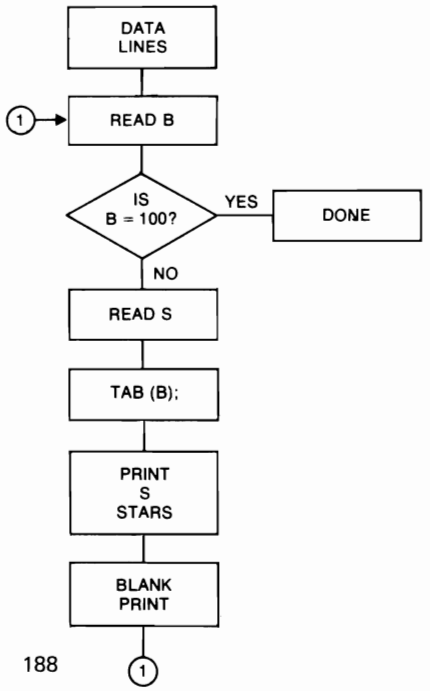
5 DATA

That completes the five-line design. To signal the computer that the design is finished, insert

6 DATA 100

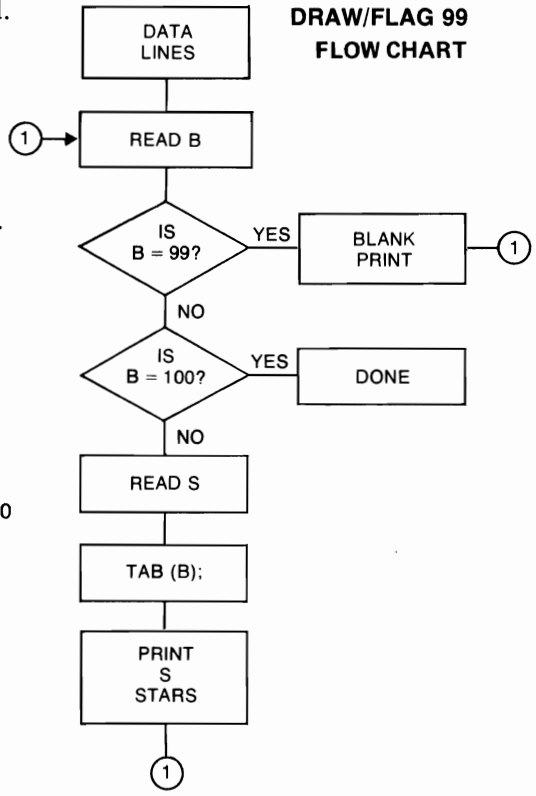
Adding 99 to the end of each data line is only half the job. A test for 99 must also be inserted in the program, after the computer reads a value for B. Look again at the flow chart for DRAW/FLAG 100.

FLOW CHART



Where should the test for this second flag be inserted? The best place is immediately after the computer reads a value for B. After the READ B box, insert a new diamond containing the question IS B = 99? If the

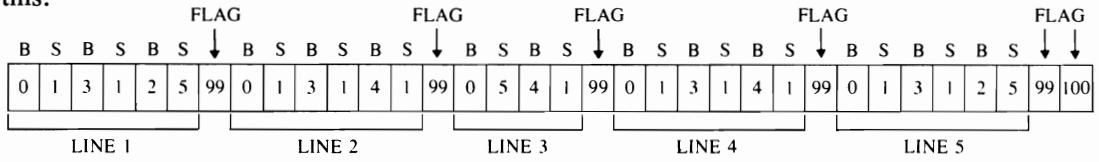
answer is YES, the blank PRINT statement is executed.



From this flow chart the following program is encoded.

```
  1 DATA 0,1,3,1,2,5,99      HI/FLAG 99
  2 DATA 0,1,3,1,4,1,99      PROGRAM
  3 DATA 0,5,4,1,99
  4 DATA 9,1,3,1,4,1,99
  5 DATA 0,1,3,1,2,5,99
  6 DATA 100
 10 READ B
 20 IF B=99 THEN 100 ← TEST FOR FLAG 99
 30 IF B=100 THEN 120 ← TEST FOR FLAG 100
 40 READ S
 50 PRINT TAB(B);
 60 FOR C=1 TO S
 70 PRINT "*";
 80 NEXT C
 90 GO TO 10
100 PRINT
110 GO TO 10
120 END
```

Let's play computer to verify that the new FLAG 99 routine functions properly. The data bank looks like this:



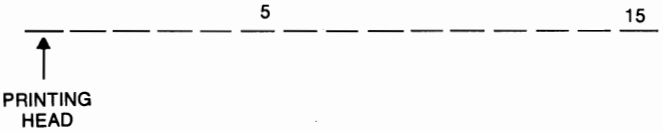
The action begins at program line 10 with READ B. B is assigned the value 0. That is, the value 0 is inserted in a memory cell labeled B:

B
0

B is tested for 99 and 100. Since it is neither of these, the computer drops to 40 READ S. S is assigned the value 1. The value 1 is inserted in a memory cell labeled S:

B	S
0	1

Since B and S data only occur in pairs, it is unnecessary to test S for 99 or 100, so the computer drops from line 40 to 50 PRINT TAB(B);. In the first pair of data B is 0, so the computer leaves 0 blanks from the left margin. If the computer had ordered the printer to print a character in space 1, even a blank, the print head would then have shifted automatically one space to the right, ready to print in space 2. In this case, however, because B is 0, the printing head just doesn't move!



Having executed line 50, the computer drops to line 60 and starts the C loop. S is 1, so a single star is drawn in space 1 and the printing head automatically shifts one space to the right, ready to print in space 2:

B
3

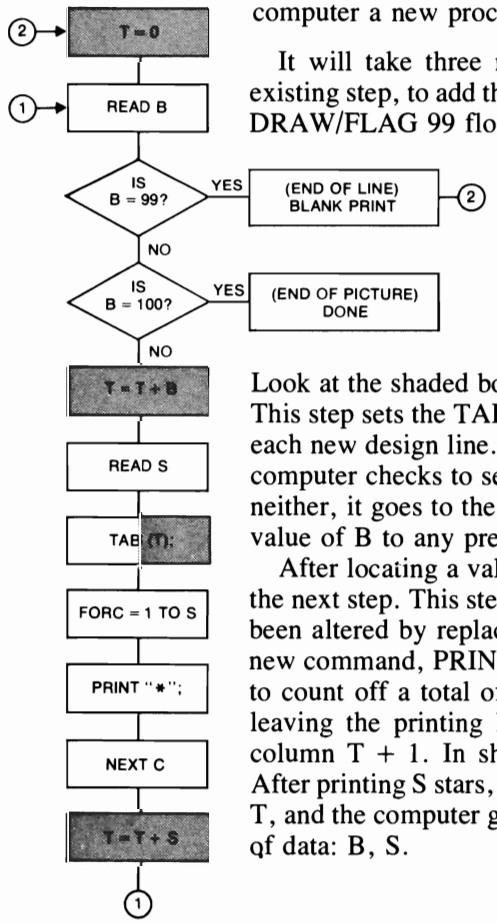
Having completed the C loop, the computer drops to 90 GO TO 10. The computer returns to line 10 and looks for a new pair of data. When 10 READ B is executed, B is assigned 3, the third number in the data bank.

B	S
3	1

B is tested for 99 and 100. Since it is neither, the computer drops to 40 READ S and assigns to S the value 1, the fourth number in the data bank.

Having found a new pair of data in lines 10 through 40, the computer drops to 50 PRINT TAB(B);. In this pair, the number of blanks, B, is 3. That means the distance from the last star is three spaces. The command TAB(B), however, instructs the computer to count off three spaces from the left margin, *not* from the last star. To make the computer TAB over from the last star, rather than from the left margin, we need to set up in the computer a new procedure called a "TAB counter."

DRAW FLOW CHART



It will take three new steps, plus a change in an existing step, to add the TAB counter, call it T, into the DRAW/FLAG 99 flow chart.

Look at the shaded box at the top of the chart: T = 0. This step sets the TAB counter at 0 at the beginning of each new design line. After locating a value for B, the computer checks to see if it is either 99 or 100. If it is neither, it goes to the second shaded box and adds the value of B to any previous value of T.

After locating a value for S, the computer moves to the next step. This step was in the earlier chart, but has been altered by replacing TAB(B) with TAB(T). The new command, PRINT TAB(T);, causes the computer to count off a total of T spaces from the left margin, leaving the printing head ready to print in space or column T + 1. In short, T is a cumulative counter. After printing S stars, the last new step adds S spaces to T, and the computer goes back to look for the next pair of data: B, S.

This flow chart is encoded as follows:

```
→10 LET T=0                    DRAW
 20 READ B
 30 IF B=99 THEN 130
 40 IF B=100 THEN 150
→50 LET T=T+B
 60 READ S
→70 PRINT TAB(T);
 80 FOR C=1 TO S
 90 PRINT "*";
 100 NEXT C
→110 LET T=T+S
 120 GO TO 20
 130 PRINT
 140 GO TO 10
 150 END
```

Lines 10, 50, 70 and 110 correspond to the shaded boxes in the flow chart and implement the TAB counter. As before, if this general purpose program is to draw HI, we add six DATA lines:

```
1 DATA 0,1,3,1,2,5,99
2 DATA 0,1,3,1,4,1,99
3 DATA 0,5,4,1,99
4 DATA 0,1,3,1,4,1,99
5 DATA 0,1,3,1,2,5,99
6 DATA 100
```

DRAW/HI

This gives us:

```
 1 DATA 0,1,3,1,2,5,99
 2 DATA 0,1,3,1,4,1,99
 3 DATA 0,5,4,1,99
 4 DATA 0,1,3,1,4,1,99
 5 DATA 0,1,3,1,2,5,99
 6 DATA 100
 10 LET T=0
 20 READ B
 30 IF B=99 THEN 130
 40 IF B=100 THEN 150
 50 LET T=T+B
 60 READ S
 70 PRINT TAB(T);
 80 FOR C=1 TO S
 90 PRINT "*";
 100 NEXT C
 110 LET T=T+S
 120 GO TO 20
 130 PRINT
 140 GO TO 10
 150 END
```

Here is a typical run.

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

You now have a general purpose program which will draw any design. Just draw your design on graph paper, count the number of blanks and stars in each design line, and match one DATA line to each design line. You can quickly correct any error in a design line by locating and repairing the corresponding DATA line.

EXERCISE SET 4.5.3

1. Create your own design and program the computer to draw it. Keep the design small, particularly if you're on a small computer, for the numbers in your data line take considerable storage space. If I were you, I wouldn't try SNOOPY just yet.

PLAYING BIG WHEEL

```
DO YOU WANT INSTRUCTIONS? YES


IMAGINE A LARGE CARNIVAL WHEEL, 6 FEET IN DIAMETER,
DIVIDED INTO 9 EQUAL PIE-SHAPED PIECES. THREE SECTIONS
ARE NUMBERED 1, THREE SECTIONS ARE NUMBERED 2, TWO
SECTIONS ARE NUMBERED 3, AND ONE SECTION IS NUMBERED 4.

THE WHEEL WILL BE SPUN AND THEN YOU WILL HAVE AN
OPPORTUNITY TO BET ON ONE OF THE FOUR NUMBERS
BEFORE THE WHEEL COMES TO REST.

AT THE START OF EACH GAME, YOU WILL RECEIVE
A CERTAIN AMOUNT OF MONEY TO BET IN WHOLE
DOLLAR AMOUNTS.

SECTION NUMBERS PAYOFF AS FOLLOWS:
1  PAYS  2  TIMES YOUR BET.
2  PAYS  2  TIMES YOUR BET.
3  PAYS  3.5 TIMES YOUR BET.
4  PAYS  8  TIMES YOUR BET.

WHEN YOU WANT TO QUIT(IF YOU DON'T GO BROKE FIRST)
TYPE 99 WHEN ASKED TO PICK 1,2,3,4.


LET'S GO.

PLEASE TELL ME YOUR NAME? PETER


YOU HAVE 80 DOLLARS TO PLAY WITH.


---WHEEL IS SPINNING
PICK 1,2,3,4? 1
BET? 20
---WHEEL STOPPED AT 2
YOU LOSE $ 20
WHAT BAD LUCK.
YOU NOW HAVE 60 DOLLARS.


---WHEEL IS SPINNING
PICK 1,2,3,4? 1
BET? 20
---WHEEL STOPPED AT 1
YOU WIN $ 40
 YOU NOW HAVE 100 DOLLARS.


---WHEEL IS SPINNING
PICK 1,2,3,4? 99
HOPE YOU HAD FUN PETER. COME BACK SOON.
```

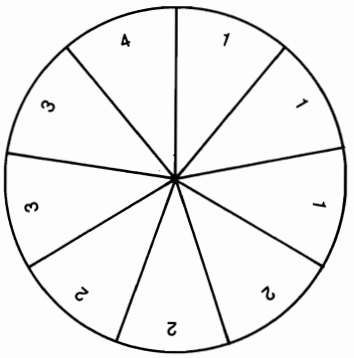
5

PLAYING BIG WHEEL

The programs we have written so far have been relatively short, no more than 60 steps. You can imagine how confused you might become when writing a program containing 100 or more steps, unless you follow some standard procedure for organizing the program. In this chapter we will demonstrate such a procedure by writing a relatively large program called BIG WHEEL, which will bring together many BASIC words and most of the programming concepts introduced in the first half of this book.

5.1 FIGURING THE PAYOFF

The game is played using a big wheel like this:



When the wheel is spun, the user is asked to pick the number, 1, 2, 3, or 4, at which he thinks the wheel will come to rest. At the same time he is asked to bet an amount of money on this choice. If the wheel comes to rest on his number, the user wins; otherwise he loses. Do you understand the rules, the sequence of play, and the procedure for determining a winner? It is important that you know these things before trying to program any game.

When the wheel spins, which number will you bet on? Because of the construction of the wheel, not all numbers are equally likely to occur. The wheel is divided into nine equal pie-shaped pieces. Three pieces are numbered 1, three pieces numbered 2, two pieces numbered 3, and one piece numbered 4. The numbers 1 and 2 are therefore more likely to occur than 3 or 4. Let's be more precise in our description of the possible outcomes.

If the wheel is spun, it may stop at any one of the pie-shaped pieces. We assume that the wheel will not come to rest on a line dividing two pieces of pie. Thus,

there are nine possible outcomes, three of which produce 1, three of which produce 2, two of which produce 3, and one which produces 4. The probability that the wheel will stop at the number 1 is greater than the probability that the wheel will stop at 4. The probability that the wheel will stop at 1 is the number of possible stopping places that produce 1, divided by the total number of stopping places on the wheel. Since 1 occurs three times on the wheel and the wheel has nine stopping places, then the probability that the wheel will stop at 1 is 3/9 or 1/3. Similarly, the probability of the wheel stopping at 2 is 3/9 or 1/3. The probability the wheel will stop at 3 is 2/9, and at 4, 1/9. Here is a table listing the probability of each outcome:

OUTCOME	PROBABILITY
Number 1	1/3
Number 2	1/3
Number 3	2/9
Number 4	1/9

Now for the payoff. In the long run do you want the player to win, lose, or come out even? Let's set it up so that, after a very large number of games, the player comes out even. A game set up this way is called a *fair game*. Since the outcomes (1, 2, 3, or 4) are not equally likely, we have to make some adjustment in the payoffs to insure that, in the long run, the player's winnings equal his losses. In short, he must be paid more for a low probability outcome, one which occurs infrequently, than he is for a high probability outcome, that is one which occurs more often.

Here is a table which summarizes the expected frequency with which he will win or lose when he picks each number.

NUMBER BET ON	NUMBER WAYS TO WIN	NUMBER WAYS TO LOSE
1	3	6
2	3	6
3	2	7
4	1	8

Suppose a stubborn player puts his money on number 1 for 900 consecutive spins. Theory tells us he will lose 600 times, and win 300 times. If he bets $1 on each spin, his losses will total $600. How much does he have to collect each time he wins in order to come out even? To offset his losses, he must collect $2 each time he wins, or two times the amount of his $1 bet.

Suppose another player with a one-track mind plays number 2 for 900 consecutive spins. Theoretically at least, he will also lose 600 times, and win 300 times. If he bets $1 on each spin, his losses will total $600. Like the player who played number 1, he must also collect $2

each time he wins, or two times the amount of his bet. In short, the payoff factor when the player bets on number 1 or number 2 is 2.

Here is a table showing the payoff factor for two outcomes.

NUMBER BET ON	NUMBER WAYS TO WIN	NUMBER WAYS TO LOSE	PAYOFF FACTOR
1	3	6	2
2	3	6	2
3	2	7	
4	1	8	

Suppose our stubborn player plays number 3 for 900 spins. According to the table he can expect to lose 700 times and win 200 times. If he bets $1 each time, how much does he have to collect each time he wins in order to offset his losses? Right, $3.50. So the payoff factor any time he bets on number 3 is 3.5.

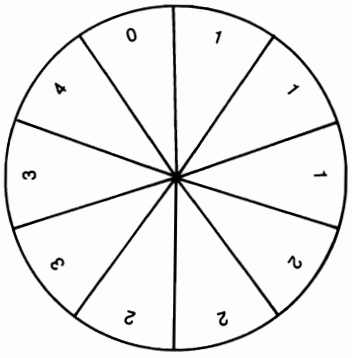
Finally, if he plays number 4 for 900 spins, he can expect to lose 800 times, and win 100 times. If he bets $1 each time, how much does he have to collect each time he wins? The payoff factor is 8.

This table lists all four payoff factors.

NUMBER BET ON	NUMBER WAYS TO WIN	NUMBER WAYS TO LOSE	PAYOFF FACTOR
1	3	6	2
2	3	6	2
3	2	7	3.5
4	1	8	8

Look at the fourth column. The payoff factor for each outcome can also be obtained by dividing the corresponding number in column three by the number in column two.

In some real life gambling situations, such as roulette, the operator of the game collects all bets if the wheel stops on a particular number. For example, here is a wheel to which we have added a section bearing the number 0.



You could set up a new game with this new wheel in which the player is not allowed to bet on the number 0. If the wheel stops on 0, the player always loses. He

cannot win. The new game is summarized in this table.

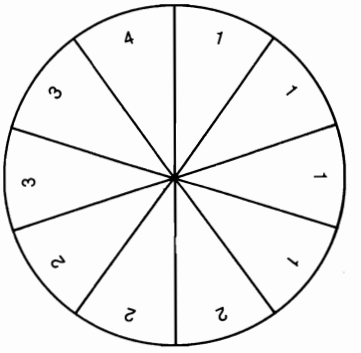
SECTION NUMBER	NUMBER WAYS TO WIN	NUMBER WAYS TO LOSE
0	0	10
1	3	7
2	3	7
3	2	8
4	1	9

In this new game, if the stubborn player bets on number 1 for 1000 consecutive spins, he can expect to lose 700 times and win 300 times. If he bets $1 each time, his losses will total $700. If the payoff factor for this number is still 2, his winnings can only total $600, leaving him $100 in the hole. Using the new 10-section wheel and the old payoff schedule, do we still have a fair game? No, we don't. To make it fair we would have to compute a new payoff schedule.

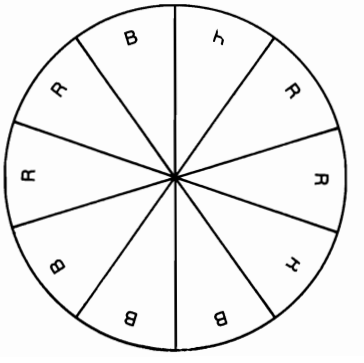
EXERCISE SET 5.1

Off-line:

1. Set up a payoff schedule for a fair game played with the following wheel. The player can bet on any number.



2. Set up a payoff schedule for a fair game in which the player bets whether the above wheel will come to rest on an even or odd number.
3. Here is a wheel divided into segments that are colored either red(R) or black(B). Set up a payoff schedule for a fair game in which the player bets on whether the wheel will come to rest on either red or black.



5.2 WRITING AND TESTING THE PROGRAM

The first step in writing the program BIG WHEEL is to list the sequence of events in a typical game:

1. Ask player to pick a number, 1, 2, 3, or 4.
2. Ask player for amount of his bet.
3. Spin the wheel.
4. Stop the wheel at a number.
5. Did player win?
6. If so, compute the payoff.
7. If not, collect his bet.

The next step is to visualize the output at the terminal. Here is some typical dialogue that may occur after the player types in RUN:

```
PICK 1, 2, 3, 4? 2 <-------- PLAYER'S NUMBER
BET? 10 <------------------- PLAYER'S BET
STOPPED AT 1 <-------------- WHEEL NUMBER
YOU LOSE $ 10 <------------- AMOUNT LOST
YOU NOW HAVE ____
```

At this point you want to tell the user how many dollars he has left, but you don't know how much he started with. So let's assume the user starts with $100. We'll tell him this at the start of the program.

```
YOU HAVE 100 DOLLARS <------- INITIAL BANKROLL
PICK 1, 2, 3, 4? 2 <-------- PLAYER'S NUMBER
BET? 10 <------------------- PLAYER'S BET
STOPPED AT 1 <-------------- WHEEL NUMBER
YOU LOSE $ 10 <------------- AMOUNT LOST
YOU NOW HAVE 90 DOLLARS <--- CURRENT BANKROLL
PICK 1,2,3,4? 1 <----------- PLAYER'S NUMBER
BET? 30 <------------------- PLAYER'S BET
STOPPED AT 1 <-------------- WHEEL NUMBER
YOU WIN $ 60 <-------------- AMOUNT WON
YOU NOW HAVE 150 DOLLARS <-- CURRENT BANKROLL
```

What about players who bet when they are bankrupt? What about players who pick a number that is not on the wheel? Certainly you'll want to do something about these possibilities later, but it is more important now to restrict yourself to the above output.

To write the program steps that will achieve the above output, you first list the variables needed in the program and the purpose of each. We need variables to represent

(1) the total amount of money the user has at any given moment,
(2) the number he selects,
(3) the amount of his bet,
(4) the number at which the wheel stops, and, not so obvious,
(5) a variable to indicate the payoff factor.

Use as names of variables letters which help you remember what they stand for:

M represents the total amount of *m*oney the user has to bet.
N represents the *n*umber selected by the user.

B represents the amount of money *b*et by the user on his number.

W represents the number at which the *w*heel comes to rest.

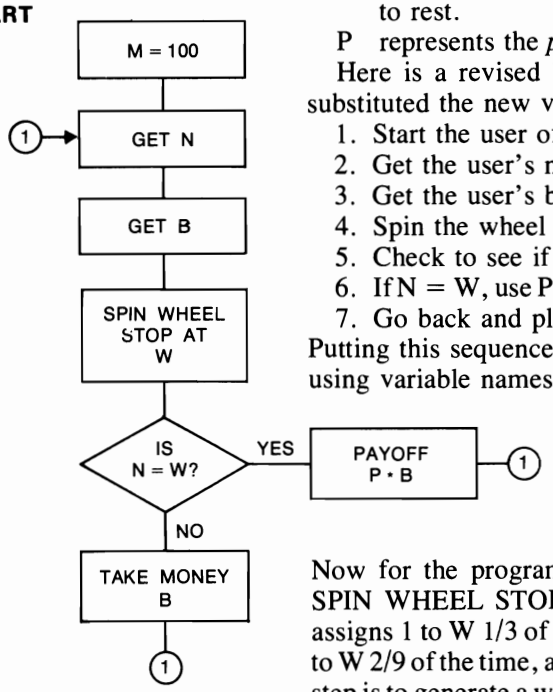
P represents the *p*ayoff factor.

Here is a revised list of events in which we have substituted the new variable names:

1. Start the user off with 100 dollars; M = 100.
2. Get the user's number, N.
3. Get the user's bet, B.
4. Spin the wheel and have it come to rest at W.
5. Check to see if N is the same as W.
6. If N = W, use P to compute payoff; if not, take B.
7. Go back and play again.

Putting this sequence of events into a flow chart, still using variable names, we have:

BIG WHEEL/ELEMENTARY FLOW CHART

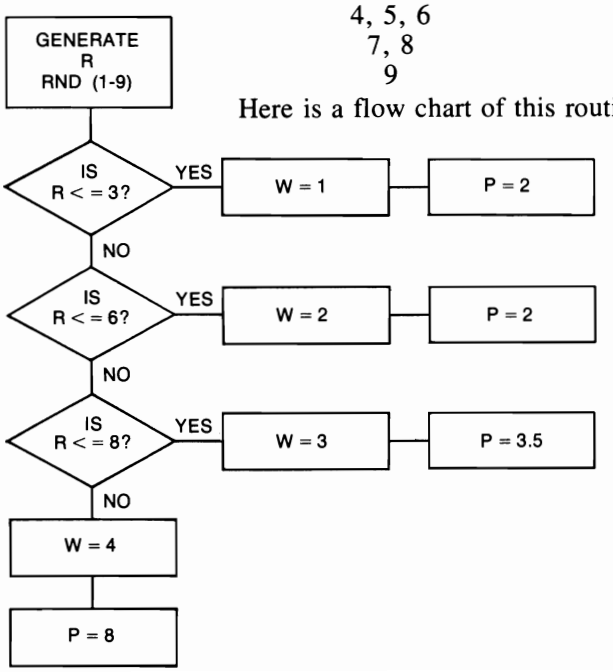


Now for the program. Let's start with the operation SPIN WHEEL STOP AT W. We need a routine that assigns 1 to W 1/3 of the time, 2 to W 1/3 of the time, 3 to W 2/9 of the time, and 4 to W 1/9 of the time. The first step is to generate a whole number R at random between 1 and 9. We then instruct the computer to output 1 if R is 1, 2, or 3; 2 if R is 4, 5, or 6; 3 if R is 7 or 8; and 4 if R is 9. Here is the correspondence between random number, R, and wheel outcome, W, summarized in a table.

RANDOM NUMBER	WHEEL NUMBER
1, 2, 3	1
4, 5, 6	2
7, 8	3
9	4

Here is a flow chart of this routine.

SPIN WHEEL STOP AT W FLOW CHART

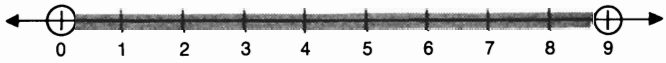


You will find that it makes your work easier if you treat this routine as a subroutine, instead of incorporating it in the main program. It is not that it is used repeatedly in the game, it is used only once, but that it makes it easier to isolate and identify individual game functions in this long program. To do this, however, we must have some idea of the general organization of the program. We must reserve some line numbers for the main program. If we reserve lines 1-199 for the main program, we can start the first subroutine at line 200.

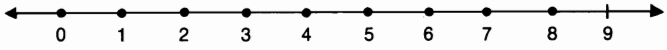
The first step in the subroutine is to write a BASIC statement which will generate a random whole number between 1 and 9 inclusive. We develop the expression as follows:

1. RND(X) gives us a number between 0 and 1 as pictured below.

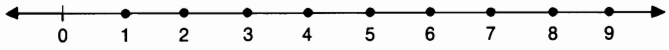
2. Expand the interval by multiplying RND(X) by 9, thus: 9*RND(X)



3. The expression to this point gives us decimal numbers, e.g.: 7.2. Use INT to transform these decimal numbers into whole numbers, thus:
   INT(9*RND(X))



4. Shift the set of numbers to the right by adding 1 to the expression, thus: INT(9*RND(X))+1



The first step of the subroutine is therefore:

200 LET R = INT(9*RND(X))+1

The remainder of the flow chart is made up of diamonds, which are translated into BASIC by using IF-THEN statements.

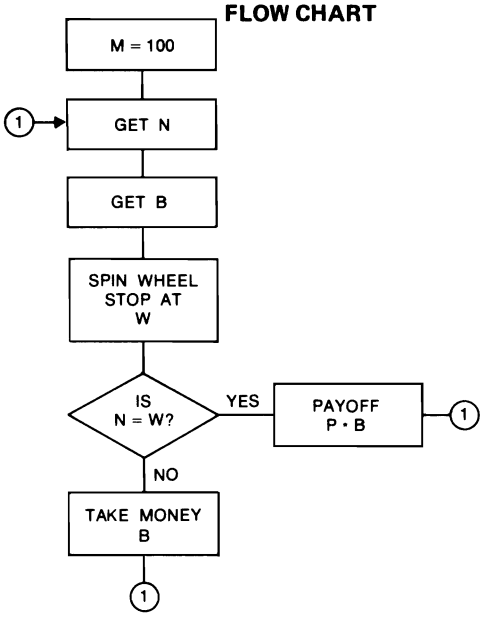
Recall that a subroutine requires a RETURN statement. This routine has four termination points, and will therefore require four RETURN statements, one at the end of each branch. Here is the complete subroutine.

SPIN WHEEL STOP AT W
SUBROUTINE

```
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

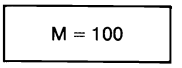
Play computer to check out the subroutine: If the computer generates a random number R = 4, at what number does the wheel come to rest and what is the payoff factor?

BIG WHEEL/ELEMENTARY FLOW CHART



Having completed the subroutine for SPIN WHEEL, STOP AT W, look again at the flow chart for the entire game.

The main program for BIG WHEEL is written as follows. The first instruction in the flow chart,



is translated into BASIC as

```
20 LET M = 100.
```

I've started with line number 20 because I want line 10 for the RANDOM statement. Usually I've placed the RANDOM statement in line 1, but I wish to reserve the first ten lines for comments which will be added later.

Following this initialization of the player's bankroll, a statement must be written to convey to the player how much money he has. We write:

```
30 PRINT "YOU HAVE" M "DOLLARS
   TO PLAY WITH."
```

The next instruction is 

It is now time to get the player's number, N, but first we tell him to pick 1, 2, 3, or 4:

```
40 PRINT "PICK 1,2,3,4";
```

The next statement, 50 INPUT N, puts a question mark on the page and instructs the computer to wait for the player to type 1, 2, 3, or 4. Notice that we added a semicolon at the end of line 40. This places the question mark after "PICK 1,2,3,4."

The next instruction is 

which must be preceded by a PRINT statement telling the user to make a bet:

```
60 PRINT "BET";
```

This is followed by:

```
70 INPUT B.
```

The next instruction, 

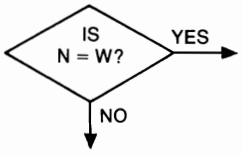
has already been encoded as a subroutine beginning at line 200. To call this routine from the main program we add:

```
80 GO SUB 200
```

When the computer returns from the subroutine with values for W and for P, the player should be told the number at which the wheel stopped:

```
90 PRINT "STOPPED AT" W
```

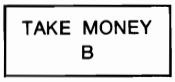
The next element in the flow chart,



is translated into BASIC using an IF-THEN statement. We write

```
100 IF N = W THEN _____
```

leaving a blank for the line number after THEN. If we take the NO exit from the diamond, we arrive at



At this point the player has lost B dollars and must be so informed by:

```
110 PRINT "YOU LOSE $" B
```

Then the amount he bet, B, must be subtracted from his bankroll, M.

```
120 LET M = M - B
```

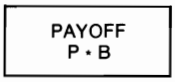
Though not indicated in the flow chart, before sending the player back to try his luck in another game, we should tell him how much he has in his bankroll.

```
130 PRINT "YOU NOW HAVE" M "DOLLARS."
```

Automatic replay is indicated in the flow chart by the number 1 in a circle at the end of the drop-down branch. This is encoded in BASIC as:

```
140 GO TO 40
```

Now for the right or YES exit from the diamond. The instruction is:



Here the player wins P times B dollars and must be so informed.

```
150 PRINT "YOU WIN $" P*B
```

Before going on to the next step in the program, recall the blank you left after the IF-THEN statement in line 100. You now can fill that blank with the number 150.

```
100 IF N = W THEN 150
```

To continue, after telling the player he won, his bet, B, must be multiplied by the payoff factor and added to his bankroll, M.

```
160 LET M = M + P*B
```

And he should be told how much he has in the bank.

```
170 PRINT "YOU NOW HAVE" M "DOL-
    LARS."
```

Just as at the end of the drop-down branch, the computer should be directed to replay the game automatically.

```
180 GO TO 40
```

The main program is now complete. Generally, when you build a program containing subroutines, a STOP

statement is required at the end of the main program to prevent the computer from running on into the subroutine section. In this case, however, the main program cannot run on into the second section, because a GO TO 40 statement has been inserted at the end of each branch.

Here is the main program as developed above.

BIG WHEEL/ELEMENTARY
MAIN PROGRAM

```
10 RANDOM
20 LET M=100
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
40 PRINT "PICK 1,2,3,4";
50 INPUT N
60 PRINT "BET";
70 INPUT B
80 GO SUB 200
90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

To the above main program we now join the subroutine SPIN WHEEL STOP AT W. We also add a RANDOM statement and an END statement. The RANDOM statement is necessary to start RND(X), used in the subroutine, at a randomly selected place in the "random number list." This is written:

```
10 RANDOM
```

Number the END statement 999 or some other large line number. This will allow us to add more subroutines later in the chapter.

```
                                                    BIG WHEEL/ELEMENTARY
MAIN PROGRAM
   10 RANDOM
   20 LET M=100
   30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
   40 PRINT "PICK 1,2,3,4";
   50 INPUT N
   60 PRINT "BET";
   70 INPUT B
   80 GO SUB 200 <-------------- ACTIVATE SPIN WHEEL
   90 PRINT "STOPPED AT" W
  100 IF N=W THEN 150
  110 PRINT "YOU LOSE $" B
  120 LET M=M-B
  130 PRINT "YOU NOW HAVE" M "DOLLARS."
  140 GO TO 40
  150 PRINT "YOU WIN $" P*B
  160 LET M=M+P*B
  170 PRINT "YOU NOW HAVE" M "DOLLARS."
  180 GO TO 40
SPIN WHEEL
  200 LET R=INT(9*RND(X))+1
  205 IF R<=3 THEN 265
  210 IF R<=6 THEN 250
  215 IF R<=8 THEN 235
  220 LET W=4
  225 LET P=8
  230 RETURN
  235 LET W=3
  240 LET P=3.5
  245 RETURN
  250 LET W=2
  255 LET P=2
  260 RETURN
  265 LET W=1
  270 LET P=2
  275 RETURN
  999 END
```

We must now test the program. Ideally, you should test the program for both winning and losing on each of the four possible numbers, 1, 2, 3, and 4. In this case, however, I have only tested the program for winning and losing on 1. You should test the other six possible outcomes.

```
YOU HAVE 100 DOLLARS TO PLAY WITH.
PICK 1,2,3,4? 1
BET? 10
STOPPED AT 3
YOU LOSE $ 10 <-------------- LOSE ON NUMBER 1
YOU NOW HAVE 90 DOLLARS.
PICK 1,2,3,4? 1
BET? 10
STOPPED AT 3
YOU LOSE $ 10 <-------------- LOSE ON NUMBER 1
YOU NOW HAVE 80 DOLLARS.
PICK 1,2,3,4? 1
BET? 10
STOPPED AT 1
YOU WIN $ 20 <-------------- WIN ON NUMBER 1
YOU NOW HAVE 100 DOLLARS.
PICK 1,2,3,4?
↑C
```

EXERCISE SET 5.2

Off-line:

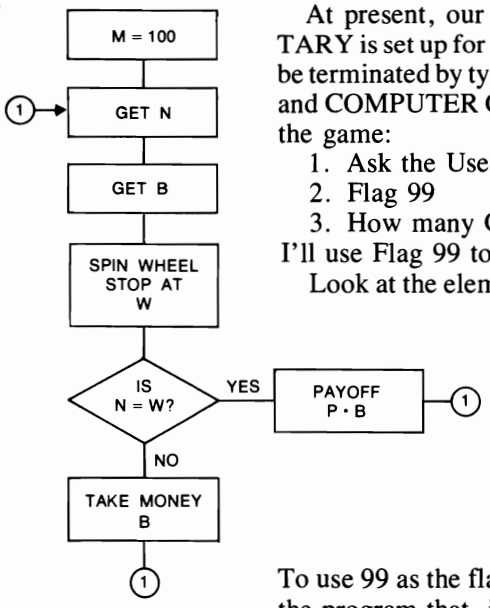
1. Design your own wheel with as many equal pie-shaped pieces as you wish. Number each section. Try not to have each different outcome occur with the same probability. That is, *do not* construct a four-section wheel and just number the sections 1, 2, 3, and 4 respectively.
2. Decide whether you want your game to be fair or biased and construct a payoff schedule for the different bets.
3. Draw (1) a flow chart for the main program of your Big Wheel game and (2) a flow chart for the SPIN WHEEL subroutine.
4. Refer to your flow charts and write a program for your big wheel. Imitate the line numbering scheme and overall program organization that I followed in writing the program BIG WHEEL/ELEMENTARY. *Do not* add instructions or any other personality at this time. These refinements will be added later. Keep your program simple.

On-line:

5. Load and run your Big Wheel program. Be sure to test it thoroughly. Save this program as YOUR BIG WHEEL for modification in subsequent sections.

5.3 "PLAY IT AGAIN, SAM?"

BIG WHEEL/ELEMENTARY FLOW CHART



At present, our program BIG WHEEL/ ELEMENTARY is set up for automatic replay. The game can only be terminated by typing CONTROL C. In YOU GUESS and COMPUTER GUESS, we had other ways to replay the game:

1. Ask the User
2. Flag 99
3. How many Games?

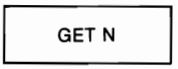
I'll use Flag 99 to replay BIG WHEEL.

Look at the elementary flow chart for BIG WHEEL.

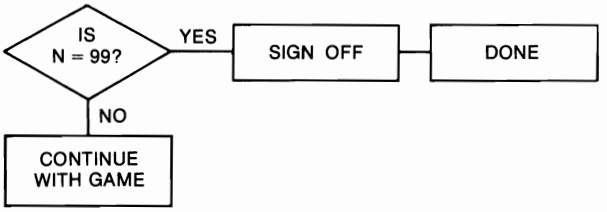
To use 99 as the flag, tell the player at the beginning of the program that, if he wants to quit, he must type 99 when the computer says "PICK 1,2,3,4." This is accomplished by inserting:

```
15 PRINT "IF YOU WISH TO QUIT, TYPE
          99 AFTER PICK 1,2,3,4."
```

Then test for the flag after the computer executes



which has been encoded as 50 INPUT N. To determine whether the player has typed in 99, we use this routine:



This routine can be encoded and inserted in the overall program three ways: (1) as an integral part of the main program, (2) as a standard subroutine employing GO SUB and RETURN, and (3) as a short routine which the computer reaches and returns from by following two GO TO instructions.

As part of the main program, it looks like this:

```
50 INPUT N
51 IF N = 99 THEN 53
52 GO TO 60
53 PRINT "HOPE YOU HAD FUN.
   COME BACK SOON."
54 STOP
60 PRINT "BET";
```

As a subroutine employing GO SUB and RETURN, it looks like this:

```
50 INPUT N
51 GO SUB 300
60 PRINT "BET";

300 IF N = 99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN.
    COME BACK SOON."
315 STOP
```

As a short routine outside the main program, which the computer reaches via 51 GO TO 300 and returns from via 305 GO TO 60, it looks like this:

```
50 INPUT N
51 GO TO 300
60 PRINT "BET";

300 IF N=99 THEN 310
305 GO TO 60
310 PRINT "HOPE YOU HAD FUN.
COME BACK SOON."
315 STOP
```

I prefer the second alternative, the standard subroutine. Here is FLAG 99 as a subroutine in BIG WHEEL/ ELEMENTARY.

BIG WHEEL/FLAG 99

PROGRAM

MAIN PROGRAM

```
10 RANDOM
15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
20 LET M=100
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
40 PRINT "PICK 1,2,3,4";
50 INPUT N
51 GO SUB 300 <-------------------- ACTIVATE FLAG 99
60 PRINT "BET";
70 INPUT B
80 GO SUB 200 <-------------------- ACTIVATE SPIN WHEEL
90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

SPIN WHEEL

```
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
999 END
```

OUTPUT

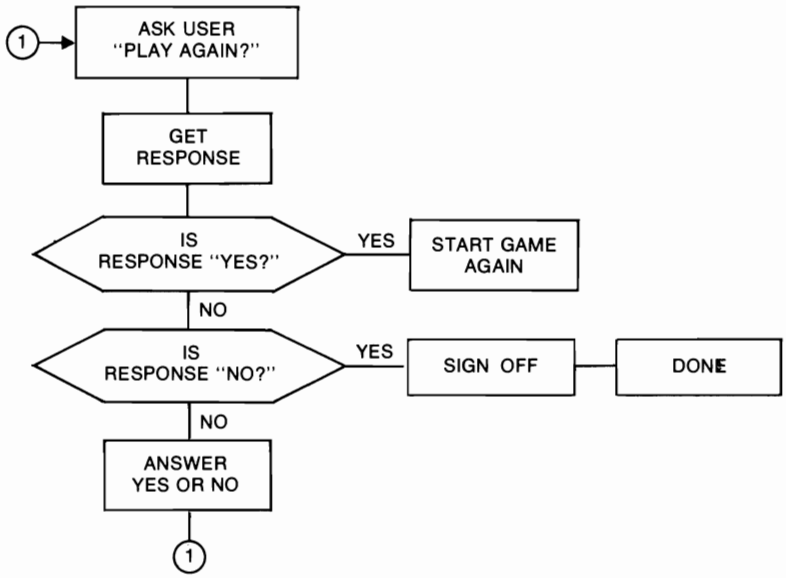
```
IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
YOU HAVE 100 DOLLARS TO PLAY WITH.
PICK 1,2,3,4? 2
BET? 10
STOPPED AT 3
YOU LOSE $ 10
YOU NOW HAVE 90 DOLLARS.
PICK 1,2,3,4? 99 <-------------- TEST FLAG 99
HOPE YOU HAD FUN. COME BACK SOON.
```

EXERCISE SET 5.3

Off-line:

1. The following flow chart outlines the logic of the replay option, Ask the User.

FLOW CHART



Based on this flow chart, do the following:

A) Code the flow chart as a subroutine which begins at line 300 in BIG WHEEL/ ELEMENTARY.

B) Insert in the main program the statement GO SUB 300 to activate this subroutine.

On-line:

2. Take YOUR BIG WHEEL program, which you saved in exercise set 5.2, and add the replay option of your choice. Save this program for further modification.

5.4 NEVER BET MORE THAN YOU HAVE

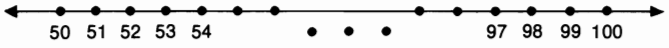
Not all games involve betting. In games of skill, such as football or tic-tac-toe, winning the game is generally satisfaction enough. Betting is often a characteristic of games of chance, however. To play a betting game, the user must start with a certain amount of money. You either let the user specify how much money he wants to draw from the "bank" (you may want to restrict the amount), or you tell him how much money he has. In the second case you can give the user a fixed amount, such as $100 or $1,000,000, or you can generate a number at random between two numbers to determine the amount of money he starts with. The game has more variety if you use the latter method, since the user starts with a different amount each time. In the present version of BIG WHEEL, the player starts with $100. Let's modify the program to start the player with a random amount of money between $50 and $100 inclusive.

Look at BIG WHEEL/FLAG 99. The program step to be changed is line 20, LET M = 100. The amount of money, 100, must be replaced with an expression which

uses RND(X) to generate a random amount of money between 50 and 100 dollars. RND(X), however, generates numbers between zero and one. To get RND(X) to work in the right interval, 50 to 100, we first determine the number of whole numbers between 50 and 100, inclusive. There are 100 − 50 + 1, or 51 whole numbers. RND(X) must therefore be multiplied by 51 to yield numbers from 0 to 51.

The expression 51*RND(X) yields decimal numbers, however. To transform them into whole numbers use INT. Thus INT(51*RND(X)) produces

Shift this set of numbers to the right, into the interval 50 to 100, by adding 50 to the expression. Thus, INT(51*RND(X)) + 50.



If you wish, you can now replace 20 LET M=100 with 20 LET M=INT(50*RND(X))+50 in BIG WHEEL/FLAG 99. Now let's examine the player's bet.

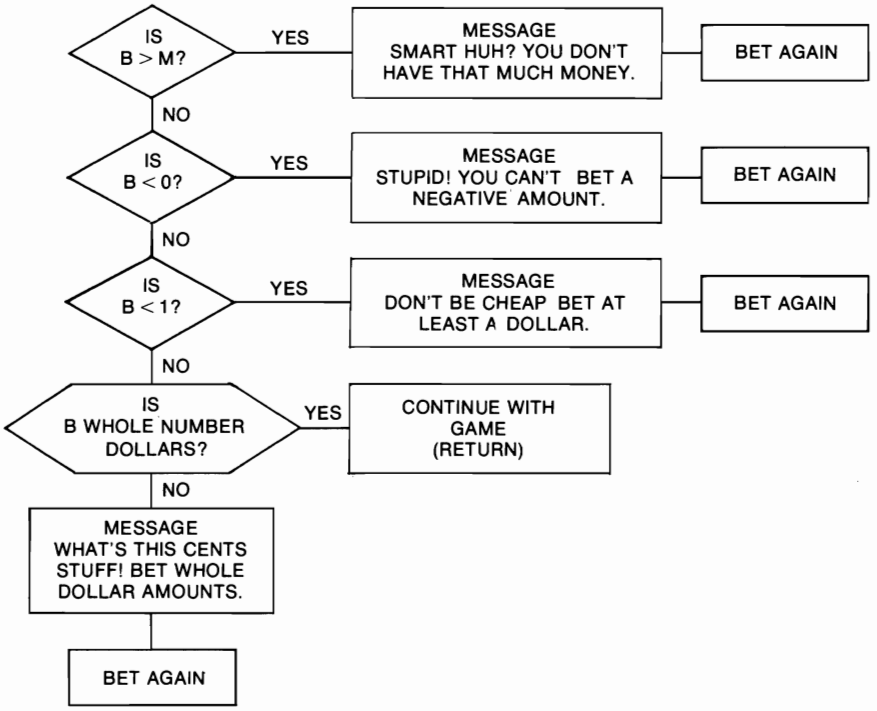
It is not wise to let the user bet more than he has, or to let him bet a negative amount of money. The first constraint is obvious, but the second isn't. You may already have discovered the following scheme for increasing the size of your bankroll when playing BIG WHEEL. Look again at the table which lists the ways you can win and lose on each number.

NUMBER BET ON	NUMBER WAYS TO WIN	NUMBER WAYS TO LOSE
1	3	6
2	3	6
3	2	7
4	1	8

On any given spin of the wheel, the most likely way to lose is to bet on the number 4. If I bet $1,000,000 on number 4, the chances are very good that I'll lose a million. On the other hand, if I bet -$1,000,000 on number 4, when I lose, I "win," because I gain $1,000,000 for my bankroll. The computer determines the size of my bankroll by executing LET M = M − B. When B = −1,000,000, the statement becomes LET M = M − (−1,000,000), which is, of course, the same as LET M = M + 1,000,000. I add a cool million to my wad. To prevent a player from using this scheme to increase his bankroll illegally, the computer must test each bet to make sure it is not a negative number.

To make the game realistic, you may require the player to bet a minimum of one dollar as well as to bet in whole dollar amounts. After all, did you ever hear of anyone betting 3.17625 dollars? Here is a flow chart for the subroutine that rejects any bet that is:

(1) more than his bankroll,
(2) negative,
(3) less than one dollar, or
(4) not a whole dollar amount.



BET OK FLOW CHART

Look at the questions in the diamonds. The first three questions can be inserted in the program without change between IF and THEN. The fourth question, IS B A WHOLE DOLLAR AMOUNT? is coded in BASIC, using the INT operation.

The question becomes, is INT(B) = B? It works this way: If B = 3.17625, the answer is "No," because INT(3.17625) equals 3, not 3.17625. On the other hand, if B = 3, then the answer is "Yes," because INT(3) equals 3. In short, if B is a whole number, the statement is true. When cast in BASIC it looks like this: IF INT(B) = B THEN _____.

The procedure for checking bets will be incorporated into BIG WHEEL as a subroutine. It is our third subroutine and will begin at line 400 There are five exit points from the subroutine. At the ends of four of the branches, a GO TO command is used to transfer control back to 60 PRINT "BET"; At the end of the remaining fifth branch, a RETURN statement is inserted. This subroutine is added to BIG WHEEL/FLAG 99. To

BIG WHEEL/BET OK

PROGRAM

activate it, GO SUB 400 must be inserted in the main program after 70 INPUT B.

```
MAIN PROGRAM
 10 RANDOM
 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
 20 LET M=100
 30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
 40 PRINT "PICK 1,2,3,4";
 50 INPUT N
 51 GO SUB 300 <-------------- ACTIVATE FLAG 99
 60 PRINT "BET";
 70 INPUT B
 71 GO SUB 400 <-------------- ACTIVATE BET OK
 80 GO SUB 200 <-------------- ACTIVATE SPIN WHEEL
 90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40

SPIN WHEEL
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN

FLAG 99
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP

BET OK
400 IF B>M THEN 455
405 IF B<0 THEN 445
410 IF B<1 THEN 435
415 IF INT(B)=B THEN 430
420 PRINT "WHAT'S THIS CENTS STUFF! BET WHOLE DOLLAR AMOUNTS."
425 GO TO 60
430 RETURN
435 PRINT "DON'T BE CHEAP. BET AT LEAST A DOLLAR."
440 GO TO 60
445 PRINT "STUPID! YOU CAN'T BET A NEGATIVE AMOUNT."
450 GO TO 60
455 PRINT "SMART HUH? YOU DON'T HAVE THAT MUCH MONEY."
460 GO TO 60
999 END
```

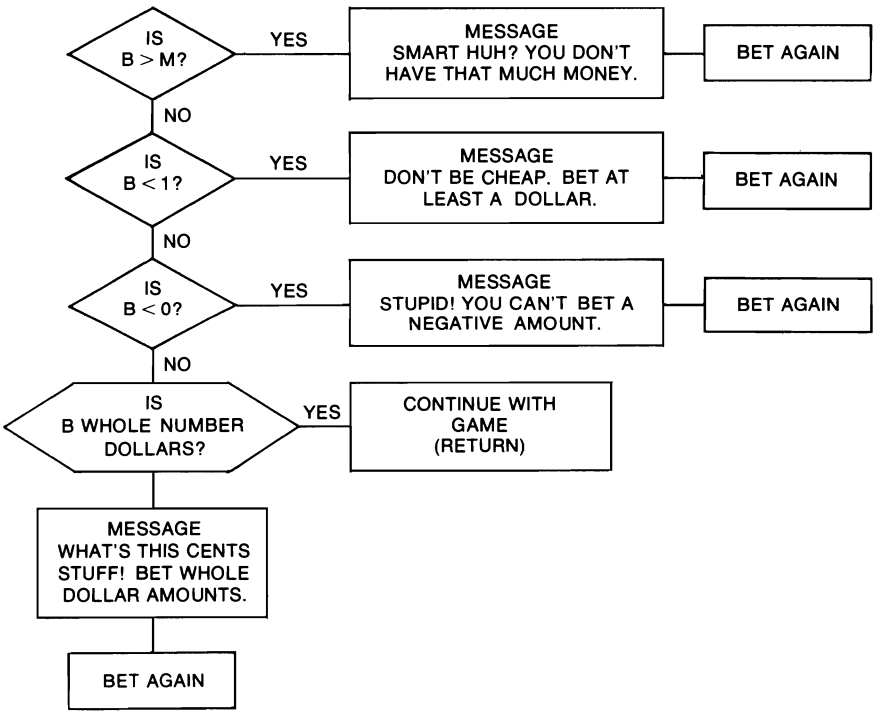
Below is a typical run of the program, one in which we test each of the four checks on the bet.

```
                IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
                YOU HAVE 100 DOLLARS TO PLAY WITH.
                PICK 1,2,3,4? 1
   B > M  ->    BET? 200
                SMART HUH? YOU DON'T HAVE THAT MUCH MONEY.
   B < 0  ->    BET? -50
                STUPID! YOU CAN'T BET A NEGATIVE AMOUNT.
   B < 1  ->    BET? 0
                DON'T BE CHEAP. BET AT LEAST A DOLLAR.
INT(B) =B ->    BET? 1.234
                WHAT'S THIS CENTS STUFF! BET WHOLE DOLLAR AMOUNTS.
                BET? 10
                STOPPED AT 2
                YOU LOSE $ 10
                YOU NOW HAVE 90 DOLLARS.
                PICK 1,2,3,4? 99
                HOPE YOU HAD FUN. COME BACK SOON.
```

EXERCISE SET 5.4

Off-line:

1. Does the order of the four questions used to check the bet make a difference? Play computer with the following flow chart and determine the output. If M = 100 and B = −1000, is the output correct?



2. Is there another sequence in which the questions can be arranged that will properly check the bet?

On-line:

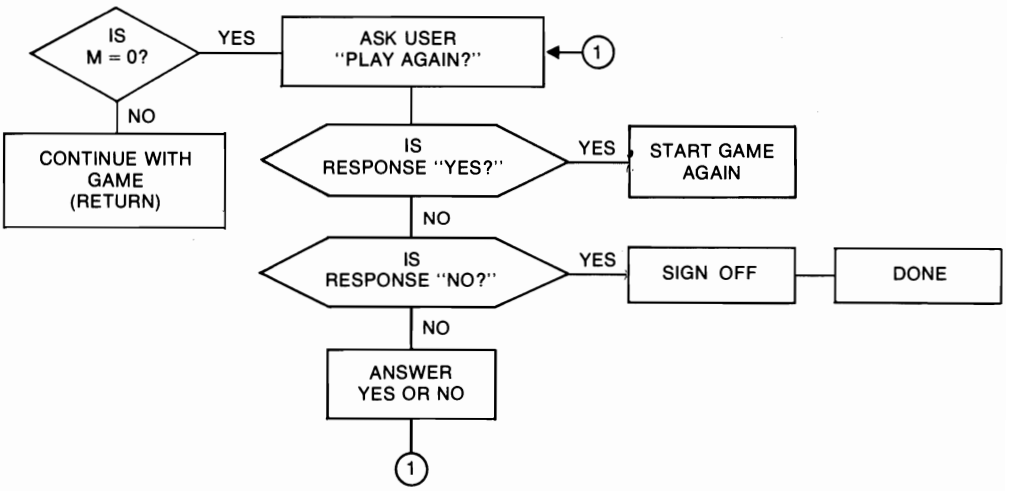
3. Take the program YOUR BIG WHEEL saved in exercise set 5.3 and add two routines. The first routine either (a) gives the player an opportunity to

draw an amount from the bank, (b) gives him $1000, or (c) gives him an amount randomly selected. The second routine checks the bet to see that the player does not bet (a) more than his bankroll or (b) a negative amount of money. You may also wish to create additional checks on the bet. Save this program for further modification.

5.5 YOU JUST BUSTED MY BANK

The BIG WHEEL game can end either of two ways: (1) the player loses all his money, or (2) he just decides to quit. He can quit by typing 99. If he loses all his money, we can either (1) terminate the game, or (2) give him a new bankroll and a chance to play again. In any case, whenever the player loses a bet, we should check the amount of money he has left. Of course, if he wins, there's no need to check the amount of money he has left; he must have some. Here is a flow chart of the routine to be followed each time the player loses. It gives the player an opportunity to start the game over again with a new bankroll if he has lost all his money, that is, if M = O.

BUSTED FLOW CHART



The new subroutine will begin at line 500. The string variable A$ is used to permit the player to answer YES or NO when asked "WOULD YOU LIKE TO PLAY AGAIN." When added to BIG WHEEL/FLAG 99, the subroutine is activated by GO SUB 500 on line 131.

BIG WHEEL/BUSTED

MAIN PROGRAM

```
10 RANDOM
15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
20 LET M=100
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
40 PRINT "PICK 1,2,3,4";
50 INPUT N
51 GO SUB 300 <-------------------- ACTIVATE FLAG 99
60 PRINT "BET";
70 INPUT B
80 GO SUB 200 <-------------------- ACTIVATE SPIN WHEEL
90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
131 GO SUB 500 <-------------------- ACTIVATE BUSTED
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

SPIN WHEEL

```
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
```

BUSTED

```
500 IF M=0 THEN 510
505 RETURN
510 PRINT "TOO BAD--YOU LOST ALL THE MONEY I GAVE YOU, BUT"
515 PRINT "I'LL BE GENEROUS AND GIVE YOU ANOTHER CHANCE."
520 PRINT "WOULD YOU LIKE TO PLAY AGAIN";
525 INPUT A$
530 IF A$="YES" THEN 20
535 IF A$="NO" THEN 550
540 PRINT "ANSWER YES OR NO."
545 GO TO 520
550 PRINT "THANKS FOR PLAYING. BETTER LUCK NEXT TIME."
555 STOP
999 END
```

Look at line 555. The STOP command ends the subroutine. Since the next line, 999, is an END statement, the STOP command may look unnecessary. You're right. But, in the next section, we're going to add another subroutine starting at line 600.

```
            IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
            YOU HAVE 100 DOLLARS TO PLAY WITH.
            PICK 1,2,3,4? 4
            BET? 100
            STOPPED AT 2
            YOU LOSE $ 100
  M = 0 --> YOU NOW HAVE 0 DOLLARS.
            TOO BAD--YOU LOST ALL THE MONEY I GAVE YOU, BUT
            I'LL BE GENEROUS AND GIVE YOU ANOTHER CHANCE.
YES OR NO --> WOULD YOU LIKE TO PLAY AGAIN? PERHAPS
            ANSWER YES OR NO.
      YES --> WOULD YOU LIKE TO PLAY AGAIN? YES
            YOU HAVE 100 DOLLARS TO PLAY WITH.
            PICK 1,2,3,4? 4
            BET? 100
            STOPPED AT 2
            YOU LOSE $ 100
  M = 0 --> YOU NOW HAVE 0 DOLLARS.
            TOO BAD--YOU LOST ALL THE MONEY I GAVE YOU, BUT
            I'LL BE GENEROUS AND GIVE YOU ANOTHER CHANCE.
       NO --> WOULD YOU LIKE TO PLAY AGAIN? NO
            THANKS FOR PLAYING. BETTER LUCK NEXT TIME.
```

EXERCISE SET 5.5

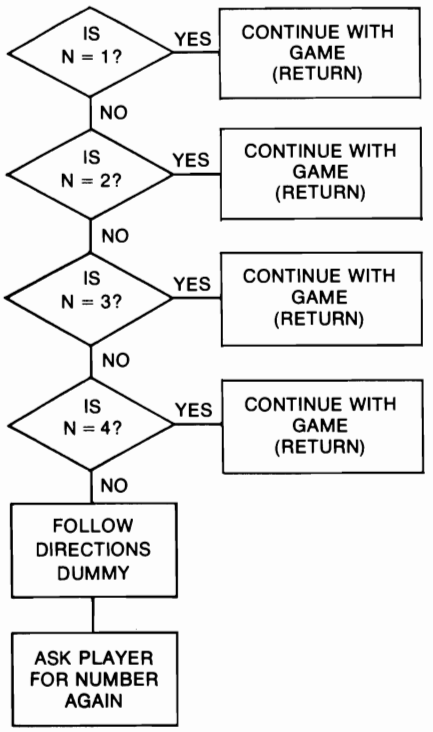
On-line:

1. Take the program YOUR BIG WHEEL saved in exercise set 5.4 and add a routine for determining if the player has lost all his money. Once again, save the new program.

5.6 VALIDATING INPUTS

The instructions for BIG WHEEL tell the player to pick either 1, 2, 3, or 4. But perhaps he wants to foil your program, or see if you've accounted for all contingencies. We need a routine to check the user's choice of number to determine if he has typed in a 1, 2, 3, or 4. If not, we ask the player to choose again. Here is the flow chart.

VALID 1
FLOW CHART



This flow chart is encoded as a subroutine beginning at line 600. To activate it, GO SUB 600 should be inserted right after 50 INPUT N. But the next line after 50 is 51 GO SUB 300 which checks for the flag. The check for the flag must come first. We'll call for the new subroutine in line 52.

PROGRAM **BIG WHEEL/VALID 1**

MAIN PROGRAM

```
 10 RANDOM
 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
 20 LET M=100
 30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
 40 PRINT "PICK 1,2,3,4";
 50 INPUT N
 51 GO SUB 300 <-------- ACTIVATE FLAG 99
 52 GO SUB 600 <-------- ACTIVATE VALID 1
 60 PRINT "BET";
 70 INPUT B
 80 GO SUB 200 <-------- ACTIVATE SPIN WHEEL
 90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
```

SPIN WHEEL

```
180 GO TO 40
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
```

VALID 1

```
600 IF N=1 THEN 630
605 IF N=2 THEN 630
610 IF N=3 THEN 630
615 IF N=4 THEN 630
620 PRINT "FOLLOW DIRECTIONS DUMMY."
625 GO TO 40
630 RETURN
999 END
```

```
OUTPUT IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
       YOU HAVE 100 DOLLARS TO PLAY WITH.
N = 7 ──► PICK 1,2,3,4? 7
       FOLLOW DIRECTIONS DUMMY.
N = -2.5 ──► PICK 1,2,3,4? -2.5
       FOLLOW DIRECTIONS DUMMY.
N = 0 ──► PICK 1,2,3,4? 0
       FOLLOW DIRECTIONS DUMMY.
N = 1 ──► PICK 1,2,3,4? 1
       BET? 10
       STOPPED AT 3
       YOU LOSE $ 10
       YOU NOW HAVE 90 DOLLARS.
N = 99 ──► PICK 1,2,3,4? 99
       HOPE YOU HAD FUN. COME BACK SOON.
```

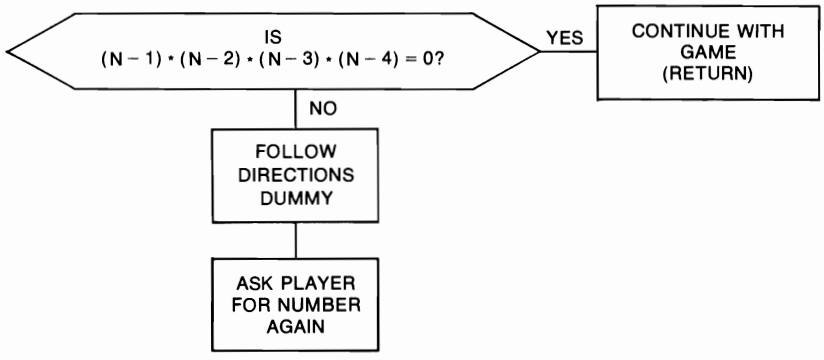
Now for a second method, which we'll refer to as VALID 2, for validating inputs. This new method employs the following IF-THEN statement:

IF (N − 1) * (N − 2) * (N − 3) * (N − 4) = 0 THEN_____

Suppose the player picks 1. If N = 1, the first factor N − 1 is zero, because 1 − 1 = 0. Regardless of the values of the other three factors the product of these four factors, 0 * (−1) * (−2) * (−3), must be zero. Likewise, if N is 2, 3, or 4, one of the other factors must be zero and the product is zero. Thus, the statement (N − 1) * (N − 2) * (N − 3) * (N − 4) = 0 is always true if N is 1, 2, 3, or 4. Any other value of N will make the statement false. For example, if N = 8 then N − 1 equals 7; N − 2 equals 6; N − 3 equals 5; and N − 4 equals 4. The product is 7*6*5*4, which is not zero.

To sum up, if the statement is true, we know he has picked a valid number. On the other hand if the statement is not true, we know he has not picked a valid number and we'll ask him to pick again. In a flow chart, the routine looks like this:

VALID 2 FLOW CHART



When encoded as a subroutine, it would look like this:

VALID 2 *SUBROUTINE*

```
600 IF (N-1)*(N-2)*(N-3)*(N-4)=0 THEN 615
605 PRINT "FOLLOW DIRECTIONS DUMMY."
610 GO TO 40
615 RETURN
```

The foregoing method, VALID 2, is satisfactory when the player has only four numbers to choose from. If the BIG WHEEL had 20 numbers to choose from, however, the required statement would look like this:

```
IF (N - 1)*(N - 2)*(N - 3)*(N - 4)*(N - 5)*(N - 6)*(N - 7)*(N - 8)* . . .
*(N - 20) = 0 THEN _____
```

That's much too cumbersome. There must be a shorter way. There is! To illustrate the third alternative, which we'll call VALID 3, I'll use the original BIG WHEEL with four valid numbers. VALID 3 requires two IF-THEN statements. Here is the first:

```
IF(N - 1)*(N - 4) < = 0 THEN _____
```

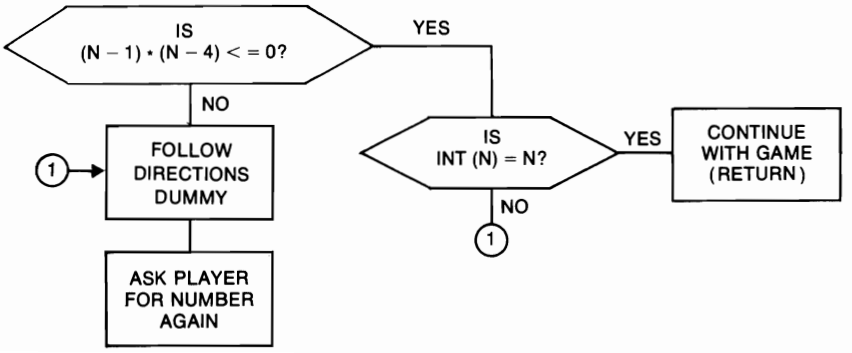
Suppose the player picks the number 1. If N = 1, then N − 1 is 0 and N − 4 is −3. The resultant product, 0*(−3), equals zero and the statement (N − 1)*(N − 4) < = 0 is true. If the player picks the number 4, the product will also be zero. If N is 2 or 3, the product will be a negative number, and again the statement is true.

So much for the valid numbers, 1, 2, 3, and 4. If the player picks 0, N − 1 will be negative and N − 4 will be negative. The resultant product is positive, and the statement (N − 1)*(N − 4) < = 0 is false. If the player picks a number greater than four, N − 1 and N − 4 will both be positive. Again the resultant product is positive, and the statement (N − 1)*(N − 4) < = 0 is false. In short, if N is a number between 1 and 4 inclusive, (N − 1)*(N − 4) < = 0 is true.

Unfortunately, 1, 2, 3, or 4 are not the only numbers between 1 and 4. Suppose the player types in 2.5. This number also makes the statement true. We need another test to determine if N is a whole number. This can be done with the statement.

```
IF INT(N) = N THEN _____
```

If N is a whole number, INT(N) = N is true. If N is not a whole number, then the statement is false. Thus the third method consists of two tests on N. The first determines if N is between 1 and 4 inclusive and the second determines if N is a whole number. If N passes both these tests, then it must be 1, 2, 3, or 4. It can't be anything else. The flow chart for VALID 3, the third method of validating the player's number, is:



VALID 3
FLOW CHART

Here is the routine encoded as a subroutine beginning at line 600.

VALID 3
SUBROUTINE

```
600 IF (N-1)*(N-4)<=0 THEN 615
605 PRINT "FOLLOW DIRECTIONS DUMMY."
610 GO TO 40
615 IF INT(N)=N THEN 625
620 GO TO 605
625 RETURN
```

EXERCISE SET 5.6

On-line:

1. Take the program YOUR BIG WHEEL saved in exercise set 5.5 and add the routine of your choice to validate the user's number. Save this program.

5.7 COMPUTER PERSONALITY

Now that YOUR BIG WHEEL program is essentially complete, let's have some fun. You can make your program much more entertaining for the player by adding some personality to it. Here are several things you can do.

1. Tell the player the name of the game. Insert:
   12 PRINT "PLAYING BIG WHEEL"
2. Give the player the feeling that a large wheel is being spun. Before you ask him to pick his number, insert:
   35 PRINT "WHEEL IS SPINNING"
3. Replace 90 PRINT "STOPPED AT" W with
   90 PRINT "WHEEL STOPPED AT" W
4. Ask the player for his name at the beginning of the program.
   16 PRINT "PLEASE TELL ME YOUR NAME";
   17 INPUT N$
   The string variable N$ holds a place for the player's name. On my computer, the string cannot contain more than six characters.
5. When the player types 99, use his name in the sign-off message. Replace
   310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
   with
   310 PRINT "HOPE YOU HAD FUN "N$
   " COME BACK SOON."
6. Cheer the winner with a peal of bells. After 150 PRINT "YOU WIN $" P*B, insert line
   151 GO SUB 350.
   At 350 (the FLAG 99 subroutine only occupies lines 300-315), insert a subroutine which, when activated, rings the bell five times. On my computer I can ring the bell once by writing PRINT CHR$(135). To ring the bell five times I use a loop,
   350 FOR C = 1 TO 5
   355 PRINT CHR$(135);
   360 NEXT C
   365 RETURN

The semicolon in line 355 prevents the output page from advancing each time the bell rings.

These are just some of the things you can do to give your program personality. Undoubtedly you can think of others. Here is BIG WHEEL/FLAG 99 with these features.

PROGRAM **BIG WHEEL/PERSONALITY**

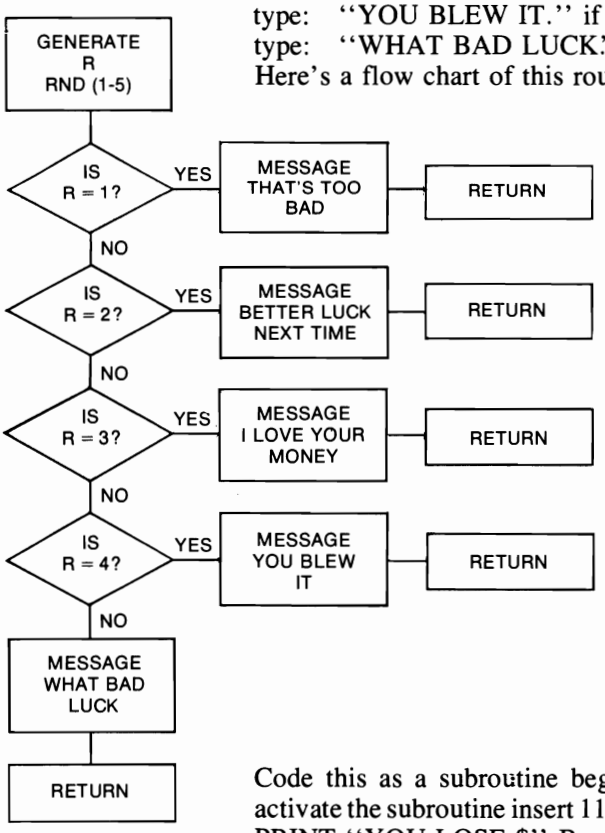
```
MAIN PROGRAM
 10 RANDOM
 12 PRINT "PLAYING BIG WHEEL"                          <- TITLE
 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
 16 PRINT "PLEASE TELL ME YOUR NAME"
 17 INPUT N$                                           <- PLAYER'S NAME
 20 LET M=100
 30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
 35 PRINT "WHEEL IS SPINNING"                          <- DESCRIPTION
 40 PRINT "PICK 1,2,3,4";
 50 INPUT N
 51 GO SUB 300                                         <- ACTIVATE FLAG 99
 60 PRINT "BET";
 70 INPUT B
 80 GO SUB 200                                         <- ACTIVATE SPIN WHEEL
 90 PRINT "WHEEL STOPPED AT" W                         <- DESCRIPTION
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
151 GO SUB 350                                         <- ACTIVATE RING BELL
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40

SPIN WHEEL
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN

FLAG 99
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN " N$ ". COME BACK SOON."   <- SIGN OFF
315 STOP

RING BELL
350 FOR C=1 TO 5
355 PRINT CHR$(135);
360 NEXT C
365 RETURN
999 END
```

```
OUTPUT PLAYING BIG WHEEL
       IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
       PLEASE TELL ME YOUR NAME? BOBBY
       YOU HAVE 100 DOLLARS TO PLAY WITH.
       WHEEL IS SPINNING
       PICK 1,2,3,4? 1
       BET? 5
       WHEEL STOPPED AT 1
       YOU WIN $ 10
        YOU NOW HAVE 110 DOLLARS.
       PICK 1,2,3,4? 99
       HOPE YOU HAD FUN BOBBY. COME BACK SOON.
```

If you think you're a wit, you can have the computer make a funny remark whenever the player loses. For the sake of variety, you can make up five different remarks, then, using the random number generator, instruct the computer to pick one of the remarks at random each time the player loses. This new remark will occur in the output after the statement YOU LOSE $____, in which the blank represents the wager. Here's the scheme, using a set of remarks I thought up.

Type: "THAT'S TOO BAD." if 1 is generated;
type: "BETTER LUCK NEXT TIME." if 2 is generated;
type: "I LOVE YOUR MONEY." if 3 is generated;
type: "YOU BLEW IT." if 4 is generated;
type: "WHAT BAD LUCK." if 5 is generated.
Here's a flow chart of this routine:

LOSER FLOW CHART



Code this as a subroutine beginning at line 700. To activate the subroutine insert 111 GO SUB 700 after 110 PRINT "YOU LOSE $" B.

PROGRAM **BIG WHEEL/LOSER**

MAIN PROGRAM

```
10 RANDOM
15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
20 LET M=100
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
40 PRINT "PICK 1,2,3,4";
50 INPUT N
51 GO SUB 300 <-------------- ACTIVATE FLAG 99
60 PRINT "BET";
70 INPUT B
80 GO SUB 200 <-------------- ACTIVATE SPIN WHEEL
90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
111 GO SUB 700 <-------------- ACTIVATE LOSER
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

SPIN WHEEL

```
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
```

LOSER

```
700 LET R=INT(5*RND(X))+1
705 IF R=1 THEN 765
710 IF R=2 THEN 755
715 IF R=3 THEN 745
720 IF R=4 THEN 735
725 PRINT "WHAT BAD LUCK."
730 RETURN
735 PRINT "YOU BLEW IT."
740 RETURN
745 PRINT "I LOVE YOUR MONEY."
750 RETURN
755 PRINT "BETTER LUCK NEXT TIME."
760 RETURN
765 PRINT "THAT'S TOO BAD."
770 RETURN
999 END
```

Here is a typical run:

```
OUTPUT IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
       YOU HAVE 100 DOLLARS TO PLAY WITH.
       PICK 1,2,3,4? 2
       BET? 10
       STOPPED AT 4
       YOU LOSE $ 10
MESSAGE 2 → BETTER LUCK NEXT TIME.
       YOU NOW HAVE 90 DOLLARS.
       PICK 1,2,3,4? 4
       BET? 10
       STOPPED AT 1
       YOU LOSE $ 10
MESSAGE 1 → THAT'S TOO BAD.
       YOU NOW HAVE 80 DOLLARS.
       PICK 1,2,3,4? 4
       BET? 10
       STOPPED AT 2
       YOU LOSE $ 10
MESSAGE 1 → THAT'S TOO BAD.
       YOU NOW HAVE 70 DOLLARS.
       PICK 1,2,3,4? 99
       HOPE YOU HAD FUN. COME BACK SOON.
```

Look at the loser's subroutine in BIG WHEEL/LOSER.

LOSER
SUBROUTINE

```
700 LET R=INT(5*RND(X))+1
705 IF R=1 THEN 765
710 IF R=2 THEN 755
715 IF R=3 THEN 745
720 IF R=4 THEN 735
725 PRINT "WHAT BAD LUCK."
730 RETURN
735 PRINT "YOU BLEW IT."
740 RETURN
745 PRINT "I LOVE YOUR MONEY."
750 RETURN
755 PRINT "BETTER LUCK NEXT TIME."
760 RETURN
765 PRINT "THAT'S TOO BAD."
770 RETURN
```

In line 700, R is assigned one of the whole numbers 1, 2, 3, 4, or 5. Four IF-THEN statements are required to determine the value of R. If R is 5, the computer falls through the four IF-THEN statements to line 725. You can save several steps in writing the loser's subroutine if your version of BASIC has the multiple branch statement ON _ GO TO. Then the four If-THEN statements can be replaced by the one statement:

ON R GO TO 765, 755, 745, 735, 725.

The new statement works this way. First of all, R must be a whole number. Then, if R is the number 1, the computer looks at the first line number after the GO TO. In this case the number is 765. The computer branches to line 765 and executes the instructions beginning at that line. If R is the number 2, the computer looks at the second line number after the GO TO. In this case the

number is 755. The computer branches to line 755. As long as R is one of the whole numbers 1, 2, 3, 4, or 5, the computer knows where to go. Here is the loser's subroutine using the multiple branch statement.

LOSER/ON–GO TO *SUBROUTINE*

```
700 LET R=INT(5*RND(X))+1
705 ON R GO TO 765,755,745,735,725
725 PRINT "WHAT BAD LUCK."
730 RETURN
735 PRINT "YOU BLEW IT."
740 RETURN
745 PRINT "I LOVE YOUR MONEY."
750 RETURN
755 PRINT "BETTER LUCK NEXT TIME."
760 RETURN
765 PRINT "THAT'S TOO BAD."
770 RETURN
```

EXERCISE SET 5.7

On-line:

1. Take the program YOUR BIG WHEEL saved in exercise set 5.6 and add two subroutines, one which makes a variety of remarks chiding a loser, and the other a variety of remarks complimenting a winner. Start the winner's subroutine at line 800. Add any other personality you wish and save this new program.

5.8 LAYOUT

Your games will be more fun if the output is well organized and easy to read. Setting off and indenting portions of the output will make your program look much more polished. Here are some things you can do to improve layout.

1. At present the title PLAYING BIG WHEEL appears at the left margin of the output. It would look more like a title if it appeared in the middle of the page. The title consists of 17 characters, including blanks. Since there are 72 printing spaces across the page, the title can be centered by leaving 27 or 28 blanks on either side. Let's count off 27 blanks from the left margin, using the TAB command. Replace line 12 with

   12 PRINT TAB(27); "PLAYING BIG WHEEL"

2. After the title is printed, skip several lines to set the title off from the rest of the output. To do this, add blank PRINT statements.

   13 PRINT
   14 PRINT

   In fact, you can use blank PRINT statements to set off *any* lines you want to have stand out from the rest of the text.

3. Two high points of the game occur when the wheel is spun and then when it stops on a number. To dramatize these events replace 35 PRINT "WHEEL IS SPINNING" with

   35 PRINT "———WHEEL IS SPINNING"

and replace 80 PRINT "WHEEL STOPPED AT" W with
90 PRINT "———WHEEL STOPPED AT" W.

Don't get carried away, however. If you print out on a Teletype, keep in mind that it is a very slow and noisy printer. Don't ask it to type out any more characters or blanks than necessary, or the player will soon lose interest in your game.

Here's my BIG WHEEL in which I've made the above changes to improve format. I've also inserted blank PRINT statements to separate successive games.

BIG WHEEL/LAYOUT

```
MAIN PROGRAM
   10 RANDOM
   12 PRINT TAB(27);"PLAYING BIG WHEEL"          <- TITLE
   13 PRINT                                      <- SKIP LINES
   14 PRINT                                      <- SKIP LINES
   15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
   16 PRINT                                      <- SKIP LINES
   17 PRINT                                      <- SKIP LINES
   20 LET M=100
   30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
   31 PRINT                                      <- SKIP LINES
   32 PRINT                                      <- SKIP LINES
   35 PRINT "---WHEEL IS SPINNING"               <- DRAMATIZE
   40 PRINT "PICK 1,2,3,4";
   50 INPUT N
   51 GO SUB 300
   60 PRINT "BET";
   70 INPUT B
   80 GO SUB 200
   90 PRINT "---WHEEL STOPPED AT" W              <- DRAMATIZE
  100 IF N=W THEN 150
  110 PRINT "YOU LOSE $" B
  120 LET M=M-B
  130 PRINT "YOU NOW HAVE" M "DOLLARS."
  140 GO TO 31                                   <- SPACING BETWEEN GAMES
  150 PRINT "YOU WIN $" P*B
  160 LET M=M+P*B
  170 PRINT "YOU NOW HAVE" M "DOLLARS."
  180 GO TO 31                                   <- SPACING BETWEEN GAMES
SPIN WHEEL
  200 LET R=INT(9*RND(X))+1
  205 IF R<=3 THEN 265
  210 IF R<=6 THEN 250
  215 IF R<=8 THEN 235
  220 LET W=4
  225 LET P=8
  230 RETURN
  235 LET W=3
  240 LET P=3.5
  245 RETURN
  250 LET W=2
  255 LET P=2
  260 RETURN
  265 LET W=1
  270 LET P=2
  275 RETURN
FLAG 99
  300 IF N=99 THEN 310
  305 RETURN
  310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
  315 STOP
  999 END
```

Here is a typical run.

```
                    PLAYING BIG WHEEL


IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.


YOU HAVE 100 DOLLARS TO PLAY WITH.


---WHEEL IS SPINNING
PICK 1,2,3,4? 3
BET? 10
---WHEEL STOPPED AT 3
YOU WIN $ 35
YOU NOW HAVE 135 DOLLARS.


---WHEEL IS SPINNING
PICK 1,2,3,4? 2
BET? 20
---WHEEL STOPPED AT 1
YOU LOSE $ 20
YOU NOW HAVE 115 DOLLARS.


---WHEEL IS SPINNING
PICK 1,2,3,4? 99
HOPE YOU HAD FUN. COME BACK SOON.
```

Look at the output. Which lines in the program BIG WHEEL/LAYOUT generated the blank lines between the first and second game?

EXERCISE SET 5.8

On-line:

1. Take the program YOUR BIG WHEEL saved in exercise set 5.7 and add program steps to improve the layout. Save this program.

5.9 INSTRUCTIONS

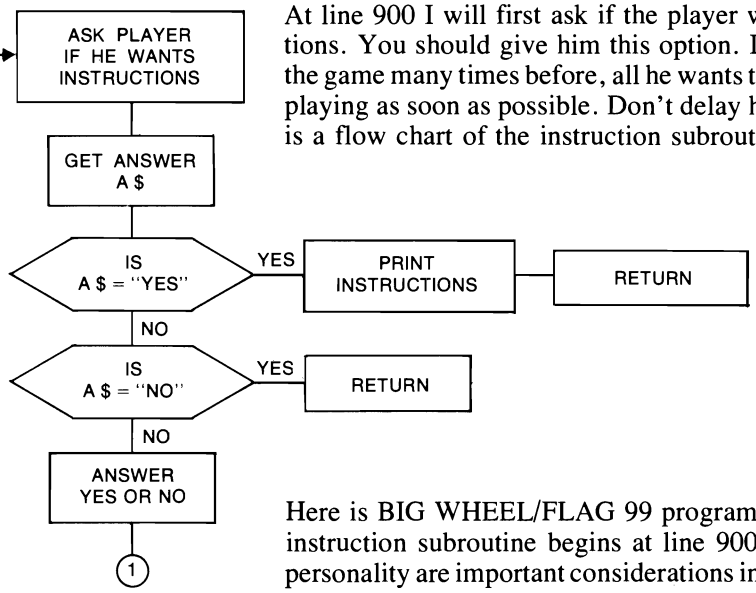
Now to add a set of instructions to the player. That's about the last step in writing a program. If you insert the instructions while you are still developing the program you yourself will have to cope with them every time you run it.

When writing instructions, try to be as thorough and as concise as possible. First, decide how much the player probably knows about the game. For instance, if you're playing a common game like tic-tac-toe, you might well assume that the player knows all the rules; you only need to tell him how your "board" is numbered.

I will place my instructions for BIG WHEEL in a subroutine beginning at line 900. To activate this subroutine I will replace 15 PRINT "WHENEVER YOU WANT TO QUIT TYPE 99" with

```
15 GO SUB 900
```

INSTRUCTION FLOW CHART



At line 900 I will first ask if the player wants instructions. You should give him this option. If he's played the game many times before, all he wants to do is to start playing as soon as possible. Don't delay his start. Here is a flow chart of the instruction subroutine.

Here is BIG WHEEL/FLAG 99 program in which the instruction subroutine begins at line 900. Layout and personality are important considerations in this routine.

BIG WHEEL/INSTRUCTION

MAIN PROGRAM

```
10 RANDOM
15 GO SUB 900 <-------- ACTIVATE INSTRUCTIONS
20 LET M=100
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
40 PRINT "PICK 1,2,3,4";
50 INPUT N
51 GO SUB 300 <-------- ACTIVATE FLAG 99
60 PRINT "BET";
70 INPUT B
80 GO SUB 200 <-------- ACTIVATE SPIN WHEEL
90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

SPIN WHEEL

```
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
```

INSTRUCTIONS

```
900 PRINT "DO YOU WANT INSTRUCTIONS";
905 INPUT A$
906 PRINT
907 PRINT
910 IF A$="NO" THEN 930
915 IF A$="YES" THEN 935
920 PRINT "ANSWER YES OR NO."
925 GO TO 900
930 RETURN
935 PRINT "IMAGINE A LARGE CARNIVAL WHEEL, 6 FEET IN DIAMETER,"
936 PRINT "DIVIDED INTO 9 EQUAL PIE-SHAPED PIECES. THREE SECTIONS"
937 PRINT "ARE NUMBERED 1, THREE SECTIONS ARE NUMBERED 2, TWO"
938 PRINT "SECTIONS ARE NUMBERED 3, AND ONE SECTION IS NUMBERED 4."
939 PRINT
940 PRINT "THE WHEEL WILL BE SPUN AND THEN YOU WILL HAVE AN"
941 PRINT "OPPORTUNITY TO BET ON ONE OF THE FOUR NUMBERS"
942 PRINT "BEFORE THE WHEEL COMES TO REST."
943 PRINT
944 PRINT "AT THE START OF EACH GAME, YOU WILL RECEIVE"
945 PRINT "A CERTAIN AMOUNT OF MONEY TO BET IN WHOLE"
946 PRINT "DOLLAR AMOUNTS."
947 PRINT
948 PRINT "SECTION NUMBERS PAYOFF AS FOLLOWS:"
949 PRINT "1   PAYS  2  TIMES YOUR BET."
950 PRINT "2   PAYS  2  TIMES YOUR BET."
951 PRINT "3   PAYS  3.5 TIMES YOUR BET."
952 PRINT "4   PAYS  8  TIMES YOUR BET."
953 PRINT
954 PRINT "WHEN YOU WANT TO QUIT(IF YOU DON'T GO BROKE FIRST)"
955 PRINT "TYPE 99 WHEN ASKED TO PICK 1,2,3,4."
956 PRINT
957 PRINT
958 PRINT "LET'S GO."
959 PRINT
960 RETURN
999 END
```

DESCRIPTION OF WHEEL

PLAY OF GAME

BANKROLL

PAYOFF TABLE

QUIT

Here is a typical run.

```
DO YOU WANT INSTRUCTIONS? YES


IMAGINE A LARGE CARNIVAL WHEEL, 6 FEET IN DIAMETER,
DIVIDED INTO 9 EQUAL PIE-SHAPED PIECES. THREE SECTIONS
ARE NUMBERED 1, THREE SECTIONS ARE NUMBERED 2, TWO
SECTIONS ARE NUMBERED 3, AND ONE SECTION IS NUMBERED 4.

THE WHEEL WILL BE SPUN AND THEN YOU WILL HAVE AN
OPPORTUNITY TO BET ON ONE OF THE FOUR NUMBERS
BEFORE THE WHEEL COMES TO REST.

AT THE START OF EACH GAME, YOU WILL RECEIVE
A CERTAIN AMOUNT OF MONEY TO BET IN WHOLE
DOLLAR AMOUNTS.

SECTION NUMBERS PAYOFF AS FOLLOWS:
1  PAYS  2  TIMES YOUR BET.
2  PAYS  2  TIMES YOUR BET.
3  PAYS  3.5 TIMES YOUR BET.
4  PAYS  8  TIMES YOUR BET.

WHEN YOU WANT TO QUIT(IF YOU DON'T GO BROKE FIRST)
TYPE 99 WHEN ASKED TO PICK 1,2,3,4.


LET'S GO.

YOU HAVE 100 DOLLARS TO PLAY WITH.
PICK 1,2,3,4? 1
BET? 10
STOPPED AT 2
YOU LOSE $ 10
YOU NOW HAVE 90 DOLLARS.
PICK 1,2,3,4? 99
HOPE YOU HAD FUN. COME BACK SOON.
```

EXERCISE SET 5.9

On-line:

1. Take the program YOUR BIG WHEEL saved in exercise set 5.8 and add a routine giving instructions to the user. Save this program for further modification.

5.10 IS BIG WHEEL BUGLESS?

Are there any bugs in YOUR BIG WHEEL program? By now you know that it is possible to write a program which you think is flawless, but which in fact is not. Sometimes you discover flaws on the first test run. Other times flaws are not discovered until much later. In fact you can never be sure that a program is flawless! You only become more confident that your program is bugless when it operates correctly many times under a variety of conditions. Thus, to insure that your program is near perfect, have a number of people play your game and ask them to let you know if they encounter any

problems. That's how I discovered a flaw in my BIG WHEEL program. A student playing my game was confronted with the following situation.

The student bet five dollars on the number 3 and won. The payoff factor for number 3 is 3.5. He won 3.5 times 5, or $17.50. Adding $17.50 to his bankroll of $50 gave him $67.50. He then bet $67 on number 2 and lost, cutting his bankroll to a mere 50 cents. He bet his 50 cents on number 1. The computer replied DON'T BE CHEAP. BET AT LEAST A DOLLAR. He tried to bet a dollar and the computer replied SMART HUH? YOU DON'T HAVE THAT MUCH MONEY. No matter what he tried to bet, he was rebuffed. He couldn't even type 99 to quit! The only way out was CONTROL C. (Gloom) His 50c bet would not pass any of the checks I built into the program. What can I do to stamp out this bug?

The problem lies with the payoff factor of 3.5. If the player bets an odd number of dollars on number 3 and wins, his bankroll will always turn out to be a whole number of dollars, plus 50 cents. If he ever loses all his whole dollars, he's stuck with 50 cents, but he can't bet less than a dollar.

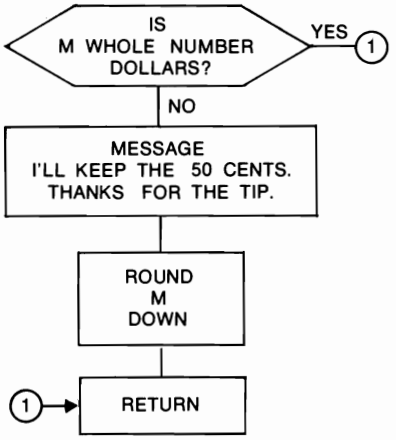
One solution is to redesign the big wheel, numbering the sections so that all payoff factors are whole numbers. Another solution is to remove the constraint that the player must bet at least a dollar. Doing this however, would encourage strange bets such as .0000163 dollars. A third alternative is to make sure the player's bankroll is always a whole number of dollars after he wins. If for example his bankroll is at some point $67.50, the computer will round off the amount to $67 by taking the 50c as a "tip."

To do this, we'll add a short subroutine to the program. The proper place to insert this "patch" is at the point where the player wins, and the computer calculates his new bankroll. This occurs between lines 150 and 180, shown here:

```
150 PRINT "YOU WIN$" P*B
160 LET M = M + P*B
170 PRINT "YOU NOW HAVE" M
    "DOLLARS."
180 GO TO 40
```

In line 150 the player is told that he won and the amount of his winnings. In line 160 the player's winnings are added to his bankroll. At this point we want the computer to determine if the player's bankroll is a whole number of dollars. So we insert at line 161 a GO SUB statement that directs the computer to the "patch." But where can I fit the "patch?" I've used up all the numbers I usually use to start subroutines, that is 200, 300, . . . , 900. I don't want to go to 1000, and there is a nice hole starting at 650. So, starting at line 650, I'll instruct the computer to check the new bankroll, M. If the bankroll is a whole number of dollars, the computer can

50 CENT TIP FLOW CHART



return to the main program. If the bankroll is not a whole number of dollars, I instruct it to type out the message: I'LL KEEP THE 50 CENTS. THANKS FOR THE TIP, and then round his bankroll down to the nearest whole dollar. Here is a flow chart of the subroutine at left.

To encode this subroutine I use the INT command to test for a whole number of dollars, as well as to round M down. Here is BIG WHEEL/FLAG 99 with the new subroutine.

BIG WHEEL/50 CENT TIP

```
MAIN PROGRAM
  10 RANDOM
  15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
  20 LET M=100
  30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
  40 PRINT "PICK 1,2,3,4";
  50 INPUT N
  51 GO SUB 300 <-------- ACTIVATE
  60 PRINT "BET";
  70 INPUT B
  80 GO SUB 200 <-------- ACTIVATE SPIN WHEEL
  90 PRINT "STOPPED AT" W
 100 IF N=W THEN 150
 110 PRINT "YOU LOSE $" B
 120 LET M=M-B
 130 PRINT "YOU NOW HAVE" M "DOLLARS."
 140 GO TO 40
 150 PRINT "YOU WIN $" P*B
 160 LET M=M+P*B
 161 GO SUB 650 <-------- ACTIVATE 50 CENT TIP
 170 PRINT "YOU NOW HAVE" M "DOLLARS."
 180 GO TO 40
SPIN WHEEL
 200 LET R=INT(9*RND(X))+1
 205 IF R<=3 THEN 265
 210 IF R<=6 THEN 250
 215 IF R<=8 THEN 235
 220 LET W=4
 225 LET P=8
 230 RETURN
 235 LET W=3
 240 LET P=3.5
 245 RETURN
 250 LET W=2
 255 LET P=2
 260 RETURN
 265 LET W=1
 270 LET P=2
 275 RETURN
FLAG 99
 300 IF N=99 THEN 310
 305 RETURN
 310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
 315 STOP
50 CENT TIP
 650 IF INT(M)=M THEN 665
 655 PRINT "I'LL KEEP THE 50 CENTS. THANKS FOR THE TIP."
 660 LET M=INT(M)
 665 RETURN
 999 END
```

To test this subroutine, I want to bet an odd number of dollars on number 3 and then have the wheel stop on that number. The wheel will stop on 3 if, in the spin-the-wheel subroutine that begins at line 200, the value of R is 8 or 9. Rather than wait for the random number generator, acting through INT(9*RND(X)) + 1, to produce 8 or 9, I'll generate a suitable value of R myself by replacing 200 LET R = INT(9*RND(X)) + 1 with 200 PRINT "SPECIFY R"; and adding 201 INPUT R. Now, when the program is running, the computer will stop at line 201 and let me specify a value for R. I will specify 8 or 9. Here is BIG WHEEL/FLAG 99 with these two changes in lines 200 and 201.

BIG WHEEL/50 CENT TIP/TEST

MAIN PROGRAM

```
 10 RANDOM
 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
 20 LET M=100
 30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
 40 PRINT "PICK 1,2,3,4";
 50 INPUT N
 51 GO SUB 300
 60 PRINT "BET";
 70 INPUT B
 80 GO SUB 200
 90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
161 GO SUB 650          <-- ACTIVATE 50 CENT TIP
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

SPIN WHEEL

```
200 PRINT "SPECIFY R";  <-- SPECIFY R
201 INPUT R
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
```

50 CENT TIP

```
650 IF INT(M)=M THEN 665
655 PRINT "I'LL KEEP THE 50 CENTS. THANKS FOR THE TIP."
660 LET M=INT(M)
665 RETURN
999 END
```

Here is a test run.

```
IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4.
YOU HAVE 100 DOLLARS TO PLAY WITH.
PICK 1,2,3,4? 3
BET? 15
SPECIFY R? 8 <-------------------- I WANT WHEEL TO STOP AT 3
STOPPED AT 3
YOU WIN $ 52.5 <-------------------- THIS IS 3.5 * 15
I'LL KEEP THE 50 CENTS. THANKS FOR THE TIP. <- 50 CENT TIP SUBROUTINE TYPED THIS
YOU NOW HAVE 152 DOLLARS. <- BANKROLL ROUNDED DOWN
PICK 1,2,3,4? 1
BET? 20
SPECIFY R? 2
STOPPED AT 1
YOU WIN $ 40
YOU NOW HAVE 192 DOLLARS.
PICK 1,2,3,4? 99
HOPE YOU HAD FUN. COME BACK SOON.
```

It works.

EXERCISE SET 5.10

On-line:

1. Take YOUR BIG WHEEL program, saved in the last exercise set, and look for bugs. If you find any, insert patches to correct them. This completes the development of YOUR BIG WHEEL program; save it for use in the next section.

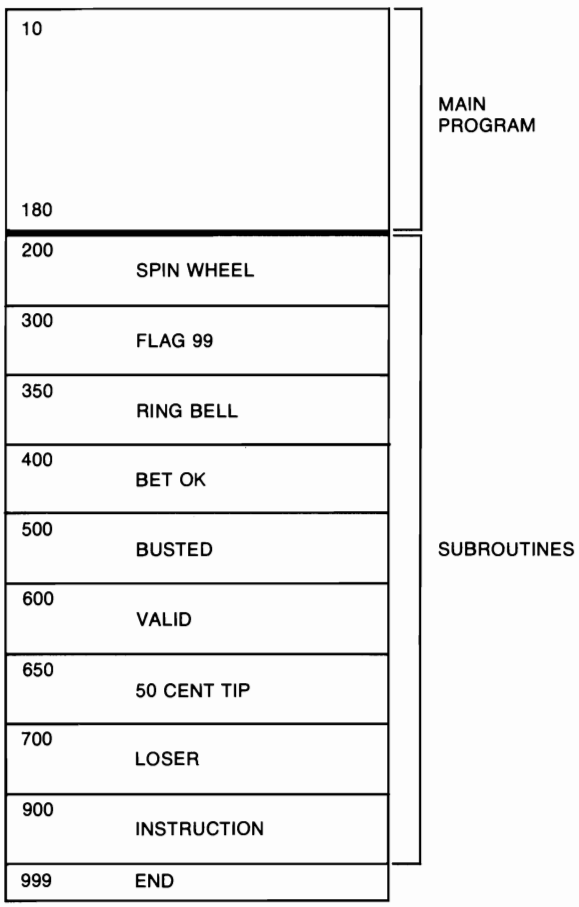
5.11 DOCUMENTATION

Your first big program is now running like a charm. That will make you, and whoever plays your game, happy. It seems as if the job is done. However, a smooth running program is not complete in itself. In school your teacher is concerned about how you wrote the program. If you're a professional programmer in business or industry, other members of your organization may have to modify your program at some future date. After all, you might leave the company. In fact, you yourself might very well want to revise your program at some future date and only then discover that you can't remember how you put it together. In short, you must support or document your program with additional information. Here's a list of the required documentation for a program.

1. *Verbal Description.* Write a paragraph which describes what your program does and how to use it. Are there any limitations on the use of the program?
2. *History of Development.* Outline the steps you followed in developing your program. For example, in BIG WHEEL we first designed a wheel and constructed a payoff table. Then we wrote a subroutine to spin the wheel and assign payoff factors. Next we attached this subroutine to the main program. The resultant program was BIG WHEEL/ELEMENTARY, a thirty-five line program. We then developed subroutines for FLAG 99, BET OK, BUSTED, VALID, RING BELL, LOSER, and

INSTRUCTIONS. We also worked on program personality and layout. Then we discovered a bug and inserted the 50 CENT TIP subroutine. Our final version of BIG WHEEL was a program of over one hundred and thirty lines.

3. *Flow Charts and Listings*. Draw a flow chart for the main program and all subroutines. Accompany each flow chart with a list of the corresponding program steps.
4. *Organizational Chart*. Draw a block diagram of the main program and subroutines. For BIG WHEEL the block diagram looks like this.



5. *Program Listing*. Attach a complete listing of the program with footnotes or remarks, brackets and labels for subroutines, and tags for all GO SUB statements.

In BASIC there is a command which allows you to insert remarks or footnotes in the program listing. The command is REM, an abbreviation for the word remark. Here is how you use it. Look at the FLAG 99 subroutine, beginning at line 300.

```
300 IF N = 99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN.
    COME BACK SOON."
315 STOP
```

To identify to the reader that the FLAG 99 routine is in lines 300 through 315, you insert at line 299.

299 REM FLAG 99 300-315

When you RUN a program, the computer skips over any line in the program which begins with REM. When you LIST a program, however, all REM statements are printed out. To make a REM statement stand out from the other lines, add dashes to make it look like this:

299 REM_____FLAG 99 300-315_____

You can also use REM commands at the beginning of the program to give the name of the program and the programmer, the date the program was written, and the location of the main program and the subroutines.

Here is my BIG WHEEL complete with REM statements and all the routines and features discussed in this chapter. I've bracketed each subroutine and pinpointed with arrows and labels the location of the GO SUB commands which activate the subroutines.

BIG WHEEL ***PROGRAM***

```
 1 REM -----PROGRAM: BIG WHEEL
 2 REM -----WRITTEN BY: EDWIN R. SAGE
 3 REM -----DATE: JANUARY 1974
 4 REM -----MAIN PROGRAM: LINES 10-180
 5 REM -----SUBROUTINES: LINES 200-960
10 RANDOM
12 PRINT TAB(27);"PLAYING BIG WHEEL"
13 PRINT
14 PRINT
15 GO SUB 900 <-------------------- ACTIVATE INSTRUCTION
16 PRINT "PLEASE TELL ME YOUR NAME";
17 INPUT N$
18 PRINT
19 PRINT
20 LET M=INT(51*RND(X))+50
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
31 PRINT
32 PRINT
35 PRINT "---WHEEL IS SPINNING"
40 PRINT "PICK 1,2,3,4";
50 INPUT N
51 GO SUB 300 <-------------------- ACTIVATE FLAG 99
52 GO SUB 600 <-------------------- ACTIVATE VALID
60 PRINT "BET";
70 INPUT B
71 GO SUB 400 <-------------------- ACTIVATE BET OK
80 GO SUB 200 <-------------------- ACTIVATE SPIN WHEEL
90 PRINT "---WHEEL STOPPED AT" W
100 IF W=N THEN 150
110 PRINT "YOU LOSE $" B
111 GO SUB 700 <------------------- ACTIVATE LOSER
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
131 GO SUB 500 <------------------- ACTIVATE BUSTED
140 GO TO 31
150 PRINT "YOU WIN $"P*B
151 GO SUB 350 <------------------- ACTIVATE RING BELL
160 LET M=M+P*B
161 GO SUB 650 <------------------- ACTIVATE 50 CENT TIP
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 31
```

MAIN PROGRAM

```
199 REM ----------SPIN WHEEL 200-275----------
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
299 REM ----------FLAG 99 300-315----------
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN " N$ ". COME BACK SOON."
315 STOP
349 REM ----------RING BELL 350-365----------
350 FOR C=1 TO 5
355 PRINT CHR$(135);
360 NEXT C
365 RETURN
399 REM ----------BET OK 400-460----------
400 IF B>M THEN 455
405 IF B<0 THEN 445
410 IF B<1 THEN 435
415 IF INT(B)=B THEN 430
420 PRINT "WHAT'S THIS CENTS STUFF! BET WHOLE DOLLAR AMOUNTS."
425 GO TO 60
430 RETURN
435 PRINT "DON'T BE CHEAP. BET AT LEAST A DOLLAR."
440 GO TO 60
445 PRINT "STUPID! YOU CAN'T BET A NEGATIVE AMOUNT."
450 GO TO 60
455 PRINT "SMART HUH? YOU DON'T HAVE THAT MUCH MONEY."
460 GO TO 60
499 REM ----------BUSTED 500-555----------
500 IF M=0 THEN 510
505 RETURN
510 PRINT "TOO BAD--YOU LOST ALL THE MONEY I GAVE YOU, BUT"
515 PRINT "I'LL BE GENEROUS AND GIVE YOU ANOTHER CHANCE."
520 PRINT "WOULD YOU LIKE TO PLAY AGAIN";
525 INPUT A$
530 IF A$="YES" THEN 20
535 IF A$="NO" THEN 550
540 PRINT "ANSWER YES OR NO."
545 GO TO 520
550 PRINT "THANKS FOR PLAYING. BETTER LUCK NEXT TIME."
555 STOP
```

SPIN WHEEL

FLAG 99

RING BELL

BET OK

BUSTED

```
599 REM ----------VALID INPUT 600-615----------
600 IF (N-1)*(N-2)*(N-3)*(N-4)=0 THEN 615
605 PRINT "FOLLOW DIRECTIONS DUMMY."
610 GO TO 40
615 RETURN
649 REM ----------50 CENT TIP 650-665----------
650 IF INT(M)=M THEN 665
655 PRINT "I'LL KEEP THE 50 CENTS. THANKS FOR THE TIP."
660 LET M=INT(M)
665 RETURN
699 REM ----------LOSING PERSONALITY 700-770----------
700 LET R=INT(5*RND(X))+1
705 ON R GO TO 765,755,745,735,725
725 PRINT "WHAT BAD LUCK."
730 RETURN
735 PRINT "YOU BLEW IT."
740 RETURN
745 PRINT "I LOVE YOUR MONEY."
750 RETURN
755 PRINT "BETTER LUCK NEXT TIME."
760 RETURN
765 PRINT "THAT'S TOO BAD."
770 RETURN
899 REM ----------INSTRUCTIONS 900-960----------
900 PRINT "DO YOU WANT INSTRUCTIONS";
905 INPUT A$
906 PRINT
907 PRINT
910 IF A$="NO" THEN 930
915 IF A$="YES" THEN 935
920 PRINT "ANSWER YES OR NO."
925 GO TO 900
930 RETURN
935 PRINT "IMAGINE A LARGE CARNIVAL WHEEL, 6 FEET IN DIAMETER,"
936 PRINT "DIVIDED INTO 9 EQUAL PIE-SHAPED PIECES. THREE SECTIONS"
937 PRINT "ARE NUMBERED 1, THREE SECTIONS ARE NUMBERED 2, TWO"
938 PRINT "SECTIONS ARE NUMBERED 3, AND ONE SECTION IS NUMBERED 4."
939 PRINT
940 PRINT "THE WHEEL WILL BE SPUN AND THEN YOU WILL HAVE AN"
941 PRINT "OPPORTUNITY TO BET ON ONE OF THE FOUR NUMBERS"
942 PRINT "BEFORE THE WHEEL COMES TO REST."
943 PRINT
944 PRINT "AT THE START OF EACH GAME, YOU WILL RECEIVE"
945 PRINT "A CERTAIN AMOUNT OF MONEY TO BET IN WHOLE"
946 PRINT "DOLLAR AMOUNTS."
947 PRINT
948 PRINT "SECTION NUMBERS PAYOFF AS FOLLOWS:"
949 PRINT "1   PAYS  2  TIMES YOUR BET."
950 PRINT "2   PAYS  2  TIMES YOUR BET."
951 PRINT "3   PAYS  3.5 TIMES YOUR BET."
952 PRINT "4   PAYS  8  TIMES YOUR BET."
953 PRINT
954 PRINT "WHEN YOU WANT TO QUIT(IF YOU DON'T GO BROKE FIRST)"
955 PRINT "TYPE 99 WHEN ASKED TO PICK 1,2,3,4."
956 PRINT
957 PRINT
958 PRINT "LET'S GO."
959 PRINT
960 RETURN
999 END
```

VALID

50 CENT TIP

LOSER

INSTRUCTIONS

6. *Typical Run.*

   To complete the documentation, attach a run of the final program which illustrates the operation of each subroutine just as I have in the typical runs in this chapter.

EXERCISE SET 5.11

Off-line and On-line:

1. Put together the following items to document your program, YOUR BIG WHEEL.
   A. *Verbal Description.* Write a verbal description of what your game does and how to play it.
   B. *History of Development.* Tell how you developed your program. Draw a picture of your big wheel and attach a copy of the payoff table. If you were to develop the program again, would you follow the same procedure?
   C. *Flow Charts and Listings.* Draw a flow chart of the main program and attach a computer listing of the corresponding program steps. Draw a flow chart of each subroutine and attach a computer listing of the corresponding program steps. Title each flow chart and listing.
   D. *Organizational Chart.* Construct a schematic diagram showing the organization of the program.
   E. *Program Listing.* Attach a complete listing of your program, including REM statements. Bracket and label the main program and subroutines. Use arrows to pinpoint GO SUBS.
   F. *Typical Run.* Attach a typical run.

PLAYING CRAPS

```
DO YOU WANT INSTRUCTIONS? YES


THIS IS A DICE GAME WHERE YOU BET ON THE SUM
OF TWO DICE. I WILL ROLL THE DICE.

IF ON THE FIRST ROLL, THE SUM OF THE DICE IS 7 OR 11,
YOU WIN. ON THE OTHER HAND, IF THE SUM IS 2,3, OR 12,
YOU LOSE. IF THE SUM, HOWEVER, IS NONE OF THESE
NUMBERS: 7,11,2,3,OR 12, THEN THE SUM, ONE OF THE
REMAINING OUTCOMES: 4,5,6,8,9,OR 10, IS KNOWN AS
YOUR 'POINT'.

I WILL THEN ROLL THE DICE REPEATEDLY UNTIL THE SUM
IS EITHER 7 OR YOUR 'POINT'. IF THE SUM IS 7, YOU
AUTOMATICALLY LOSE. IF THE SUM IS YOUR 'POINT',
YOU WIN.

THE PAYOFF IS 1 TO 1.

WHEN YOU WANT TO QUIT(IF YOU DON'T GO BROKE FIRST)
TYPE 999 WHEN ASKED TO BET.


LET'S GO.

YOU HAVE 69 DOLLARS TO PLAY WITH.


I'M ROLLING THE DICE.
BET PLEASE? 35

YOU ROLLED 5 AND 5 . SUM IS 10 .
YOUR 'POINT' IS 10
YOUR NEXT ROLL: 4 AND 6 . SUM IS 10 .
YOU WIN.
 CONGRATULATIONS.
YOU NOW HAVE 104 DOLLARS.


I'M ROLLING THE DICE.
BET PLEASE? 50

YOU ROLLED 1 AND 1 . SUM IS 2 .
YOU LOSE.
I LOVE YOUR MONEY.
YOU NOW HAVE 54 DOLLARS.


I'M ROLLING THE DICE.
BET PLEASE? 999
HOPE YOU HAD FUN. COME BACK SOON.
```

6

COIN AND DICE GAMES

In previous chapters you have seen two or three methods for generating a "random" output. In the Guess-A-Number games in Chapters 2 and 3 you used your own "random" mind, or the computer "looked up" a number in a table of random numbers. In BIG WHEEL the device for generating a random output was the wheel commonly used at carnivals. Over the centuries people have invented many other simple mechanisms for generating random outputs. One is the coin; another is the die, or, if you have two or more, dice. The popularity of the coin and the die are based in part on their portability and in part on their intuitive symmetry. After all, it's hard to argue with a nickel.

First we'll discuss the basic operation of coin tossing, particularly the probability of the outcomes when you toss one or more coins. You will write a program called TWO COIN, which simulates the tossing of two coins and allows a player to bet on the outcome. You will find this new game is a special case of BIG WHEEL, so it will be expedient simply to modify your existing program, YOUR BIG WHEEL.

Next we will discuss the basic operation of die tossing. You will write a program called CRAPS, which simulates the tossing of two dice and allows a player to bet on the outcome. Like TWO COIN, this program has a family relationship to BIG WHEEL, but you'll build it from scratch. In any case, by the time you're through, you'll be a better craps shooter.

While the games in this chapter are used to teach computer programming, you should be aware that these same events, e.g. flipping one or two coins, rolling one or two dice, are commonly used to elucidate probability theory. It is not our purpose to teach probability in this book, however.

6.1 HEADS OR TAILS: TEACHING THE COMPUTER TO TOSS A COIN

Write a program to simulate the tossing of one coin twenty times. When you flip a coin, you assume from its symmetry that there are two possible outcomes, one called heads and the other tails (no edges, please), and that the probability of the coin's landing heads is ½, or .5, and the probability of its landing tails is also ½, or .5. These are the calculated or theoretical probabilities. In real life the evidence may not fit the theory perfectly. Let's see.

If we toss a coin, record the outcome, and toss it again, we expect that, after a great number of tosses, the number of heads obtained will be about the same as the number of tails. Not the same, but close. Further, if there are no defects in the coin, the difference between the number of heads and tails will be reduced as we toss the coin more times. The same is true when you use a computer to toss a coin. The more tosses, the more nearly equal the number of heads and tails will be.

Since there are two equally probable outcomes to a coin toss, we will use the random number generator, RND(X), to pick a number at random between 0 and 1. If the number is less than .5, it's a head. If the number is greater than or equal to .5, it's a tail. You might think that a greater number of tails will occur since .5 has been included. How often will .5 occur? Let's find out. My computer prints out seven digits. How many different seven-digit numbers are there? We can consider a seven-digit number to be composed of seven slots, each to be filled with a digit from 0 to 9.

SLOT

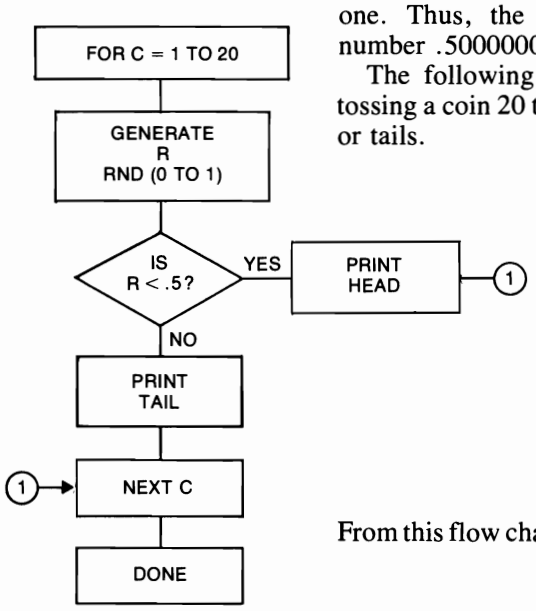
1	2	3	4	5	6	7

SEVEN DIGIT NUMBER

The first slot can be filled in one of ten ways; the second in one of ten ways; and so on. If you multiply the number of ways you can fill the first slot by the number of ways you can fill the second slot, and so on, you get $10*10*10*10*10*10*10$, or 10^7. That's 10,000,000. But since RND(X) cannot generate .0000000 you get 9,999,999 seven-digit numbers, of which .5000000 is one. Thus, the probability of getting the specific number .5000000 is close to zero.

ONE COIN/20
FLOW CHART

The following flow chart outlines the routine for tossing a coin 20 times and printing the outcome, heads or tails.

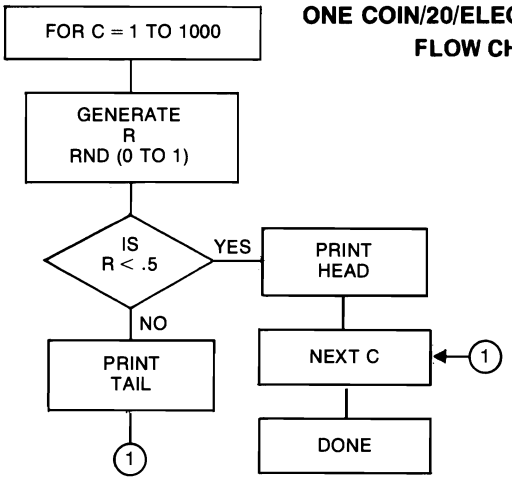


From this flow chart we write the following program.

```
                 1 RANDOM                    PROGRAM
               ┌10 FOR C=1 TO 20
               │ 20 LET R=RND(X)
20 TOSSES      │ 30 IF R<.5 THEN 70
               │ 40 PRINT "TAILS"
               └50 NEXT C
                 60 STOP
                 70 PRINT "HEADS"
                 80 GO TO 50
                 90 END
```

```
HEADS  OUTPUT
HEADS
HEADS
HEADS
TAILS
TAILS
HEADS
HEADS
TAILS
HEADS
TAILS
HEADS
HEADS
TAILS
TAILS
HEADS
HEADS
HEADS
TAILS
HEADS
```

Incidentally, if in the above flow chart the NEXT statement had been put in the YES branch instead of the NO branch, the flow chart would look like this.



The program would look like this.

```
                 1 RANDOM                    PROGRAM
               ┌10 FOR C=1 TO 20
               │ 20 LET R=RND(X)
20 TOSSES      │ 30 IF R<.5 THEN 60
               │ 40 PRINT "TAILS"
               │ 50 GO TO 70
               │ 60 PRINT "HEADS"
               └70 NEXT C
                 80 END
```

Look at the two programs, ONE COIN/20 and ONE COIN/20/ELEGANT, side by side. I've indicated the loop in each with a bracket.

ONE COIN/20

```
                 1 RANDOM                    PROGRAM
               ┌10 FOR C=1 TO 20
               │ 20 LET R=RND(X)
20             │ 30 IF R<.5 THEN 70
TOSSES         │ 40 PRINT "TAILS"
               └50 NEXT C
                 60 STOP
                 70 PRINT "HEADS"
                 80 GO TO 50
                 90 END
```

ONE COIN/20/ELEGANT

```
                 1 RANDOM                    PROGRAM
               ┌10 FOR C=1 TO 20
               │ 20 LET R=RND(X)
               │ 30 IF R<.5 THEN 60
20 TOSSES      │ 40 PRINT "TAILS"
               │ 50 GO TO 70
               │ 60 PRINT "HEADS"
               └70 NEXT C
                 80 END
```

We've saved a step. That's good. As in a geometric proof, the fewer the steps, the more "elegant" the program. In short, if you construct a loop which contains questions, and therefore branches, you can save a step by placing the NEXT statement at the end of the first YES branch. What you save is the STOP statement. ONE COIN/20/ELEGANT not only reduces the length of ONE COIN/20 by one step, but also makes the structure of the program more apparent by putting both PRINT statements inside the loop. So much for saving a step.

Perhaps you would rather have the computer use its random number generator to generate a 1 or 2, 1 meaning heads and 2 meaning tails. To help you remember that 1 is heads and 2 is tails, recall that 1 comes before 2 on the number line, just as *h* comes before *t* in the alphabet. To make this change, we change line 20 to include the INT expression: LET R=INT(2*RND(X))+1.

ONE COIN/1 OR 2

PROGRAM

```
 1 RANDOM
10 FOR C=1 TO 20
20 LET R=INT(2*RND(X))+1
30 IF R=1 THEN 60
40 PRINT "TAILS"
50 GO TO 70
60 PRINT "HEADS"
70 NEXT C
80 END
```

20 TOSSES

OUTPUT

```
TAILS
HEADS
TAILS
HEADS
HEADS
TAILS
HEADS
HEADS
TAILS
HEADS
TAILS
HEADS
TAILS
HEADS
TAILS
TAILS
TAILS
TAILS
HEADS
TAILS
```

In a run of a thousand tosses, it will save space to print H for HEADS and T for TAILS horizontally across the page. We can do this by using the semicolon at the end of lines 40 and 60.

ONE COIN/HT

PROGRAM

```
 1 RANDOM
10 FOR C=1 TO 20
20 LET R=INT(2*RND(X))+1
30 IF R=1 THEN 60
40 PRINT "T";
50 GO TO 70
60 PRINT "H";
70 NEXT C
80 END
```

20 TOSSES

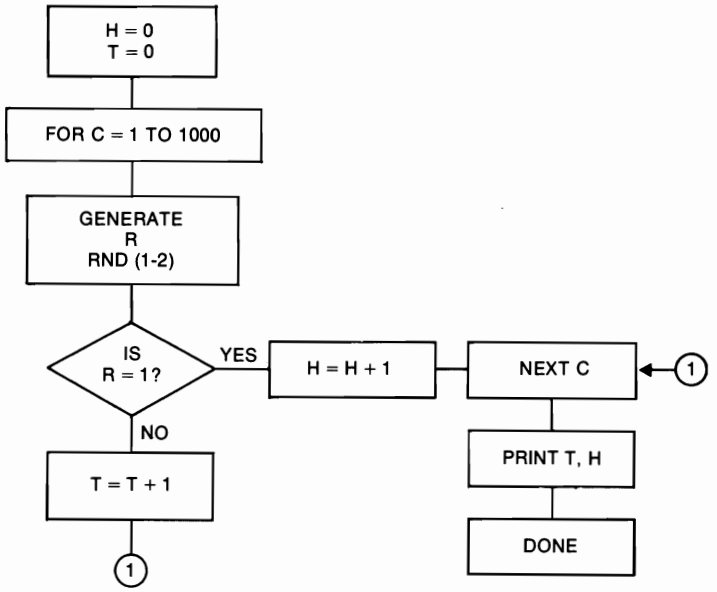
OUTPUT

```
THHHTHHHTTHTHTHHTTTH
```

Look at the above output. There are eleven heads and nine tails. Although it's pretty close, the outcome does not match the theoretical expectation of ten heads and ten tails. Let's flip a coin 1,000 times and see how close the results come to the 50/50 distribution of heads and tails forecast by theory.

Write a program to toss a coin 1,000 times. The computer can simulate the tossing of a coin much faster than the Teletype can print out the results of each toss. Since we're concerned only with the totals after 1,000 tosses, let's put two counters, T and H, in the computer to keep running totals of the number of tails and the number of heads and print the results just once, after 1,000 tosses. Here is the algorithm for 1,000 consecutive tosses in flow chart form:

ONE COIN/1000
FLOW CHART



The following program is encoded from this flow chart.

PROGRAM

```
  1 RANDOM
  5 LET T=0
  6 LET H=0
 10 FOR C=1 TO 1000
 20 LET R=INT(2*RND(X))+1
 30 IF R=1 THEN 60
 40 LET T=T+1
 50 GO TO 70
 60 LET H=H+1
 70 NEXT C
 80 PRINT "HEADS","TAILS"
 90 PRINT H,T
100 END
```

1000 TOSSES

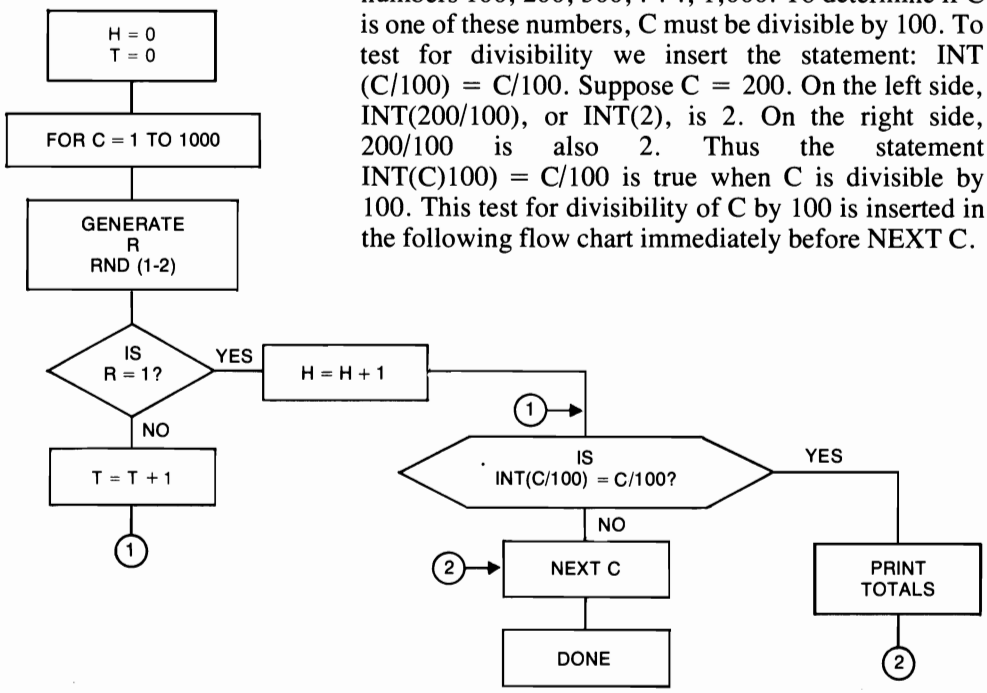
OUTPUT

```
HEADS     TAILS
 489       511
```

Look at the output. The difference between the number of heads and tails is 22. This difference of 22 in 1000 tosses is proportionately less than the difference of 2 in 20 tosses. Thus, the greater the number of tosses the closer the results come to the 50/50 distribution as forecast by theory.

Was the number of tails at each point in the 1,000 tosses greater than the number of heads? To answer this question, instruct the computer to stop momentarily

ONE COIN/1000/100 FLOW CHART



after each 100 tosses, that is when C is one of the numbers 100, 200, 300, . . ., 1,000. To determine if C is one of these numbers, C must be divisible by 100. To test for divisibility we insert the statement: INT (C/100) = C/100. Suppose C = 200. On the left side, INT(200/100), or INT(2), is 2. On the right side, 200/100 is also 2. Thus the statement INT(C)100) = C/100 is true when C is divisible by 100. This test for divisibility of C by 100 is inserted in the following flow chart immediately before NEXT C.

Here is the new program. It gives us the score of heads versus tails after each 100 tosses of the coin.

PROGRAM

```
  1 RANDOM
  5 LET T=0
  6 LET H=0
  8 PRINT "TOSSES","HEADS","TAILS"
 10 FOR C=1 TO 1000
 20 LET R=INT(2*RND(X))+1
 30 IF R=1 THEN 60
 40 LET T=T+1
 50 GO TO 70
 60 LET H=H+1
 70 IF INT(C/100)=C/100 THEN 100
 80 NEXT C
 90 STOP
100 PRINT C,H,T
110 GO TO 80
120 END
```

COLUMN HEADING

1000 TOSSES

PRINT TOTALS

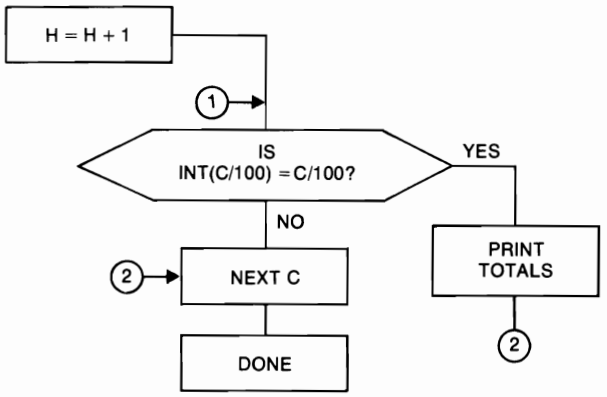
OUTPUT

TOSSES	HEADS	TAILS
100	43	57
200	104	96
300	158	142
400	211	189
500	261	239
600	315	285
700	359	341
800	404	396
900	453	447
1000	507	493

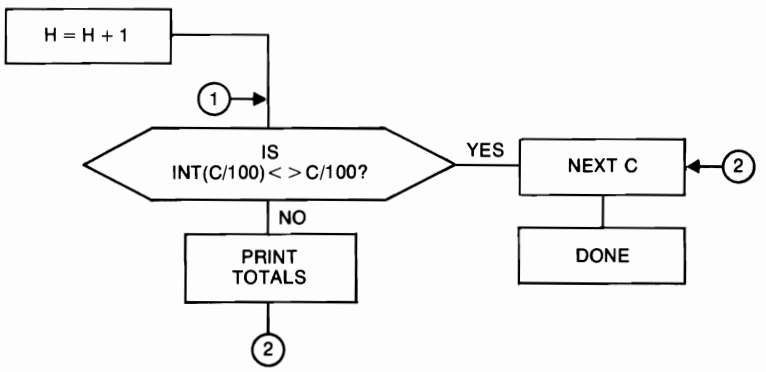
The PRINT statements in lines 8 and 100 both call for three items to be printed. Since the items in each case are separated by commas, they will be printed in the first three of the five columns into which the computer divides the page.

How does the distribution of heads and tails after 1,000 tosses compare with the distribution of heads and tails obtained using the program ONE COIN/1000? By chance, this time there are more heads than tails.

Look again at the FIRST YES BRANCH in the flow chart ONE COIN/1000/100.



As I explained in developing ONE COIN/20/ELEGANT, the number of steps in the overall program can be reduced if NEXT C is placed at the end of the YES branch. To accomplish this in the above flow chart, however, the form of the question must be changed. The present question asks if INT(C/100) is equal to C/100. The new question will ask if INT(C/100) is *not* equal to C/100. The phrase, is not equal to, is usually represented by an equal sign with a slash through it ($\neq$). This symbol does not exist on the Teletype, however. Instead we use the two symbols for "less than" (<) and "greater than" (>) together, thus, < >.



The use of negations in a flow chart usually requires more careful thought than positive statements. If we ask the question: Is INT(C/100) < >C/100? the answer is

YES when C is *not* divisible by 100 and NO when C *is* divisible by 100. For example, suppose the counter, C, is 325. Then INT(325/100) < >325/100 becomes 3< >3.25, which is true, and the answer to: Is INT(325/100) < > 325/100? is YES. The counter, C, is not divisible by 100, so the computer moves on to NEXT C. On the other hand, if C is 500, then INT(500/100)< >500/100 becomes 5< >5 which is false. Five *does* equal five, and the answer to: Is INT (C/100)< >C/100? is NO. Since C is divisible by 100, the computer prints the score. The negative approach is encoded this way:

```
 70 IF INT(C/100)< >C/100 THEN 90
 80 PRINT C,H,T
 90 NEXT C
100 END
```

These four steps replace lines 70 through 120 in the old program, ONE COIN/1000/100. You save two steps, obtaining a more elegant program.

Here is a complete listing of the program using the negative approach. A bracket shows the loop. As before, all three branches now fall within the loop.

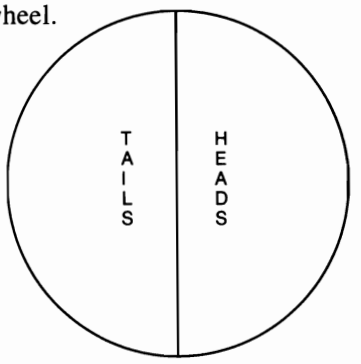
ONE COIN/1000/100/ELEGANT

PROGRAM

```
 1 RANDOM
 5 LET T=0
 6 LET H=0
 8 PRINT "TOSSES","HEADS","TAILS"
10 FOR C=1 TO 1000
20 LET R=INT(2*RND(X))+1
30 IF R=1 THEN 60
40 LET T=T+1 <------ COUNT HEADS BRANCH
50 GO TO 70
60 LET H=H+1 <------ COUNT TAILS BRANCH
70 IF INT(C/100)<>C/100 THEN 90
80 PRINT C,H,T <------ PRINT TOTALS BRANCH
90 NEXT C
100 END
```

1000 TOSSES

So much for tossing one coin. In a few minutes we'll have the computer toss two coins. In the meantime, has it occurred to you that tossing a coin is equivalent to spinning a big wheel which has been divided into two equal sections, one labeled heads, the other tails? Here is such a wheel.



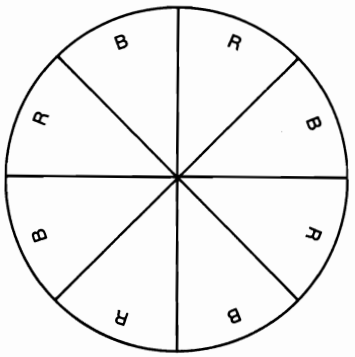
This means a program to simulate 1000 spins of the above wheel is the same as ONE COIN/1000/100. We

need only change the column heading TOSSES to SPINS. In general, to generate any two outcomes which occur with equal probability, instruct the computer to generate the numbers 1 and 2 at random and then assign an appropriate label to each number.

EXERCISE SET 6.1

On-line:

1. Draw a flow chart and write the most elegant program possible to simulate the spinning of the following wheel 1000 times.



There are two outcomes when the wheel comes to rest, red or black. Have your program produce an output similar to this:

SPINS	RED	BLACK
100	54	46
200	105	95
300	161	139
400	203	197
500	252	248
600	309	291
700	356	344
800	405	395
900	452	448
1000	500	500

2. Program the computer to simulate this "real life" event: A pail contains nine colored tennis balls. Three are colored red, three green, two blue, and one yellow. Draw a ball without peeking, look at the color, and keep track of the outcome. Put the ball back in the pail. Draw again, look at the color, and note the outcome. Repeat this procedure 900 times. After every 100 draws record on a piece of paper the total number of red, green, blue and yellow balls you have drawn to that point. Predict the number of balls of each color you will have after 900 draws. To write your program you may wish to "construct" a Big Wheel which is divided into nine equal pie-shaped pieces, one to represent each of the nine

tennis balls. Your program should produce an output like this:

DRAWS	RED	GREEN	BLUE	YELLOW
100	38	29	26	7
200	75	60	45	20
300	109	98	62	31
400	136	137	84	43
500	166	178	101	55
600	201	206	125	68
700	236	236	148	80
800	271	267	174	88
900	314	295	190	101

6.2 TOSSING TWO COINS

An easy way to generate a more complex set of random outcomes is to toss *two* coins, this way: Take a coin, toss it and record the outcome, heads or tails. Then take the same coin or a second coin, toss it and record the outcome. The set of *four* possible outcomes is:

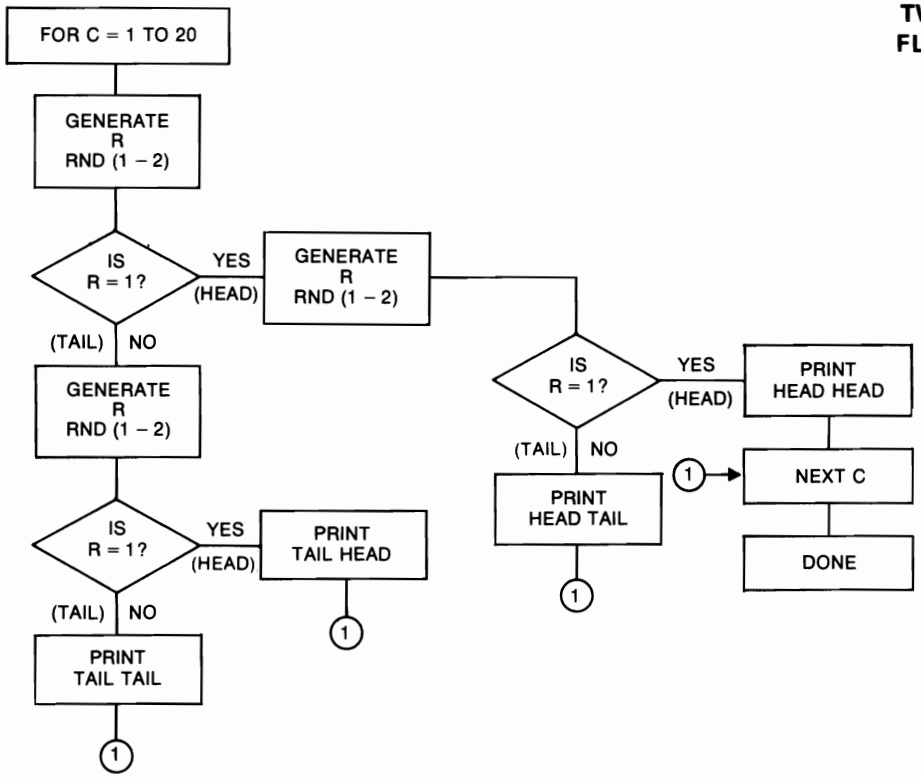
FIRST TOSS	SECOND TOSS	OUTCOME	
HEAD	HEAD	HEAD	HEAD
HEAD	TAIL	HEAD	TAIL
TAIL	HEAD	TAIL	HEAD
TAIL	TAIL	TAIL	TAIL

Perhaps you already know the probability of each outcome. But perhaps you don't. I'll show you how to use the computer to find the answer. We will program the computer to simulate the repeated tossing of two coins, followed each time by the recording of the outcome. The tossing and recording will be done 1000 times. After 1000 tosses, the experimental probability of each of the four outcomes can be obtained by dividing the number of times each outcome occurred by 1000. As in tossing one coin, the more often you toss two coins, the closer the experimental values will be to the theoretical values. Here's how it's done.

Write a program to toss two coins twenty times.

We still make use of the fact that, when a coin is tossed, the probability of each event, head or tail, is 1/2. To "toss" the first coin, instruct the computer to generate a random number, R, which is a 1 or 2. This number is tested for head, R = 1, or for tail, R = 2. Then, to "toss" the second coin, instruct the computer to again generate a 1 or 2 at random. It is also tested for head, R = 1, or for tail, R = 2. At this point the first trial is complete; one of the four outcomes has occurred. This process is repeated again for a total of twenty trials. Four branches are required in the following flow chart. (HEAD) and (TAIL) are used to help you keep track of the outcome of each question.

TWO COIN/20
FLOW CHART



Look at the flow chart. Suppose the computer generates a 1 as the first value for R. The answer to the first question is YES; the outcome is a head. The computer immediately generates a second value for R. Suppose it is again a 1. The answer to the second question is YES; the outcome is again a head. The computer then prints HEAD HEAD. Here's the program:

```
    1 RANDOM
   10 FOR C=1 TO 20
   20 LET R=INT(2*RND(X))+1
   30 IF R=1 THEN 100
   40 LET R=INT(2*RND(X))+1
   50 IF R=1 THEN 80
   60 PRINT "TAIL TAIL"
   70 GO TO 150
   80 PRINT "TAIL HEAD"
   90 GO TO 150
  100 LET R=INT(2*RND(X))+1
  110 IF R=1 THEN 140
  120 PRINT "HEAD TAIL"
  130 GO TO 150
  140 PRINT "HEAD HEAD"
  150 NEXT C
  160 END
```

20 TRIALS

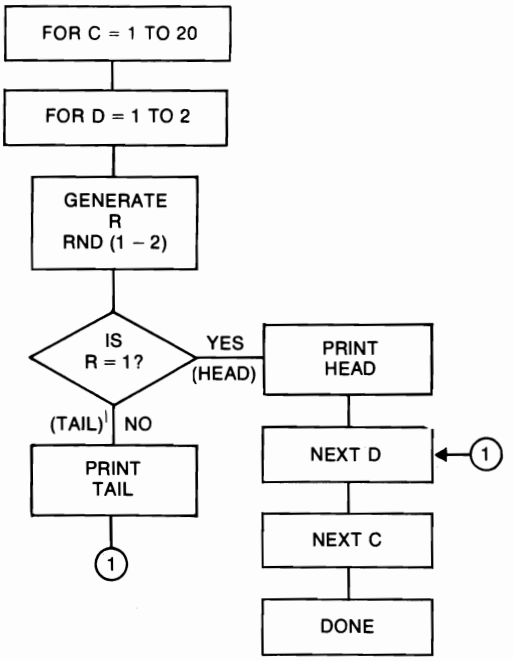
PROGRAM OUTPUT

HEAD	TAIL
TAIL	HEAD
TAIL	HEAD
TAIL	TAIL
TAIL	HEAD
TAIL	TAIL
TAIL	TAIL
TAIL	HEAD
HEAD	TAIL
HEAD	TAIL
HEAD	HEAD
TAIL	HEAD
HEAD	TAIL
TAIL	HEAD
TAIL	TAIL
HEAD	HEAD
HEAD	HEAD
TAIL	HEAD
HEAD	TAIL
TAIL	HEAD

The program TWO COIN/20 can be shortened by using only one expression instead of three to generate a random number. The expression is LET R = INT (2*RND(X)) + 1. To execute it twice, once for each

toss, we'll create a loop. Since C is used to designate the number-of-trials counter, we'll use D to designate the number-of-tosses counter. Here is the flow chart.

TWO COIN/20/2-LOOP FLOW CHART



Assume that on the first trial, the toss of the first coin yields R = 1 and the toss of the second coin yields R = 2. Play computer, following the instructions in the above flow chart, to determine the outcome. So much for the flow chart. Here is the program and a typical run.

20 TRIALS
2 TOSSES

```
PROGRAM   1 RANDOM
         10 FOR C=1 TO 20
         20 FOR D=1 TO 2
         30 LET R=INT(2*RND(X))+1
         40 IF R=1 THEN 70
         50 PRINT "TAIL ";
         60 GO TO 80
         70 PRINT "HEAD ";
         80 NEXT D
         90 PRINT
        100 NEXT C
        110 END
```

```
OUTPUT HEAD HEAD
       TAIL TAIL
       HEAD TAIL
       HEAD HEAD
       HEAD TAIL
       HEAD TAIL
       TAIL HEAD
       TAIL HEAD
       TAIL HEAD
       HEAD HEAD
       HEAD HEAD
       TAIL TAIL
       HEAD TAIL
       TAIL TAIL
       TAIL HEAD
       TAIL TAIL
       TAIL TAIL
       TAIL HEAD
       TAIL TAIL
       HEAD TAIL
```

Play computer again, this time following the instructions in the above program. As above, assume that, on the first trial, the toss of the first coin yields R = 1 and the toss of the second coin yields R = 2. Don't get hung

up at line 80 NEXT D when you have exhausted the two tosses. When the inner D loop is satisfied, you, the computer, just go to the next line, in this case 90. Look at lines 50 and 70. The semicolon is used so that, after the first outcome, HEADS or TAILS, is typed, the outcome of the second toss will be typed on the same line. The blank space within the quotes at the end of the word TAILS or HEADS separates the two words when they are printed on the same line. What is the function of line 90? On each line we want only the two outcomes that constitute one trial. Each time the D loop is satisfied, the PRINT statement on line 90 signals the computer to stop printing on that particular line and to "linefeed" to the next one.

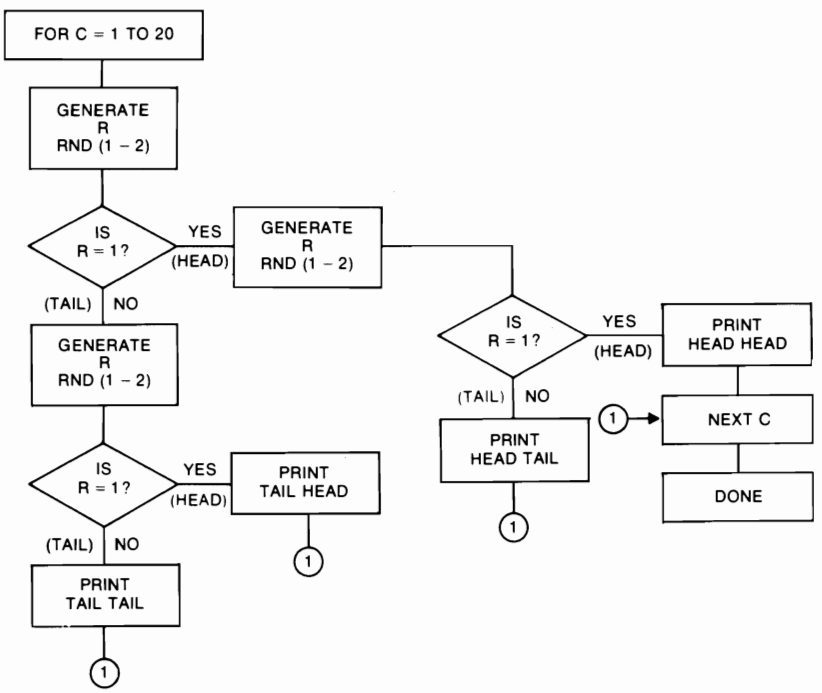
So much for the program. Now let's look at the output. The results are summarized in the following table.

OUTCOME		FREQUENCY
HEAD	HEAD	4
HEAD	TAIL	5
TAIL	HEAD	5
TAIL	TAIL	6

It looks as if each outcome occurs about 25 percent of the time. But, as in the one-coin experiment, we would have more confidence in this generality if we had more data.

Write a program to toss two coins 1000 times. Look again at the flow chart for TWO COIN/20.

TWO COIN/20
FLOW CHART



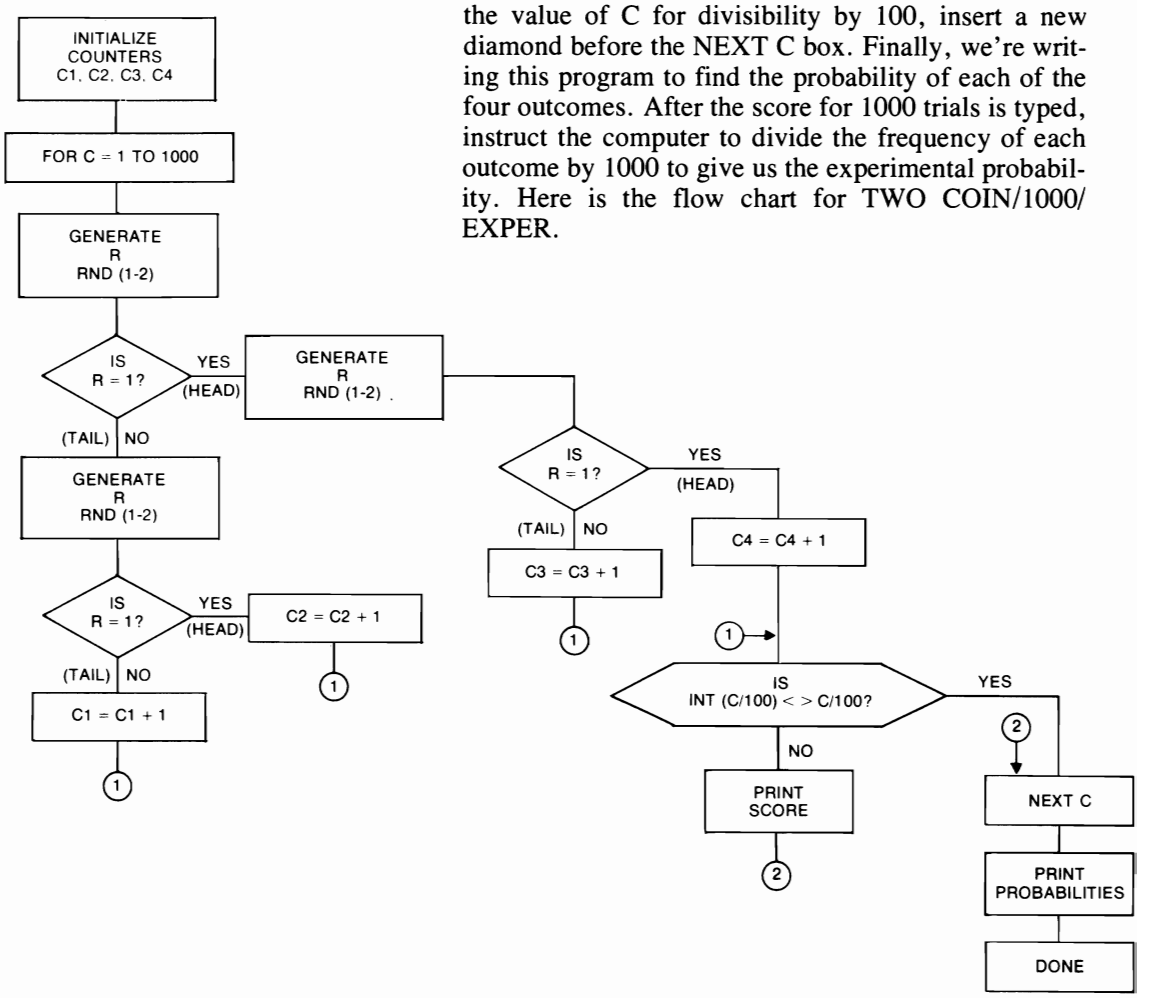
We are interested, not in a trial by trial analysis of the outcomes, but in the total number of TAIL TAIL, TAIL

HEAD, HEAD TAIL, and HEAD HEAD outcomes in 1000 trials. Thus, the PRINT statements can be eliminated and in their place counters can be inserted to count the number of times each outcome has occurred. The new counters are:

(1) C1 counts the number of TAIL TAIL outcomes.
(2) C2 counts the number of TAIL HEAD outcomes.
(3) C3 counts the number of HEAD TAIL outcomes.
(4) C4 counts the number of HEAD HEAD outcomes.

A second change is to replace FOR C = 1 TO 20 with FOR C = 1 TO 1000 in the first box of the flow chart. However we're interested not only in the outcomes after 1000 trials but in the score after each 100 trials. To test the value of C for divisibility by 100, insert a new diamond before the NEXT C box. Finally, we're writing this program to find the probability of each of the four outcomes. After the score for 1000 trials is typed, instruct the computer to divide the frequency of each outcome by 1000 to give us the experimental probability. Here is the flow chart for TWO COIN/1000/EXPER.

TWO COIN/1000/EXPER FLOW CHART



Look at the new diamond in which C is tested for divisibility by 100. We've used the negative approach. The following program is encoded from this flow chart.

PROGRAM

```
 1 RANDOM
 5 LET C1=0
 6 LET C2=0
 7 LET C3=0
 8 LET C4=0
 9 PRINT "TOSSES","TAIL TAIL","TAIL HEAD","HEAD TAIL","HEAD HEAD"
10 FOR C=1 TO 1000
20 LET R=INT(2*RND(X))+1
30 IF R=1 THEN 100
40 LET R=INT(2*RND(X))+1
50 IF R=1 THEN 80
60 LET C1=C1+1 <-------- TAIL TAIL COUNTER
70 GO TO 150
80 LET C2=C2+1 <-------- TAIL HEAD COUNTER
90 GO TO 150
100 LET R=INT(2*RND(X))+1
110 IF R=1 THEN 140
120 LET C3=C3+1 <-------- HEAD TAIL COUNTER
130 GO TO 150
140 LET C4=C4+1 <-------- HEAD HEAD COUNTER
150 IF INT(C/100)<>C/100 THEN 170
160 PRINT C,C1,C2,C3,C4 <-- PRINT SCORE
170 NEXT C
180 PRINT "PROBABILITY",C1/1000,C2/1000,C3/1000,C4/1000 <-- FINAL RESULTS
190 END
```

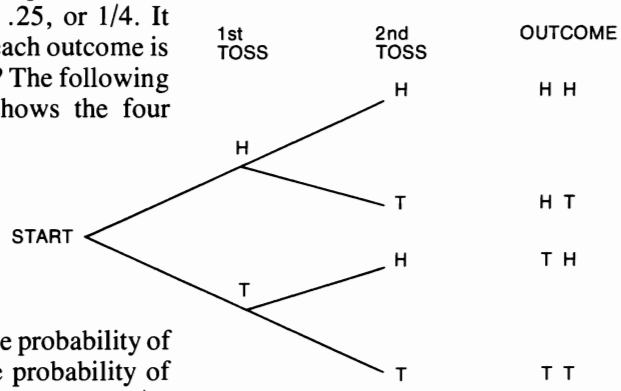
1000 TRIALS

Suppose the first value of R is 1 and the second value is 2; play computer to determine which outcome occurs, that is, which counter, C1, C2, C3, or C4, is increased by 1.

OUTPUT

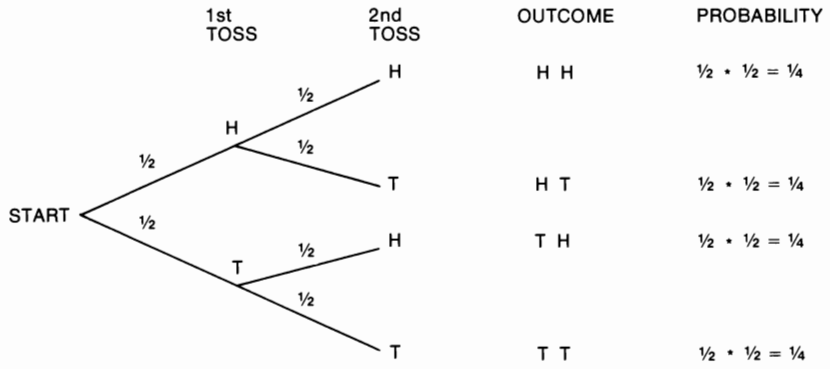
TOSSES	TAIL TAIL	TAIL HEAD	HEAD TAIL	HEAD HEAD
100	25	16	28	31
200	45	51	46	58
300	61	76	70	93
400	85	102	94	119
500	115	127	120	138
600	140	154	145	161
700	170	177	172	181
800	193	207	198	202
900	220	228	227	225
1000	245	251	252	252
PROBABILITY	.245	.251	.252	.252

Look at the last line of the output. As expected, each one of the decimal numbers is close to .25, or 1/4. It looks as if the theoretical probability of each outcome is in fact .25. Can we show this is the case? The following diagram, known as a tree diagram, shows the four possible outcomes.



When the first coin is tossed, we know the probability of a HEAD or TAIL is 1/2. Likewise, the probability of obtaining a HEAD or TAIL with the second coin is 1/2.

Whatever happens on the first toss has no influence on the outcome of the second toss. Since this is a two-coin experiment, the probability of an outcome such as HEAD HEAD is the probability of a HEAD on the first toss times the probability of a HEAD on the second toss. In the following tree diagram, the probability of each branch is shown. To the right is a list of the outcomes with their probabilities, obtained by multiplying the probabilities along the branches.



The theoretical probability of each outcome is 1/4.

The last program, TWO COIN/1000/EXPER, arrived at the four outcomes by simulating the real world experiment of tossing two coins repeatedly. In that program I used my knowledge of the theoretical probability of a HEAD or a TAIL when *one* coin is tossed. Now that I know the theoretical probabilities of the four outcomes, HH, HT, TH and TT, I want to write a new, shorter program which generates the outcomes, not by "tossing" two coins, but by using the theoretical probabilities. I will instruct the computer, on each trial, to generate at random one of four equally likely numbers: 1, 2, 3, or 4. The following convention will be used for the assignment of outcomes.

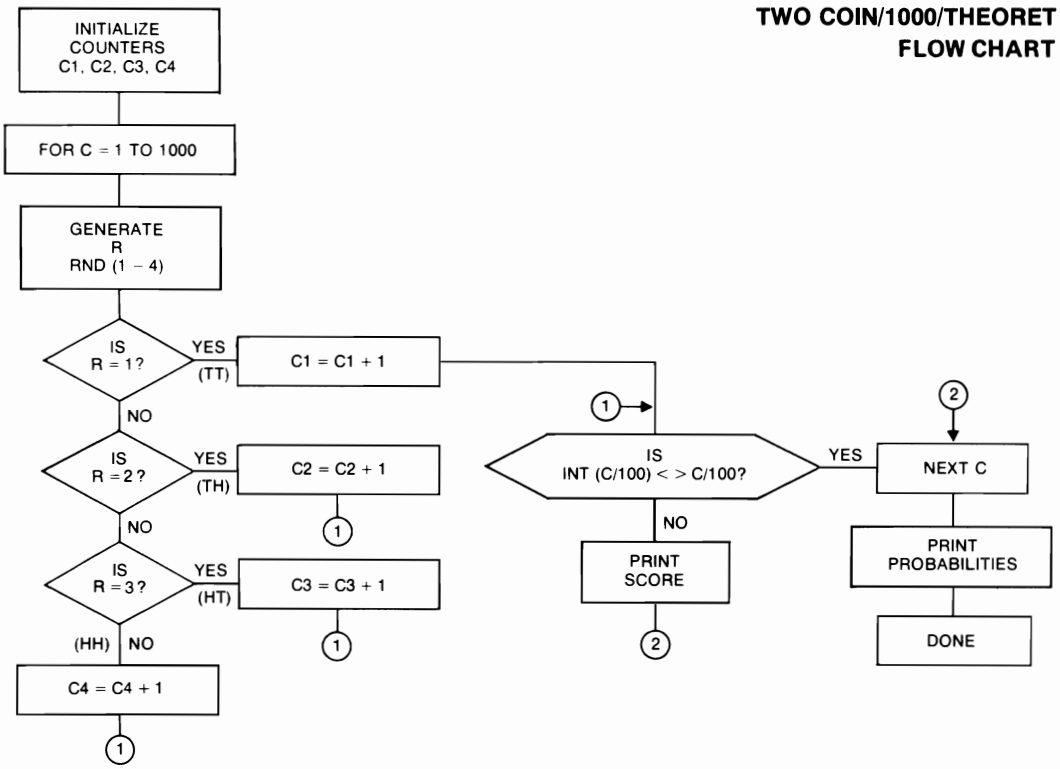
(1) If R = 1 then the outcome TAIL TAIL has occurred.
(2) If R = 2 then the outcome TAIL HEAD has occurred.
(3) If R = 3 then the outcome HEAD TAIL has occurred.
(4) If R = 4 then the outcome HEAD HEAD has occurred.

As in TWO COIN/1000/EXPER, the following counters will be used to count each outcome.

(1) C1 counts the number of TAIL TAIL outcomes.
(2) C2 counts the number of TAIL HEAD outcomes.
(3) C3 counts the number of HEAD TAIL outcomes.
(4) C4 counts the number of HEAD HEAD outcomes.

Here is the flow chart.

TWO COIN/1000/THEORET
FLOW CHART



Look at the flow chart. The computer generates a number, R, at random, tests the number to determine the outcome, and then adds 1 to the appropriate counter. Here is the program.

PROGRAM

```
  1 RANDOM
  5 LET C1=0
  6 LET C2=0
  7 LET C3=0
  8 LET C4=0
  9 PRINT "TOSSES","TAIL TAIL","TAIL HEAD","HEAD TAIL","HEAD HEAD"
 10 FOR C=1 TO 1000
 20 LET R=INT(4*RND(X))+1
 30 IF R=1 THEN 120
 40 IF R=2 THEN 100
 50 IF R=3 THEN 80
 60 LET C4=C4+1 <-------- HEAD HEAD COUNTER
 70 GO TO 130
 80 LET C3=C3+1 <-------- HEAD TAIL COUNTER
 90 GO TO 130
100 LET C2=C2+1 <-------- TAIL HEAD COUNTER
110 GO TO 130
120 LET C1=C1+1 <-------- TAIL TAIL COUNTER
130 IF INT(C/100)<>C/100 THEN 150
140 PRINT C,C1,C2,C3,C4 <-------- PRINT SCORE
150 NEXT C
160 PRINT "PROBABILITY",C1/1000,C2/1000,C3/1000,C4/1000 <-- FINAL RESULTS
170 END
```

1000 TRIALS

Look at the program. Suppose the computer randomly generates the number 3, as instructed in line 20. Which

outcome does 3 represent? Now play computer. Which counter is increased by 1? So much for the program. Here is a typical run.

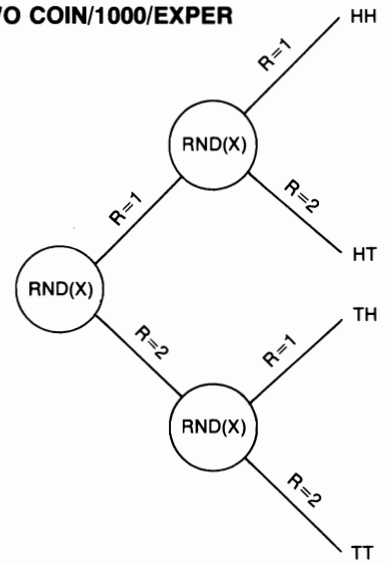
OUTPUT

TOSSES	TAIL TAIL	TAIL HEAD	HEAD TAIL	HEAD HEAD
100	19	34	18	29
200	42	56	49	53
300	71	81	71	77
400	105	99	97	99
500	124	124	127	125
600	152	147	150	151
700	169	179	179	173
800	203	198	208	191
900	221	223	239	217
1000	246	248	260	246
PROBABILITY	.246	.248	.26	.246

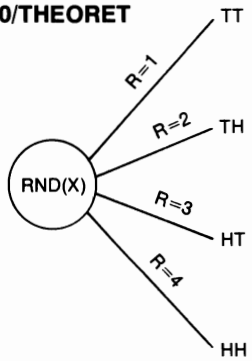
Look at the last line of the output. The results are close to the theoretical probabilities. Using the known probabilities of the outcomes is just as good as "tossing" two coins. Either approach works.

Here is a schematic representation of the two simulation programs.

TWO COIN/1000/EXPER



TWO COIN/1000/THEORET



In the first program, TWO COIN/1000/EXPER, the random number generator, RND(X), is used at two stages. In TWO COIN/1000/THEORET it's used only once, but selects from four numbers. Both methods have their place in real life. The multiple stage approach is used when, because of the complexity of the system, the theoretical probabilities of the final outcomes are not known and are virtually impossible to calculate. The analysis of a complex traffic network is an example. The probability of a car turning left, right or going straight ahead at each of several hundred intersections might be known from direct measurement. You're interested, however, in the overall impact on a bridge, tunnel, or other exit from the network.

The second, single stage method is used when, based on experience or theory, you know the probabilities of the final outcomes, and may be interested for example in betting on the outcome one trial at a time.

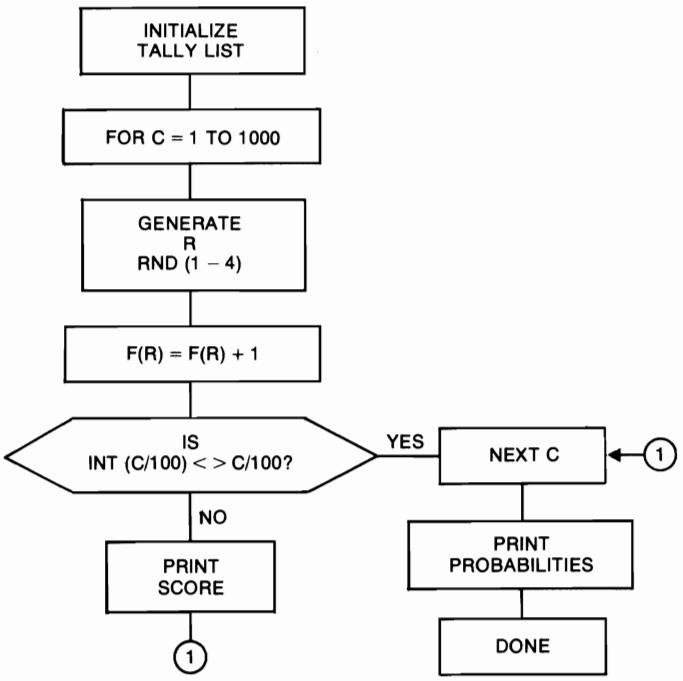
At this point both programs are about the same length. TWO COIN/1000/EXPER has 25 steps and TWO COIN/1000/THEORET has 23. The program which uses the known probabilities of the final outcomes, e.g. TWO COIN/1000/THEORET, has an advantage, however, in that a subscripted variable can be used to set up a tally list replacing the four counters, C1, C2, C3, and C4. This will give us a program of only 14 steps, 11 steps shorter than TWO COIN/1000/EXPER! Recall the use of a tally list when programming the computer to play 1000 COMPUTER GUESS games at the end of Chapter 3. This time four boxes will be set up in the computer's memory to replace the four counters, and labeled as follows:

F(1) holds the frequency of TAIL TAIL outcomes.
F(2) holds the frequency of TAIL HEAD outcomes.
F(3) holds the frequency of HEAD TAIL outcomes.
F(4) holds the frequency of HEAD HEAD outcomes.

F(1)	F(2)	F(3)	F(4)
0	0	0	0

At the outset, the value in each box will be zero. When LET R = INT(4*RND(X))+ 1 is executed and a value for R is generated, say 2, a TAIL HEAD has occurred. The counter F(2) is increased by 1. We do not need three IF-THEN statements to find out what R is. We go on to the next step, LET F(R) = F(R) +1. Here is the flow chart for TWO COIN/1000/TALLY.

TWO COIN/1000/TALLY FLOW CHART



The following program is encoded from this flow chart.

PROGRAM

```
    1 RANDOM
    2 DIM F(4)
INITIALIZE TALLY LIST
    5 FOR I=1 TO 4
    6 LET F(I)=0
    7 NEXT I
    9 PRINT "TOSSES","TAIL TAIL","TAIL HEAD","HEAD TAIL","HEAD HEAD"
1000 TRIALS
   10 FOR C=1 TO 1000
   20 LET R=INT(4*RND(X))+1
   30 LET F(R)=F(R)+1 <---------- TALLY STATEMENT
   40 IF INT(C/100)<>C/100 THEN 60
   50 PRINT C,F(1),F(2),F(3),F(4)
   60 NEXT C
   70 PRINT "PROBABILITY",F(1)/1000,F(2)/1000,F(3)/1000,F(4)/1000
   80 END
```

Look at line 2. The DIM statement is used to save four boxes for the subscripted variable. Now a zero must be put in each of the four boxes. This is done in lines 5, 6, and 7. The variable I is used to designate the box number, and I takes on the values 1 to 4.

Suppose the values of the counters in the computer's memory are

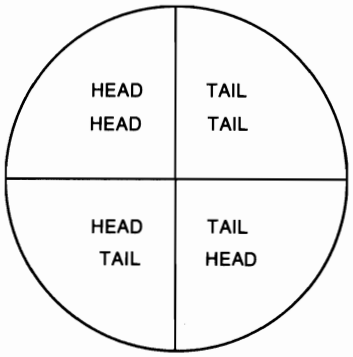
C	F(1)	F(2)	F(3)	F(4)
200	48	47	51	53

as the computer approaches trial 200. Trial 200 begins and the computer generates a 4 for R in line 20. Play computer. Add 1 to the proper counter and proceed to line 40. Where do you go from there? To verify that the program operates as intended, here is a typical run.

OUTPUT

TOSSES	TAIL TAIL	TAIL HEAD	HEAD TAIL	HEAD HEAD
100	22	27	28	23
200	51	57	49	43
300	80	80	79	61
400	103	106	94	97
500	129	135	111	125
600	152	163	136	149
700	176	190	159	175
800	204	213	185	198
900	223	234	211	232
1000	255	253	239	253
PROBABILITY	.255	.253	.239	.253

So much for the use of the subscripted variable and tally list in a two-coin simulation. The same set of outcomes can be obtained by spinning a Big Wheel divided into four equal sections and labeled as follows.



To simulate the use of the above wheel, instruct the computer to generate at random a 1, 2, 3 or 4, each number corresponding to one of these two-coin outcomes.

EXERCISE SET 6.2

Off-line:

1. Draw a flow chart for tossing *three* coins 20 times. Your flow chart will have eight branches.
2. Write a three-loop program to simulate the tossing of three coins 20 times, imitating the procedure used in TWO COIN/20/2-LOOP.
3. Draw a tree diagram to show all the possible outcomes when three coins are tossed. Label each branch with its probability. Determine the outcome obtained at the end of each branch. Find the probability of each outcome by multiplying the probabilities along each branch.

On-line:

4. Write a program which uses the theoretical probabilities determined in exercise 3 above to generate, trial by trial, the outcomes when three coins are tossed 800 times. Your program should imitate the procedure used in TWO COIN/1000/TALLY and produce an output similar to this one:

TOSSES	TTT	TTH	THT	THH	HTT	HTH	HHT	HHH	*OUTPUT*
100	14	11	14	13	10	13	12	13	
200	30	21	26	32	21	21	24	25	
300	45	34	43	46	34	27	36	35	
400	53	46	56	55	47	42	53	48	
500	63	59	66	68	59	53	67	65	
600	76	68	83	84	66	68	77	78	
700	88	85	90	100	73	78	92	94	
800	101	99	98	112	86	93	104	107	

6.3 THE TWO COIN GAME

In this section you will modify an old program, BIG WHEEL/FLAG 99, to play a new game called TWO COIN. The game is played as follows. The computer flips two coins and gives the player a chance to bet on the number of heads. From the previous section, the

four outcomes and their probabilities are:

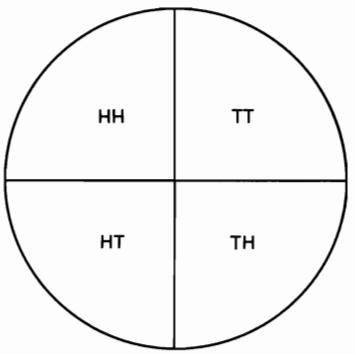
Outcome		Number of Heads	Probability
HEAD	HEAD	2	1/4
HEAD	TAIL	1	1/4
TAIL	HEAD	1	1/4
TAIL	TAIL	0	1/4

The total number of heads is either 2, 1, or 0. The probability of 2 heads is 1/4, of 1 head 1/2 and of 0 heads 1/4. The player picks either 2, 1 or 0 heads. The computer compares the player's input with its outcome to determine whether the player has won or lost.

Look at the above table and complete the following table of payoff factors for each outcome.

OUTCOME NUMBER OF HEADS	NUMBER WAYS TO WIN	NUMBER WAYS TO LOSE	PAYOFF FACTOR
2	1	3	3
1			
0			

The game TWO COIN is just like playing BIG WHEEL using the wheel shown here.



Since the game TWO COIN can be played with the above wheel, this suggests that the program for TWO COIN could be similar in structure to the program for BIG WHEEL. Let's look at a copy of BIG WHEEL/ FLAG 99.

PROGRAM **BIG WHEEL/FLAG 99**

MAIN PROGRAM

```
 10 RANDOM
 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1,2,3,4."
 20 LET M=100
 30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
 40 PRINT "PICK 1,2,3,4";
 50 INPUT N
 51 GO SUB 300
 60 PRINT "BET";
 70 INPUT B
 80 GO SUB 200
 90 PRINT "STOPPED AT" W
100 IF N=W THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

SPIN WHEEL

```
200 LET R=INT(9*RND(X))+1
205 IF R<=3 THEN 265
210 IF R<=6 THEN 250
215 IF R<=8 THEN 235
220 LET W=4
225 LET P=8
230 RETURN
235 LET W=3
240 LET P=3.5
245 RETURN
250 LET W=2
255 LET P=2
260 RETURN
265 LET W=1
270 LET P=2
275 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
999 END
```

What changes would you make in BIG WHEEL/FLAG 99 to make it play the game of TWO COIN, producing

TWO COIN

an output that looks like this?

```
OUTPUT  IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 0,1,2,HEADS
        YOU HAVE 100 DOLLARS TO PLAY WITH.
        PICK 0,1,2,HEADS? 1
        BET? 10
GAME 1  HEAD HEAD
        NUMBER OF HEADS IS 2
        YOU LOSE $ 10
        YOU NOW HAVE 90 DOLLARS.
        PICK 0,1,2,HEADS? 1
        BET? 20
GAME 2  HEAD HEAD
        NUMBER OF HEADS IS 2
        YOU LOSE $ 20
        YOU NOW HAVE 70 DOLLARS.
        PICK 0,1,2,HEADS? 1
        BET? 40
GAME 3  HEAD TAIL
        NUMBER OF HEADS IS 1
        YOU WIN $ 40
        YOU NOW HAVE 110 DOLLARS.
        PICK 0,1,2,HEADS? 1
        BET? 20
GAME 4  TAIL HEAD
        NUMBER OF HEADS IS 1
        YOU WIN $ 20
        YOU NOW HAVE 130 DOLLARS.
        PICK 0,1,2,HEADS? 99
        HOPE YOU HAD FUN. COME BACK SOON.
```

As in the past, we could draw a new flow chart as a guide to setting up the new program, but you are sufficiently experienced now to use the program BIG WHEEL/FLAG 99 as a guide. First we'll make the necessary changes in the main program of BIG WHEEL/FLAG 99. Only four changes are required:

(1) Since the new game is a coin game in which the player is betting on 0, 1, or 2 heads, the instruction in line 15 must be changed. Replace 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 1, 2, 3, 4" with 15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 0, 1, 2 HEADS."

(2) Replace 40 PRINT "PICK 1,2,3,4"; with 40 PRINT "PICK 0,1,2,HEADS";.

(3) In BIG WHEEL the computer tells the player the number, W, at which the wheel stops. Similarly in TWO COIN, it must tell him the number of heads, H, that occurred. H should be given a value in the COIN TOSS subroutine. Replace 90 PRINT "NUMBER IS" W with 90 PRINT "NUMBER OF HEADS IS" H.

(4) To compare the number of heads guessed by the player with the number of heads obtained by the computer, replace 100 IF W = N THEN 150 with 100 IF H = N THEN 150. Here is the main program from BIG WHEEL/FLAG 99 with the foregoing changes. The arrows point to alterations.

MAIN PROGRAM **TWO COIN**

```
  10 RANDOM
→15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 0,1,2,HEADS"
  20 LET M=100
  30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
→40 PRINT "PICK 0,1,2,HEADS";
  50 INPUT N
  51 GO SUB 300
  60 PRINT "BET";
  70 INPUT B
  80 GO SUB 200
→90 PRINT "NUMBER OF HEADS IS" H
→100 IF H=N THEN 150
 110 PRINT "YOU LOSE $" B
 120 LET M=M-B
 130 PRINT "YOU NOW HAVE" M "DOLLARS."
 140 GO TO 40
 150 PRINT "YOU WIN $" P*B
 160 LET M=M+P*B
 170 PRINT "YOU NOW HAVE" M "DOLLARS."
 180 GO TO 40
```

So much for the changes in the main program. Now to make the necessary changes in the "spin wheel" subroutine. Since the TWO COIN game is based on four equally likely outcomes, HH, HT, TH, TT, you want to change the wheel from nine sections to four sections. First, line 200 must be replaced by: 200 LET R = INT(4*RND(X))+1, which generates one of the whole numbers 1, 2, 3 or 4. Next we want to assign to each of the four numbers one of the outcomes. Here is one way to do it:

R	OUTCOME
1	TT
2	TH
3	HT
4	HH

Having generated one of the numbers 1, 2, 3 or 4, the computer must find out which number was generated. This requires three IF-THEN statements.

```
205 IF R = 1 THEN _____
210 IF R = 2 THEN _____
215 IF R = 3 THEN _____
220
```

If the computer falls through to line 220, R is 4, and the outcome must be HEAD HEAD.

As we do in writing a program from a flow chart, we encode one branch at a time. Because there are four outcomes, there will be four branches. Let's start with the branch/outcome R = 4, or HH. To tell the player the outcome of the toss, add this print statement:

```
220 PRINT "HEAD HEAD"
```

In order to compare the player's guess, the input on line 50, with the outcome of the toss, the number of heads obtained on the toss is represented by the variable H in the following new line:

```
225 LET H = 2
```

The next step is to set the payoff factor, P, at 3.

```
230 LET P = 3
```

To finish the branch you write:

```
235 RETURN
```

To sum up, lines 220 to 235 below comprise the HH branch of the COIN TOSS, ex-SPINWHEEL, subroutine. You can fill in the steps for the other three branches. I'll give you some hints.

```
200 LET R = INT(4*RND(X))+1
DETERMINE R
205 IF R = 1 THEN _____
210 IF R = 2 THEN _____
215 IF R = 3 THEN _____
HH BRANCH
220 PRINT "HEAD  HEAD"
225 LET H = 2
230 LET P = 3
235 RETURN
HT BRANCH
240 PRINT "_____" <-- OUTCOME IF R = 3
245 LET H = _____ <-- NUMBER OF HEADS
250 LET P = _____ <-- PAYOFF FACTOR — LOOK AT TABLE
255 RETURN
TH BRANCH
260 PRINT "_____" <-- OUTCOME IF R = 2
265 LET H = _____
270 LET P = _____
275 RETURN
TT BRANCH
280 PRINT "_____" <-- OUTCOME IF R = 1
285 LET H = _____
290 LET P = _____
295 RETURN
```

Did you fill in the blanks in the IF-THEN statements? Your completed COIN TOSS subroutine should look like this:

COIN TOSS
SUBROUTINE

```
200 LET R=INT(4*RND(X))+1
DETERMINE R
205 IF R=1 THEN 230
210 IF R=2 THEN 260
215 IF R=3 THEN 240
HH BRANCH
220 PRINT "HEAD HEAD"
225 LET H=2
230 LET P=3
235 RETURN
HT BRANCH
240 PRINT "HEAD TAIL"
245 LET H=1
250 LET P=1
255 RETURN
TH BRANCH
260 PRINT "TAIL HEAD"
265 LET H=1
270 LET P=1
275 RETURN
TT BRANCH
280 PRINT "TAIL TAIL"
285 LET H=0
290 LET P=3
295 RETURN
```

So much for the alterations. Now attach the new COIN TOSS subroutine to the revised main program. This gives us the new program, which I'll call TWO COIN/FLAG 99.

PROGRAM **TWO COIN/FLAG 99**

MAIN PROGRAM

```
10 RANDOM
15 PRINT "IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 0,1,2,HEADS"
20 LET M=100
30 PRINT "YOU HAVE" M "DOLLARS TO PLAY WITH."
40 PRINT "PICK 0,1,2,HEADS";
50 INPUT N
51 GO SUB 300 <-------------------- ACTIVATE 99
60 PRINT "BET";
70 INPUT B
80 GO SUB 200 <-------------------- ACTIVATE COIN TOSS
90 PRINT "NUMBER OF HEADS IS" H
100 IF H=N THEN 150
110 PRINT "YOU LOSE $" B
120 LET M=M-B
130 PRINT "YOU NOW HAVE" M "DOLLARS."
140 GO TO 40
150 PRINT "YOU WIN $" P*B
160 LET M=M+P*B
170 PRINT "YOU NOW HAVE" M "DOLLARS."
180 GO TO 40
```

COIN TOSS

```
200 LET R=INT(4*RND(X))+1
205 IF R=1 THEN 230
210 IF R=2 THEN 260
215 IF R=3 THEN 240
220 PRINT "HEAD HEAD"
225 LET H=2
230 LET P=3
235 RETURN
240 PRINT "HEAD TAIL"
245 LET H=1
250 LET P=1
255 RETURN
260 PRINT "TAIL HEAD"
265 LET H=1
270 LET P=1
275 RETURN
280 PRINT "TAIL TAIL"
285 LET H=0
290 LET P=3
295 RETURN
```

FLAG 99

```
300 IF N=99 THEN 310
305 RETURN
310 PRINT "HOPE YOU HAD FUN. COME BACK SOON."
315 STOP
999 END
```

Here is a typical run of this program.

```
IF YOU WISH TO QUIT, TYPE 99 AFTER PICK 0,1,2,HEADS
YOU HAVE 100 DOLLARS TO PLAY WITH.
PICK 0,1,2,HEADS? 2
BET? 10
HEAD TAIL
NUMBER OF HEADS IS 1
YOU LOSE $ 10
YOU NOW HAVE 90 DOLLARS.
PICK 0,1,2,HEADS? 1
BET? 20
TAIL TAIL
NUMBER OF HEADS IS 0
YOU LOSE $ 20
YOU NOW HAVE 70 DOLLARS.
PICK 0,1,2,HEADS? 99
HOPE YOU HAD FUN. COME BACK SOON.
```

GAME 1

GAME 2

EXERCISE SET 6.3

On-Line:

1. Take the program TWO COIN/FLAG 99 and modify it as follows:
   - A. Add a subroutine to make sure the player typed in 0, 1, or 2 for the number of heads.
   - B. Add a subroutine to make sure the bet (a) does not exceed his bankroll, (b) is not a negative amount, and (c) is a whole number of dollars.
   - C. Add a subroutine to determine if the player lost all his money.
   - D. Add a subroutine to give instructions.
   - E. Add personality to your program. Make sure it has a good layout or format.
   - F. Document your program using REM statements.

6.4 TEACHING THE COMPUTER TO ROLL A DIE

The die is a clever, portable little device for generating at random six equally likely outcomes. Each of the six faces has a different number of dots on it; the number of dots ranges from 1 to 6. To simulate the rolling of a die, instruct the computer to generate at random a number between 1 and 6. Here is a program that simulates the rolling of a die 20 times.

ONE DIE/20

PROGRAM

```
   1 RANDOM
  10 FOR C=1 TO 20
  20 LET R=INT(6*RND(X))+1
  30 PRINT "I ROLL" R
  40 NEXT C
  50 END
```

20 ROLLS

OUTPUT

```
I ROLL 4
I ROLL 2
I ROLL 4
I ROLL 5
I ROLL 3
I ROLL 3
I ROLL 4
I ROLL 5
I ROLL 3
I ROLL 5
I ROLL 3
I ROLL 6
I ROLL 6
I ROLL 6
I ROLL 1
I ROLL 4
I ROLL 3
I ROLL 5
I ROLL 5
I ROLL 5
```

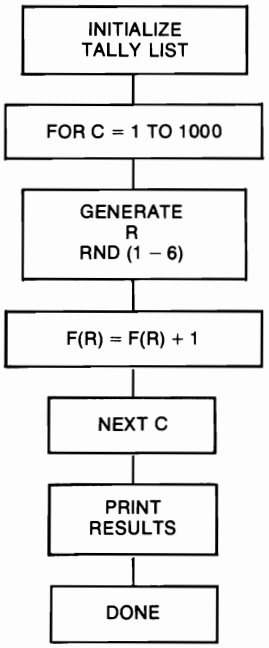
As we did with a coin, let's calculate the theoretical probability of each outcome. There are six possible outcomes, 1, 2, 3, 4, 5, or 6, and each is just as likely to occur as any other. The theoretical probability of any particular number occurring on the roll of a die is therefore 1/6 or .1666667 (correct to 7 decimal places). To check the accuracy of our computer simulation, let's roll a die 1000 times and keep track of the frequency of occurrence of 1, 2, 3, 4, 5, and 6. We'll divide each frequency by 1000 and compare the result to the theoretical probability of each outcome. The programming technique to be used here, the tally list, was used earlier in TWO COIN/1000/TALLY.

To count the numbers of 1's, 2's, 3's, 4's, 5's, and 6's as they occur, we will set aside at the start of the program six storage boxes labeled and initialized as follows:

F(1)	F(2)	F(3)	F(4)	F(5)	F(6)
0	0	0	0	0	0

The box F(1) contains the frequency with which 1 has occurred; the box F(2) contains the frequency with which 2 has occurred; etc. Each time a "roll" occurs (the equivalent of picking at random a number between 1 and 6), the computer is instructed to add 1 to the box representing the outcome. If R is the number generated, then the statement LET F(R) = F(R) + 1 causes 1 to be added to the appropriate box. Here is a flow chart of this procedure.

ONE DIE/1000
FLOW CHART



In the flow chart, the first step, INITIALIZE TALLY LIST, instructs the computer to put zeros in all the boxes. In the following program the DIM statement in line 2 tells the computer to set aside six boxes for the tally list. Initialization is accomplished by a loop between lines 3 and 5, in which I holds a place for the value of the subscript. Any other letter could have been used.

Look at the second loop. After the computer rolls the die one thousand times, the second loop is satisfied and the computer drops to line 50 which prints the column headings. The boxes now contain the total frequencies for each outcome. The third loop, between lines 60 and 80, prints the frequencies one box at a time. Here is the program followed by a typical run.

PROGRAM

```
 1 RANDOM
 2 DIM F(6)
 3 FOR I=1 TO 6
 4 LET F(I)=0
 5 NEXT I
10 FOR C=1 TO 1000
20 LET R=INT(6*RND(X))+1
30 LET F(R)=F(R)+1
40 NEXT C
50 PRINT "DIE NUMBER","FREQUENCY","PROBABILITY"
60 FOR I=1 TO 6
70 PRINT I,F(I),F(I)/1000
80 NEXT I
90 END
```

INITIALIZE TALLY LIST (lines 3–5)

1000 ROLLS (lines 10–40)

TALLY STATEMENT (line 30)

PRINT RESULTS (lines 60–80)

OUTPUT

DIE NUMBER	FREQUENCY	PROBABILITY
1	158	.158
2	174	.174
3	157	.157
4	162	.162
5	179	.179
6	170	.17

Look at the output. Each of the numbers in the probability column is close to the theoretical probability of .1666667. We conclude that the program ONE DIE/1000 can closely approximate the theoretical probabilities of the six outcomes of a die.

6.5 ROLLING TWO DICE

An easy way to produce a more complex set of outcomes is to roll *two* of our handy, portable random number generators, or dice. What are the possible outcomes? The outcome of the first die is one of the numbers 1, 2, 3, 4, 5, 6. Likewise the outcome of the second die is any one of the same six numbers. Without much fanfare or explanation here is a program to simulate the rolling of two dice 20 times, printing the outcome after each roll. The random number generator is used twice, in lines 20 and 30, once for each die. In line 40 the outcome is printed.

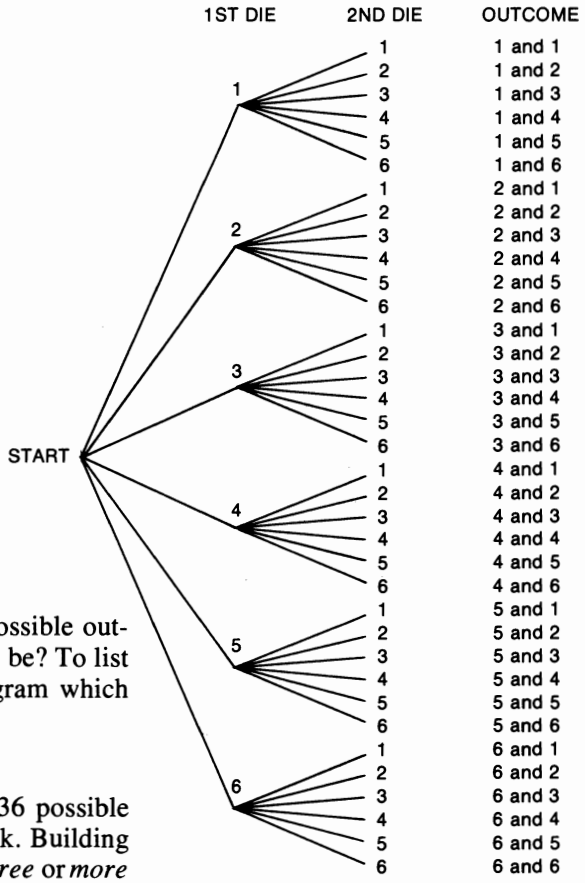
TWO DICE/20

PROGRAM

```
 1 RANDOM
10 FOR C=1 TO 20
20 LET D1=INT(6*RND(X))+1
30 LET D2=INT(6*RND(X))+1
40 PRINT "I ROLL" D1 "AND" D2
50 NEXT C
60 END
```

20 ROLLS (lines 10–50)

```
I ROLL 1 AND 5   OUTPUT
I ROLL 6 AND 6
I ROLL 2 AND 1
I ROLL 3 AND 1
I ROLL 4 AND 3
I ROLL 4 AND 1
I ROLL 5 AND 4
I ROLL 2 AND 5
I ROLL 6 AND 1
I ROLL 2 AND 6
I ROLL 6 AND 4
I ROLL 1 AND 5
I ROLL 5 AND 3
I ROLL 4 AND 5
I ROLL 6 AND 6
I ROLL 2 AND 2
I ROLL 5 AND 1
I ROLL 6 AND 6
I ROLL 2 AND 6
I ROLL 2 AND 3
```



Look at the output. Does it show all the possible outcomes? No. How many outcomes can there be? To list all the possible outcomes draw a tree diagram which looks like this.

Constructing a tree diagram showing the 36 possible outcomes of rolling two dice is a tedious task. Building one to list all possible outcomes of rolling *three* or *more* dice is just too much to ask of a human. But it's a good job for our dumb friend, the computer. The following short program instructs the computer to list the 36 possible outcomes of rolling two dice. The procedure followed by the computer parallels the procedure I used when I constructed the tree diagram: (1) The computer records one of the six possible outcomes of rolling the first die, then generates each of the six possible outcomes to be obtained with the second die. (2) It then records another one of the six possible outcomes of rolling the first die, and generates the six possible outcomes to be obtained with the second die. And so on. This procedure suggests two loops, one for the first die and another for the second. The first loop assigns a sequence of numbers, 1 to 6, to the first die.

```
10 FOR D1 = 1 TO 6
    •
    •
    •
50 NEXT D1
```

We put the second loop inside the first loop. It generates a sequence of numbers, 1 to 6, for the second die each

time a new number is assigned to the first die. The nested loops look like this.

```
10 FOR D1 = 1 TO 6
20 FOR D2 = 1 TO 6
     •
     •
     •
40 NEXT D2
50 NEXT D1
```

Inside both loops we insert a PRINT statement to tell us the values of D1 and D2 at that point. Here's the complete program.

TWO DICE/OUTCOMES

PROGRAM

```
 5 PRINT "DIE 1","DIE 2"
10 FOR D1=1 TO 6
20 FOR D2=1 TO 6
30 PRINT D1,D2
40 NEXT D2
50 NEXT D1
60 END
```

DIE 1 / DIE 2

OUTPUT

DIE 1	DIE 2
1	1
1	2
1	3
1	4
1	5
1	6
2	1
2	2
2	3
2	4
2	5
2	6
3	1
3	2
3	3
3	4
3	5
3	6
4	1
4	2
4	3
4	4
4	5
4	6
5	1
5	2
5	3
5	4
5	5
5	6
6	1
6	2
6	3
6	4
6	5
6	6

To sum up, no matter how you generate the list, there are 36 possible outcomes. Since the 36 outcomes are equally likely to occur, the probability of any one is 1/36. This probability of the complex outcome, that is 1/36, can also be obtained by multiplying the probability of any outcome of one die, that is 1/6, by the probability of any outcome of the second die, which is also 1/6.

In many dice games the sum of the dice is the focal point, not the individual outcomes of each die. What are the possible sums when two dice are rolled? Look at either the tree diagram or the computer-generated list and find the sum of each pair of numbers. The following table is a compact way to record the sum of each pair.

		DIE 1					
		1	2	3	4	5	6
	1	2	3	4	5	6	7
	2	3	4	5	6	7	8
DIE 2	3	4	5	6	7	8	9
	4	5	6	7	8	9	10
	5	6	7	8	9	10	11
	6	7	8	9	10	11	12

Better yet, modify TWO DICE/OUTCOMES to find the sum of D1 and D2. Insert 25 LET S= D1 + D2, and add the column heading SUM to line 5, and the variable S to line 30.

PROGRAM

```
 5 PRINT "DIE 1","DIE 2","SUM"
10 FOR D1=1 TO 6
20 FOR D2=1 TO 6
25 LET S=D1+D2
30 PRINT D1,D2,S
40 NEXT D2
50 NEXT D1
60 END
```

DIE 1 / DIE 2

TWO DICE/SUM
OUTPUT

DIE 1	DIE 2	SUM
1	1	2
1	2	3
1	3	4
1	4	5
1	5	6
1	6	7
2	1	3
2	2	4
2	3	5
2	4	6
2	5	7
2	6	8
3	1	4
3	2	5
3	3	6
3	4	7
3	5	8
3	6	9
4	1	5
4	2	6
4	3	7
4	4	8
4	5	9
4	6	10
5	1	6
5	2	7
5	3	8
5	4	9
5	5	10
5	6	11
6	1	7
6	2	8
6	3	9
6	4	10
6	5	11
6	6	12

Look at either the table or output of TWO DICE/SUM. The different sums are: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12. Two is the sum of 1 and 1, better known as "snake eyes." Twelve is the sum of 6 and 6, also known as "box cars." We now determine the probability of each sum.

What is the probability of rolling a sum of 2? Look at the table and count the number of times 2 occurs. The sum 2 occurs only once. Since there are 36 possible outcomes, the probability of a sum of 2 is 1/36. What is the probability of rolling a 5? Look at the table and count the number of times 5 occurs. The sum 5 occurs four times out of 36. The probability of a sum of 5 is 4/36. To find the probability of the other sums, make a frequency table listing the different sums and the number of times each occurs. It will look like this.

Sum	Frequency	Probability
2	1	
3	2	
4	3	
5	4	
6	5	
7	6	
8	5	
9	4	
10	3	
11	2	
12	1	

In the third column of the above table, enter as a fraction the probability of each sum. Generally, the probability of an outcome is expressed as a decimal number between 0.0 and 1.0. Convert the above fractions to decimals, with the help of the computer. Here is a program that performs the necessary calculations, giving answers correct to seven decimal places.

HELP

PROGRAM

```
 5 PRINT "NUMBER","NUMBER/36"
10 FOR C=1 TO 6
20 PRINT C,C/36
30 NEXT C
40 END
```

OUTPUT

```
NUMBER          NUMBER/36
 1               .02777778
 2               .05555556
 3               .08333333
 4               .1111111
 5               .1388889
 6               .1666667
```

The sums, frequencies, and associated probabilities are summarized in the following table.

Sum	Frequency	Probability
2	1	1/36 or .02777778
3	2	2/36 or .05555556
4	3	3/36 or .08333333
5	4	4/36 or .1111111
6	5	5/36 or .1388889
7	6	6/36 or .1666667
8	5	5/36 or .1388889
9	4	4/36 or .1111111
10	3	3/36 or .08333333
11	2	2/36 or .05555556
12	1	1/36 or .02777778

This table can be produced entirely by the computer if we further modify the program TWO DICE/SUM. TWO DICE/SUM already lists the 36 possible combinations of two dice and the sum of each. We only need to add a routine to compute the frequency with which each sum occurs, and the decimal probability of each sum occurring. Here is TWO DICE/SUM, showing the two loops that generate the die values.

PROGRAM **TWO DICE/SUM**

```
DIE 1     5 PRINT "DIE 1","DIE 2","SUM"
          10 FOR D1=1 TO 6
DIE 2     20 FOR D2=1 TO 6
          25 LET S=D1+D2
          30 PRINT D1,D2,S
          40 NEXT D2
          50 NEXT D1
          60 END
```

First, change the column headings by replacing line 5 with 5 PRINT "SUM", "FREQUENCY", "PROBABILITY". Next, pull out line 30 and replace it with a line which keeps track of the sum obtained in line 25. The line looks like this: 30 LET F(S) = F(S)+1 What we have done is to set up, once again, a tally list in the computer. The tally list used here looks like this.

F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)	F(9)	F(10)	F(11)	F(12)
0	0	0	0	0	0	0	0	0	0	0	0

In this program, however, since no sum of two dice is ever 1, we'll not use the box labelled F(1). When you use a subscripted variable to set up a tally list, you need a DIM statement and an initializing loop. The DIM statement instructs the computer to set aside the required number of "boxes." In this case, you need twelve boxes. The DIM statement looks like this: 1 DIM F(12). The loop to initialize the contents of the 12 boxes at zero looks like this:

```
2 FOR I = 1 TO 12
3 LET F(I) = 0
4 NEXT I
```

So far our revised program looks like this:

TWO DICE/SUM/REVISED

PROGRAM

```
 1 DIM F(12)
 2 FOR I=1 TO 12
 3 LET F(I)=0
 4 NEXT I
 5 PRINT "SUM","FREQUENCY","PROBABILITY"
10 FOR D1=1 TO 6
20 FOR D2=1 TO 6
25 LET S=D1+D2
30 LET F(S)=F(S)+1
40 NEXT D2
50 NEXT D1
60 END
```

INITIALIZE TALLY LIST (lines 2–4)

DIE 1 (lines 10–50)

DIE 2 (lines 20–40)

CALCULATE SUM (line 25)

TALLY STATEMENT (line 30)

Finally, after the 36 outcomes have been generated and the two loops are satisfied, another loop must be added to output the new results: sum, frequency, and probability. Remember no sum is 1; therefor S starts at 2.

```
60 FOR S = 2 TO 12
70 PRINT S, F(S), F(S)/36
80 NEXT S
```

Here is the fully revised program to produce the table.

TWO DICE/TABLE ***PROGRAM***

```
 1 DIM F(12)
 2 FOR I=1 TO 12
 3 LET F(I)=0
 4 NEXT I
 5 PRINT "SUM","FREQUENCY","PROBABILITY"
10 FOR D1=1 TO 6
20 FOR D2=1 TO 6
25 LET S=D1+D2
30 LET F(S)=F(S)+1
40 NEXT D2
50 NEXT D1
60 FOR S=2 TO 12
70 PRINT S,F(S),F(S)/36
80 NEXT S
90 END
```

INITIALIZE TALLY LIST (lines 2–4)

DIE 1 (lines 10–50)

DIE 2 (lines 20–40)

CALCULATE SUM (line 25)

TALLY STATEMENT (line 30)

S STARTS AT 2 (line 60)

OUTPUT LOOP (lines 60–80)

OUTPUT

SUM	FREQUENCY	PROBABILITY
2	1	.02777778
3	2	.05555556
4	3	.08333333
5	4	.1111111
6	5	.1388889
7	6	.1666667
8	5	.1388889
9	4	.1111111
10	3	.08333333
11	2	.05555556
12	1	.02777778

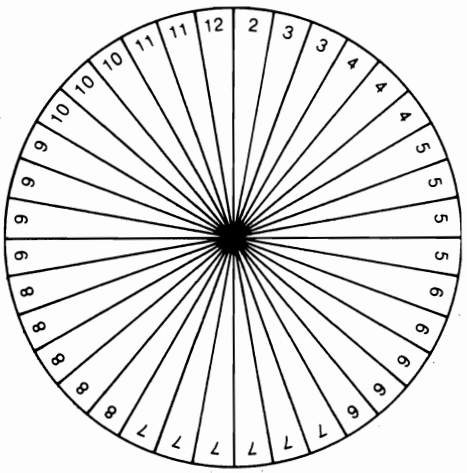
Look at the table. The numbers in the probability column are all decimal numbers between 0 and 1. A craps shooter might find these difficult to memorize. However, look in the frequency column. What is the pattern? The pattern is a sequence of numbers which starts at 1, counts to 6, and then counts back down to 1 again! This

pattern is easy to remember. We will look for the pattern 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1 in the outputs of the next two programs.

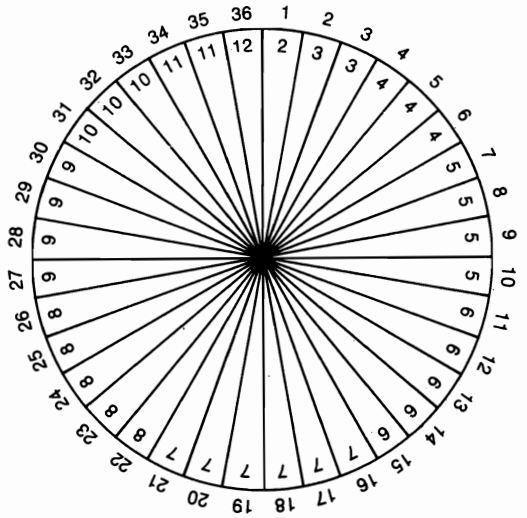
To this point, all we've done is use the computer as a tool to calculate the theoretical probabilities of the sums of two dice. Now that we know the probabilities, we are ready to roll two dice many times.

As in the flipping of two coins, there are two approaches to simulating the tossing of two dice. One approach is to generate at random the sums 2 through 12 in proportion to their theoretical probability of occurrence. A second way is to simulate the toss of each die separately, add the two results to obtain the sum, and repeat the process as many times as required.

Write a program to simulate the tossing of two dice 3600 times. In the first program, a rather long one, I will use the theoretical probabilities just determined to dictate the frequency of occurrence of the eleven different sums. Since all the sums *are not* equally likely, we cannot just generate the numbers 2 to 12 at random. Each of these numbers must occur with a frequency to reflect its probability. To help you visualize the scheme I will use, draw a Big Wheel and divide it into 36 equal sections. Each of the 36 sections will be numbered with one of the numbers 2 to 12. Each number, 2 to 12, will appear on the wheel as many times as indicated in the table above, e.g. there will be four sections numbered 5, five sections numbered 6, etc.



Let's write a subroutine which will cause each sum to occur with its corresponding probability. To do this we will assign each section of the wheel a number from 1 to 36.



The computer will generate at random a section number between 1 and 36. Each section number will be assigned an outcome or sum as shown below.

SECTION NUMBER	SUM
1	2
2, 3	3
4, 5, 6	4
7, 8, 9, 10	5
11, 12, 13, 14, 15,	6
16, 17, 18, 19, 20, 21	7
22, 23, 24, 25, 26	8
27, 28, 29, 30	9
31, 32, 33	10
34, 35	11
36	12

As the first step in the subroutine, a random number between 1 and 36 is generated. Call it R. R must then be tested to determine which sum it corresponds to. Ten IF-THEN statements will be needed to do this. Once the corresponding sum is found, it is designated S so that further operations can be performed in the main program. Here is the subroutine beginning at line 200.

TWO DICE/3600/THEORET
SUBROUTINE

```
200 LET R=INT(36*RND(X))+1
205 IF R<=1 THEN 355
210 IF R<=3 THEN 345
215 IF R<=6 THEN 335
220 IF R<=10 THEN 325
225 IF R<=15 THEN 315
230 IF R<=21 THEN 305
235 IF R<=26 THEN 295
240 IF R<=30 THEN 285
245 IF R<=33 THEN 275
250 IF R<=35 THEN 265
255 LET S=12
260 RETURN
265 LET S=11
270 RETURN
275 LET S=10
280 RETURN
285 LET S=9
290 RETURN
295 LET S=8
300 RETURN
305 LET S=7
310 RETURN
315 LET S=6
320 RETURN
325 LET S=5
330 RETURN
335 LET S=4
340 RETURN
345 LET S=3
350 RETURN
355 LET S=2
360 RETURN
```

Look at the subroutine. Suppose the computer generates R = 18. What is S?

So much for the subroutine which generates a sum. Now for the main program which rolls the two dice 3600 times, and tallies the frequency with which each sum occurs. A FOR-NEXT loop will be set up to limit the number of rolls to 3600:

```
FOR C = 1 TO 3600

NEXT C
```

Next, to reach the subroutine, insert GO SUB 200 after the FOR statement.

```
FOR C = 1 TO 3600
GO SUB 200
NEXT C
```

When the computer completes the subroutine and returns to the main program with a value for S, we must have ready a tally list using the subscripted variable F(S). It looks like this:

F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)	F(9)	F(10)	F(11)	F(12)
0	0	0	0	0	0	0	0	0	0	0	0

To set this up, you must insert in the main program a DIM statement followed by a three-step initializing loop. To add 1 to the box which is recording the frequency of a sum, S, we insert after GO SUB 200, LET F(S) = F(S)+1.

```
FOR C = 1 TO 3600
GO SUB 200
LET F(S) = F(S)+1
NEXT C
```

Let's play computer. On the first roll, C = 1, the computer goes to the subroutine and gets the value of S, say 7. On returning to the main program, it goes to the box F(7) and adds 1 to the contents. Since this is the first roll, the frequency of a 7 is now 1.

Here is the complete program with the RANDOM and DIM statements in lines 1 and 2. The initializing loop is in lines 3, 4, and 5. When the C loop is complete, and 3600 sums have been generated and tallied, the results must be published. This is done in lines 50 to 80.

TWO DICE/3600/THEORET ***PROGRAM***

```
 1 RANDOM
 2 DIM F(12)
 3 FOR I=1 TO 12
 4 LET F(I)=0
 5 NEXT I
10 FOR C=1 TO 3600
20 GO SUB 200
30 LET F(S)=F(S)+1
40 NEXT C
50 PRINT "SUM OF DICE","FREQUENCY"
60 FOR S=2 TO 12
70 PRINT S,F(S)
80 NEXT S
90 STOP
200 LET R=INT(36*RND(X))+1
205 IF R<=1 THEN 355
210 IF R<=3 THEN 345
215 IF R<=6 THEN 335
220 IF R<=10 THEN 325
225 IF R<=15 THEN 315
230 IF R<=21 THEN 305
235 IF R<=26 THEN 295
240 IF R<=30 THEN 285
245 IF R<=33 THEN 275
250 IF R<=35 THEN 265
255 LET S=12
260 RETURN
265 LET S=11
270 RETURN
275 LET S=10
280 RETURN
285 LET S=9
290 RETURN
295 LET S=8
300 RETURN
305 LET S=7
310 RETURN
315 LET S=6
320 RETURN
325 LET S=5
330 RETURN
335 LET S=4
340 RETURN
345 LET S=3
350 RETURN
355 LET S=2
360 RETURN
999 END
```

INITIALIZE TALLY LIST

3600 ROLLS

← TALLY STATEMENT

PRINT RESULTS

SUBROUTINE

Here is a typical run of the program.

```
SUM OF DICE    FREQUENCY  OUTPUT
 2              88
 3              205
 4              284
 5              389
 6              489
 7              594
 8              479
 9              416
 10             320
 11             227
 12             109
```

Look at the numbers in the frequency column. If you round off each number to the nearest hundred, the pattern looks like this:

100
200
300
400
500
600
500
400
300
200
100

This sequence recalls the pattern of theoretical frequencies for the sums 2 to 12: 1,2,3,4,5,6,5,4,3,2,1. Now you can understand my reason for selecting 3600 as the number of rolls. By selecting a number which is a multiple of 36, i.e. 3600, each number in the pattern of obtained frequencies will be close to 100 times the corresponding number in the pattern of theoretical frequencies. Suppose you selected 1800 as the number of rolls? Since 1800 is 36 times 50, the expected pattern is 50, 100, 150, 200, 250, 300, 250, 200, 150, 100, 50.

Now for the second program to simulate the sum of two dice when they are rolled 3600 times. In this program, we will "roll" each die separately, add the two outcomes, add 1 to the counter which is counting that sum, and perform the experiment again. We will use the theoretical probabilities of the outcomes 1 to 6, rather than the theoretical probabilities of the sums, 2 to 12. A subscripted variable, F(S), is used again to set up a tally list.

Here is the program.

TWO DICE/3600/EXPER ***PROGRAM***

```
 1 RANDOM
 2 DIM F(12)
 3 FOR I=1 TO 12
 4 LET F(I)=0
 5 NEXT I
10 FOR C=1 TO 3600
20 LET D1=INT(6*RND(X))+1
30 LET D2=INT(6*RND(X))+1
40 LET S=D1+D2
50 LET F(S)=F(S)+1
60 NEXT C
70 PRINT "SUM OF DICE","FREQUENCY"
80 FOR S=2 TO 12
90 PRINT S,F(S)
100 NEXT S
110 END
```

INITIALIZE TALLY LIST

3600 ROLLS

TALLY STATEMENT

PRINT RESULTS

Here is a typical run.

OUTPUT

```
SUM OF DICE    FREQUENCY
 2              86
 3              205
 4              282
 5              408
 6              495
 7              640
 8              474
 9              389
 10             315
 11             203
 12             103
```

Look at the frequency column. If you round off each number to the nearest hundred you obtain the pattern

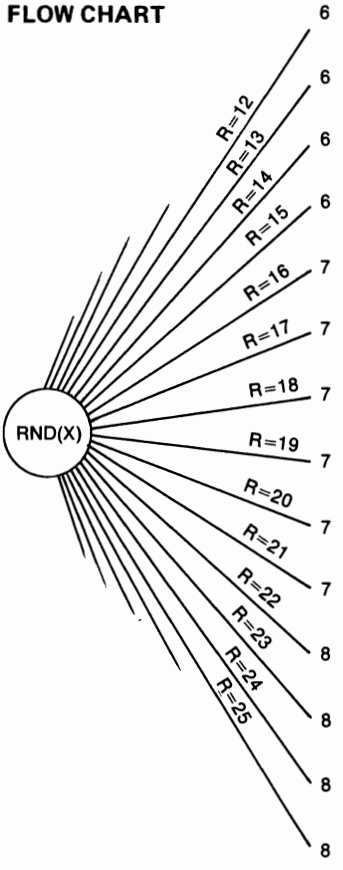
100
200
300
400
500
600
500
400
300
200
100

The pattern of obtained frequencies approximates the pattern of theoretical frequencies.

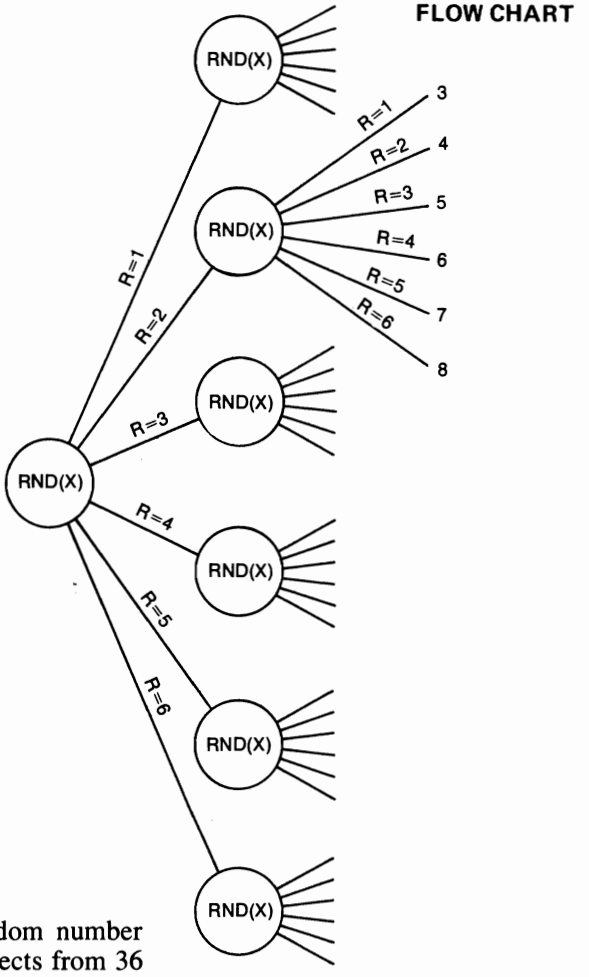
So much for teaching the computer to "roll" two dice. We've done this two ways. The first program, called TWO DICE/3600/THEORET, simulated the sum of two dice by using the random number generator to produce one of the whole numbers from 1 to 36. Each whole number was then associated with one of the

eleven sums, 2 to 12. The second program, called TWO DICE/3600/EXPER, simulated the sum of two dice by using the random number generator twice to produce a whole number from 1 to 6. The two whole numbers generated were then added to produce one of the sums 2 to 12. Here is a schematic representation of the two simulation programs.

TWO DICE/3600/THEORET
FLOW CHART



TWO DICE/3600/EXPER
FLOW CHART



In TWO DICE/3600/THEORET the random number generator, RND(X), is used once, but selects from 36 numbers. In the program TWO DICE/3600/EXPERT the random number generator is used at two stages.

In the next section you will write a program to play a popular game called "craps" in which the player bets on the sum of two dice "rolled" by the computer. In writing your program to play craps, which simulation routine would you use to obtain the sum on each roll? There are two reasons to use the two-stage routine. The routine is shorter, and it gives the player a look at both dice.

EXERCISE SET 6.5

Off-line:

1. Draw several branches of a tree diagram to indicate how you could list all the possible outcomes when you roll three dice.
2. Use your tree diagram to:

A. Determine the number of outcomes.
B. Determine the theoretical probability of each outcome.
C. Determine the range of numbers for the sum of the three dice.

On-line:

3. Write a program to:

A. List all possible outcomes when you roll three dice.
B. Find the sum of each possible outcome.
C. Set up a tally list to count the numbers of times each sum occurs when all possible outcomes have been generated once.
D. Print out a frequency table.

For a hint on how to proceed, look at the program TWO DICE/TABLE. To analyze the frequency pattern, list the increments between successive frequencies.

4. Now write a program which uses the random number generator to roll three dice many times, and lists the frequency of the different sums obtained. Then answer the following questions.

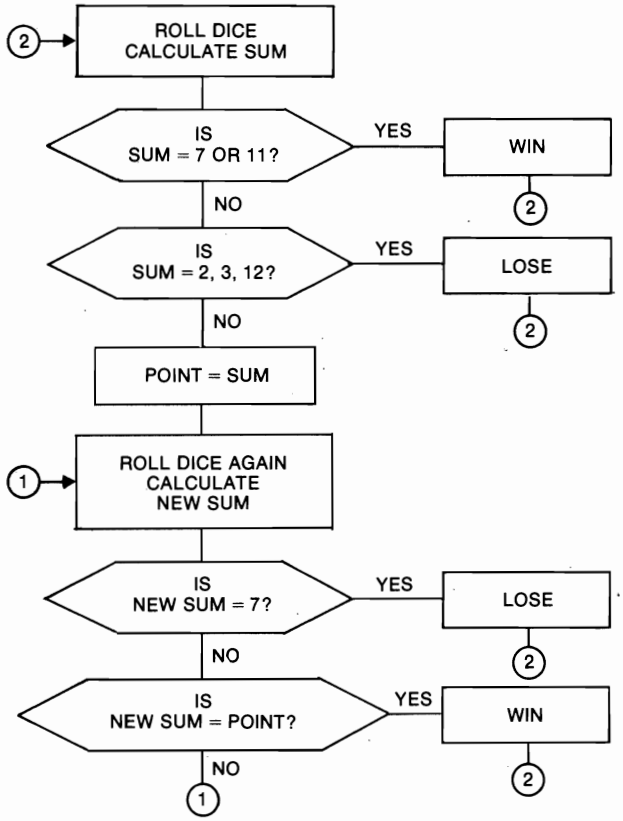
A. Explain why you chose to roll the three dice as many times as you did.
B. Compare the pattern of numbers in the frequency column of your output with the theoretical pattern produced by the program you wrote in exercise 3.

6.6 CRAPS

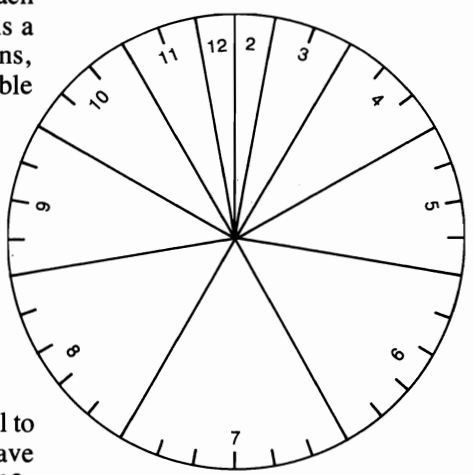
Now you're ready to play a well known two-dice game, Craps. Craps is played as follows. You roll two dice and calculate the sum. If the sum is 7 or 11, you win; if it is 2, 3, or 12, you lose. If the sum is none of these numbers, it becomes your "point," and you must roll the dice again. The object of your second roll is to obtain a sum that matches your point, but if you roll a 7 in the process of trying to match your point, you lose.[1] In a flow chart here is the sequence of play:

1. For a detailed description of Craps, as well as many other games computers can play, I recommend *Game Playing with Computers* by Donald D. Spencer, published by Hayden.

CRAPS/ELEMENTARY FLOW CHART



The game of craps owes much of its popularity to the fact that it is traditionally played with our handy, portable random number generators, dice. Few people recognize that the same game can be played by using the more cumbersome Big Wheel. The Big Wheel has the advantage, however, of showing the probability of each possible outcome when you throw two dice. Here is a picture of a Big Wheel divided into eleven sections, each section representing one of the eleven possible sums obtained when two dice are rolled.



The area of each pie-shaped section is proportional to the probability with which that sum occurs. If you have forgotten the probability with which each sum, 2 to 12, occurs, here is a copy of the table our computer generated in the last section.

```
TWO DICE/TABLE     OUTPUT  SUM          FREQUENCY      PROBABILITY
                           2              1             .02777778
                           3              2             .05555556
                           4              3             .08333333
                           5              4             .1111111
                           6              5             .1388889
                           7              6             .1666667
                           8              5             .1388889
                           9              4             .1111111
                           10             3             .08333333
                           11             2             .05555556
                           12             1             .02777778
```

Look at the table. The sum of 4 occurs with probability 3/36. So, in the wheel, the area of the sector labelled 4 is 3/36 of the area of the circle. In other words, the Craps Wheel is the same wheel we designed in the last section. We have simply erased the spokes, except for those which separate the sections representing different sums. When you look at the geometry of the Craps Wheel, can you see reflected there the frequency pattern printed out by TWO DICE/TABLE?

To play Craps with the Craps Wheel, you spin the wheel to generate the outcome which would be obtained if you were to roll the dice. What is the probability of winning on the first spin? According to the rules you win on a 7 or 11. The probability of a 7 is 6/36, and of an 11, 2/36. Therefore the probability of winning on the first "roll" is 6/36 + 2/36, or 8/36. That's just a little less than 9/36, or one out of four. What is the probability of *losing* on the first spin? You lose if the wheel comes to rest at 2, 3, or 12. The probability of a 2 is 1/36, of a 3 is 2/36, and of a 12 is 1/36. The total is 4/36 or 1/9. Your chances of losing on the first "roll" is one out of nine.

That sounds pretty good, 8 out of 36 to win, and 4 out of 36 to lose. But that's only 12 of the 36 possible outcomes of the first roll. What about the other 24 outcomes? What are your chances of winning on the second roll? Or the third roll?

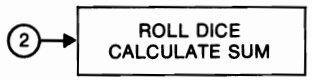
As always, there are two ways to find out. One is to compute the theoretical probabilities of the many possible outcomes.[1] The second way is to use our fast and tireless computer to play Craps, say, a thousand times. But really what you want to know is whether or not you have at least a 50/50 chance of winning at Craps if you play it repeatedly. In short, is Craps a fair game?

To find out if Craps is a fair game, let's first program the computer to play Craps. As soon as we check out the program, we'll modify it to play 1000 games. We'll be

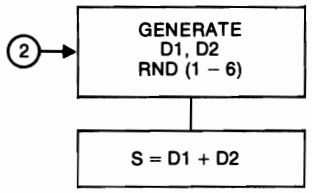
1. See Dorn, Greenberg and Keller, *Mathematical Logic and Probability with BASIC Programming* published by Prindle, Weber, and Schmidt, 1973.

able to decide if the game is fair by comparing the number of games won with the number of games lost.

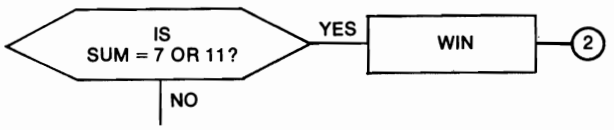
Look again at the Craps flow chart. Replacing some of the words with variable names, let's redraw this flow chart. This will bring us a step closer to BASIC. Separate the box



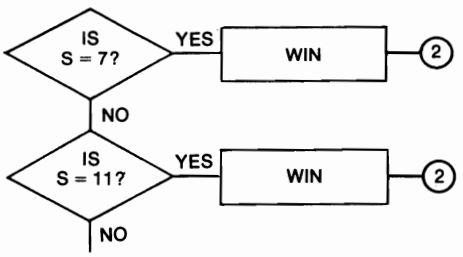
into two boxes, one to roll the dice and another to calculate the sum. To roll the dice, the computer must generate two random numbers between 1 and 6. Let D1 and D2 represent these numbers. Then the sum, S, must be calculated. Here are the two replacement boxes.



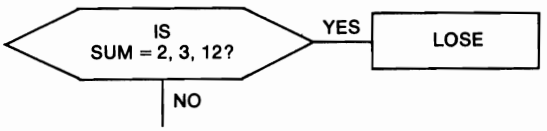
Now separate the diamond



into two diamonds: one which asks IS S = 7? and one which asks IS S = 11?

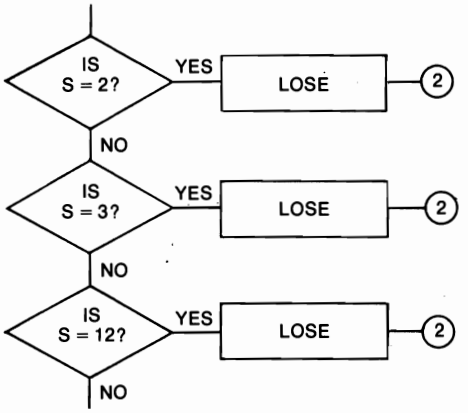


Separate the next diamond

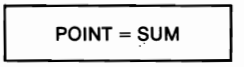


into three diamonds: one which asks IS S = 2?, one

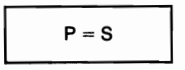
which asks IS S= 3?, and one which asks IS S = 12?.



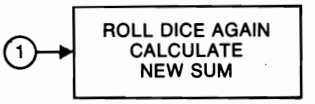
The next box



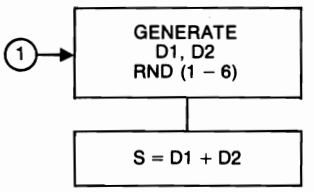
is drawn as:



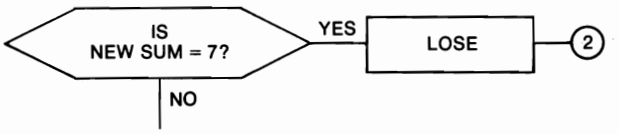
where P represents the "point."
The box



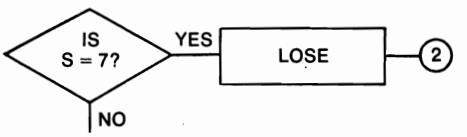
is separated into two boxes.



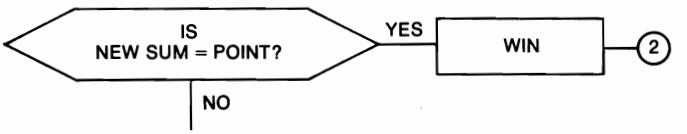
The next diamond



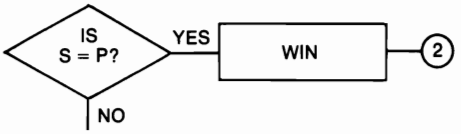
is recoded as:



The last diamond

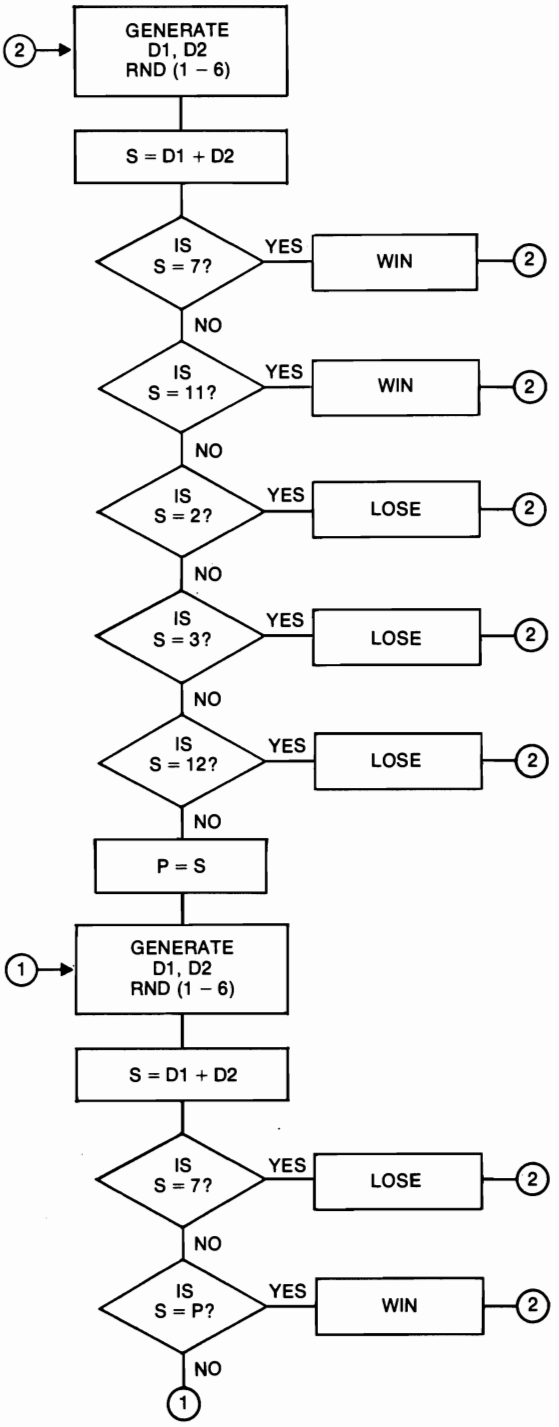


is recoded as:



Here is the new flow chart for Craps.

CRAPS
FLOW CHART



I'm going to let you write the program from this flow chart with just a little help. Look at the flow chart and fill in the following blanks and missing statements. I'll give you some hints along the way.

```
1   ____________________        Whenever you use RND(X) you must
                                write _____.

10 LET D1 = _____               Generate Random Numbers 1-6
                                using RND(X).
20 LET D2 = _____
30 PRINT "YOU ROLLED" D1 "AND" D2
40 ____________________         Calculate sum.
50 PRINT "THE SUM IS"S
60 IF S = 7 THEN ____
70 IF S = 11 THEN ___
80 IF S = 2 THEN ____           Fill in blanks with line numbers
                                of YOU WIN or YOU LOSE
90 IF S = 3 THEN ____
100 IF S = 12 THEN ___
110 ____________________        Save sum as point.
120 PRINT "YOUR POINT IS" P
130 ____________________        Roll dice again.
140 ____________________
150 PRINT "YOU ROLLED" D1 "AND" D2
160 ____________________        Calculate sum.
170 IF S = 7 THEN ____          Fill in blanks with line numbers
180 IF S = P THEN ____          of YOU WIN OR YOU LOSE
190 GO TO 130
200 PRINT "YOU WIN"
210 ____________________        Separate games with blank line.
220 GO TO 10
230 PRINT "YOU LOSE"
240 ____________________        Separate games with blank line.
250 GO TO 10
260 END
```

Your program should look like this.

PROGRAM **CRAPS**

```
  1 RANDOM
 10 LET D1=INT(6*RND(X))+1
 20 LET D2=INT(6*RND(X))+1
 30 PRINT "YOU ROLLED" D1 "AND" D2
 40 LET S=D1+D2
 50 PRINT "THE SUM IS" S
 60 IF S=7 THEN 200
 70 IF S=11 THEN 200
 80 IF S=2 THEN 230
 90 IF S=3 THEN 230
100 IF S=12 THEN 230
110 LET P=S
120 PRINT "YOUR POINT IS" P
130 LET D1=INT(6*RND(X))+1
140 LET D2=INT(6*RND(X))+1
150 PRINT "YOU ROLLED" D1 "AND" D2
160 LET S=D1+D2
170 IF S=7 THEN 230
180 IF S=P THEN 200
190 GO TO 130
200 PRINT "YOU WIN."
210 PRINT
220 GO TO 10
230 PRINT "YOU LOSE."
240 PRINT
250 GO TO 10
260 END
```

Enter your completed program into the computer and give it a test run. Here is a test run of my program.

GAME 1

```
YOU ROLLED 2 AND 3
THE SUM IS 5
YOUR POINT IS 5
YOU ROLLED 5 AND 1
YOU ROLLED 5 AND 4
YOU ROLLED 6 AND 5
YOU ROLLED 3 AND 6
YOU ROLLED 1 AND 6
YOU LOSE.
```

GAME 2

```
YOU ROLLED 1 AND 5
THE SUM IS 6
YOUR POINT IS 6
YOU ROLLED 4 AND 1
YOU ROLLED 3 AND 3
YOU WIN.
```

GAME 3

```
YOU ROLLED 2 AND 6
THE SUM IS 8
YOUR POINT IS 8
YOU ROLLED 5 AND 6
YOU ROLLED 1 AND 4
YOU ROLLED 4 AND 6
YOU ROLLED 1 AND 1
YOU ROLLED 5 AND 4
YOU ROLLED 2 AND 1
YOU ROLLED 6 AND 5
YOU ROLLED 2 AND 3
YOU ROLLED 3 AND 5
YOU WIN.

YOU ROLLED 4
↑C
```

Convinced that your program operates correctly, modify it to have the computer play craps 1000 times.

We want the computer to play 1000 games as quickly as possible. Information about individual outcomes can be eliminated from our program, since we are interested only in the total number of wins. Insert a FOR statement and a NEXT statement to direct the computer to play 1000 games. In place of 200 PRINT "YOU WIN." put a counter, W, which counts the number of wins. Delete 230 PRINT "YOU LOSE." and change all references to line number 230 in the IF-THEN statements to the line number of the NEXT statement. There is no need to count the number of games lost. Below is a copy of my Craps program with all the changes to be made indicated at the right.

CRAPS

PROGRAM

```
  1 RANDOM                          <- INSERT: 4 LET W = 0
                                               5 FOR C = 1 TO 1000
 10 LET D1=INT(6*RND(X))+1
 20 LET D2=INT(6*RND(X))+1
 30 PRINT "YOU ROLLED" D1 "AND" D2  <- DELETE
 40 LET S=D1+D2
 50 PRINT "THE SUM IS" S            <- DELETE
 60 IF S=7 THEN 200
 70 IF S=11 THEN 200
 80 IF S=2 THEN 230                 <- CHANGE 230 TO 250
 90 IF S=3 THEN 230                 <-
100 IF S=12 THEN 230                <-
110 LET P=S
120 PRINT "YOUR POINT IS" P         <- DELETE
130 LET D1=INT(6*RND(X))+1
140 LET D2=INT(6*RND(X))+1
150 PRINT "YOU ROLLED" D1 "AND" D2  <- DELETE
160 LET S=D1+D2
170 IF S=7 THEN 230                 <- CHANGE 230 TO 250
180 IF S=P THEN 200
190 GO TO 130
200 PRINT "YOU WIN."                <- REPLACE WITH: 200 LET W =W + 1
210 PRINT                           <- DELETE
220 GO TO 10                        <- REPLACE WITH: 220 GO TO 250
230 PRINT "YOU LOSE."               <- DELETE
240 PRINT                           <- DELETE
250 GO TO 10                        <- REPLACE WITH: 250 NEXT C
260 END                             <- INSERT: 255 PRINT "GAMES WON:" W
```

Here's the program with all the changes.

```
 1 RANDOM                    CRAPS/1000
 4 LET W=0                     OUTPUT
 5 FOR C=1 TO 1000
10 LET D1=INT(6*RND(X))+1
20 LET D2=INT(6*RND(X))+1
30
40 LET S=D1+D2
50
60 IF S=7 THEN 200
70 IF S=11 THEN 200
80 IF S=2 THEN 250
90 IF S=3 THEN 250
100 IF S=12 THEN 250
110 LET P=S
120
130 LET D1=INT(6*RND(X))+1
140 LET D2=INT(6*RND(X))+1
150
160 LET S=D1+D2
170 IF S=7 THEN 250
180 IF S=P THEN 200
190 GO TO 130
200 LET W=W+1
210
220 GO TO 250
230
240
250 NEXT C
255 PRINT "GAMES WON:" W
260 END
```

Here are three typical runs of the program.

```
GAMES WON: 510
GAMES WON: 472
GAMES WON: 496
```

Look at the output. Is Craps a fair game? Theory says the probability of winning is .4929292 . . ., or just slightly less than 1/2. That means that, in the long run, the player will lose slightly more than he wins. In theory, if he starts with $500 and bets one dollar on each of 1000 games, he will have only $492.92 to take home at the end of the day. Craps is not a fair game, but it's pretty close. So close that most people who play the game make the payoff 1 to1.

EXERCISE SET 6.6

Off-line:

1. Craps is an ancient game, played with two dice, and follows the rules as outlined in this section. Invent a new version of Craps, call it Sparc, played with 1, 2 or 3 dice and draw a flow chart of your Sparc game.

On-line:

2. Write an elementary program from your flow chart for Sparc. Run the program and play several games. Is Sparc a fair game?
3. To determine if Sparc is a fair game, set up your program to have the computer play 1000 games.

PLAYING BLACKJACK

```
DO YOU WANT INSTRUCTIONS?NO

YOU HAVE $1000 TO PLAY WITH.

-----THE CARDS ARE BEING SHUFFLED-----

I'M READY TO DEAL.

BET PLEASE?100

YOUR DOWN CARD IS:    10 OF DIAMONDS
YOUR UP CARD IS:      8 OF CLUBS
MY UP CARD IS:        7 OF SPADES
YOUR TOTAL IS 18
HIT?NO

MY DOWN CARD IS:      8 OF HEARTS
MY TOTAL IS 15
MY NEXT CARD IS:      5 OF HEARTS
MY TOTAL IS 20
I STICK.
I WIN. YOU LOSE $100
YOU NOW HAVE $900 TO PLAY WITH.


BET PLEASE?500

YOUR DOWN CARD IS:    8 OF SPADES
YOUR UP CARD IS:      4 OF CLUBS
MY UP CARD IS:        JACK OF DIAMONDS
YOUR TOTAL IS 12
HIT?YES
YOUR NEXT CARD IS:    3 OF HEARTS
YOUR TOTAL IS 15
HIT?YES
YOUR NEXT CARD IS:    5 OF CLUBS
YOUR TOTAL IS 20
HIT?NO

MY DOWN CARD IS:      2 OF HEARTS
MY TOTAL IS 12
MY NEXT CARD IS:      QUEEN OF SPADES
MY TOTAL IS 22
I BUST. YOU WIN $500
YOU NOW HAVE $1400 TO PLAY WITH.


BET PLEASE?9999

GLAD YOU HAD FUN. COME BACK SOON.
```

7

CARD GAMES

As we pointed out in the last chapter, coins and dice are handy little portable random number generators. People have been betting on the outcomes of these devices for centuries. Another portable random number generator that has found much favor with the public is a deck of cards. Again, people have been betting on the outcomes of cards for centuries. How does a deck of cards function as a random number generator?

First of all, there are 52 unique cards in a deck. They are not numbered from 1 to 52, however, but are grouped in four suits of 13 cards each. After the 52 cards are thoroughly shuffled, they are in random order. By picking or dealing a single card off the top of the deck, the dealer generates a single random number between 1 and 52 inclusive. This is analogous to rolling one die, in which case the outcome is a single random number between 1 and 6 inclusive.

Recall that you obtain a more complex outcome by rolling two dice, a combination of two numbers, each between 1 and 6 inclusive. When the two numbers are added together, the outcome is a number between 2 and 12 inclusive. The outcomes from 2 to 12 do not, however, occur with equal likelihood. The likelihood of each outcome from 2 to 12 was presented in a table in the last chapter.

Similarly, you can obtain a more complex outcome by dealing two or more cards at one time, that is a combination of two or more numbers, each between 1 and 52 inclusive. Unlike dice, however, the unique numbers of the dealt cards are not added together, although the face values may be, as in Twenty-one or Blackjack. Because each deck is divided into four suits, there are many possible combinations of face value and suits. Poker is based on some of the best known five-card combinations or "hands." There are 2,598,960 different five-card hands, of which the following have special value in poker:

Hands	Number of Ways
Royal Flush (A, K, Q, J, 10 in the same suit)	4
Straight Flush (five cards in numerical sequence in the same suit)	36
Four-of-a-Kind (four cards of equal face value and one other)	624

Full House (three-of-a-kind and a pair)	3,744
Flush (any five cards in the same suit)	5,108
Straight (five cards in numerical sequence regardless of suit)	10,200
Three-of-a-Kind (three cards of equal face value and two others)	54,912
Two pairs (two pairs and one other)	123,552
One Pair (two cards of equal face value and three others)	1,098,240

The above combinations are listed in order of decreasing rank in the game of poker. The remaining 1,302,540 hands are ranked by the highest card in the hand.

In the balance of this chapter, we will teach the computer to deal cards, and then use the card-dealing routine in three card games, the last of which is Blackjack or Twenty-one.

7.1 TEACHING THE COMPUTER TO DEAL CARDS

An ordinary deck of cards consists of four suits called spades, hearts, diamonds, and clubs. Each suit contains one each of the following cards: ace, king, queen, jack, 10, 9, 8, 7, 6, 5, 4, 3, 2. As the computer deals the 52 cards one at a time, the output will look something like this:

```
QUEEN OF SPADES
3 OF DIAMONDS
10 OF CLUBS
JACK OF HEARTS
8 OF HEARTS
  .
  .
  .
```

To teach the computer to deal one of the 52 cards, first have it generate a random whole number between 1 and 52 inclusive. This is done by using the random number generator RND(X). First multiply RND(X) by 52 to generate mixed numbers between 1 and 52. Thus, 52*RND(X). Next use INT to transform the mixed numbers into whole numbers. Thus, INT(52* RND(X)). Finally, add 1 to shift the whole numbers 0 to 51 one unit to the right. This gives INT(52* RND(X)) + 1.

Now that we have an expression to generate numbers between 1 and 52 inclusive, we need a scheme to determine which card each number represents. For example, suppose the number 23 is generated. Which card in the deck does 23 represent?

There are many schemes which can be used for pairing each number from 1 to 52 with a card in the deck.

Writing a routine to associate a card with a number is easy if the assignment scheme follows a pattern. Let's follow the hierarchy of suits used in bridge. From lowest to highest, the suits are ranked clubs, diamonds, hearts, and spades. Let the numbers 1 to 13 represent clubs, 14 to 26 represent diamonds, 27 to 39 represent hearts, and 40 to 52 represent spades. Within each suit, the rank of cards will be 2 (lowest), 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace (highest). The relationship between number and card is given in the following table:

NUMBER	CARD		
1	2 of clubs	27	2 of hearts
2	3 of clubs	28	3 of hearts
3	4 of clubs	29	4 of hearts
4	5 of clubs	30	5 of hearts
5	6 of clubs	31	6 of hearts
6	7 of clubs	32	7 of hearts
7	8 of clubs	33	8 of hearts
8	9 of clubs	34	9 of hearts
9	10 of clubs	35	10 of hearts
10	Jack of clubs	36	Jack of hearts
11	Queen of clubs	37	Queen of hearts
12	King of clubs	38	King of hearts
13	Ace of clubs	39	Ace of hearts
14	2 of diamonds	40	2 of spades
15	3 of diamonds	41	3 of spades
16	4 of diamonds	42	4 of spades
17	5 of diamonds	43	5 of spades
18	6 of diamonds	44	6 of spades
19	7 of diamonds	45	7 of spades
20	8 of diamonds	46	8 of spades
21	9 of diamonds	47	9 of spades
22	10 of diamonds	48	10 of spades
23	Jack of diamonds	49	Jack of spades
24	Queen of diamonds	50	Queen of spades
25	King of diamonds	51	King of spades
26	Ace of diamonds	52	Ace of spades

Now that we have settled on a scheme for assigning a number to each card, let's write a program to "deal" all 52 cards in random order, as they might occur in a well-shuffled deck. We start by using FOR and NEXT to build a loop. Since there are 52 cards in a deck, the FOR and NEXT statements will be set up this way:

```
FOR C = 1 TO 52
NEXT C
```

Inside the loop, at line 20, we insert the expression to

generate a random whole number between 1 and 52. Don't forget the RANDOM statement.

```
 1 RANDOM
10 FOR C = 1 TO 52
20 LET R = INT(52*RND(X)) + 1
30 NEXT C
```

When a real player deals from a real deck, he does not replace each card in the deck after it is dealt. He thereby eliminates the possibility of duplicate outcomes. How do we teach the computer to do the same thing?

When we taught the computer to play COMPUTER GUESS in Chapter 3, we taught it to avoid duplicate guesses by setting up a check list in the computer's memory. Let's do the same thing here. Instead of the ten-box check list required for COMPUTER GUESS, we need a 52-box check list, one box for each of the 52 cards. Each box will be given an initial value of zero. The computer's check list looks like:

N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	N(11)	N(12)	N(13)

N(14)	N(15)	N(16)	N(17)	N(18)	N(19)	N(20)	N(21)	N(22)	N(23)	N(24)	N(25)	N(26)

N(27)	N(28)	N(29)	N(30)	N(31)	N(32)	N(33)	N(34)	N(35)	N(36)	N(37)	N(38)	N(39)

N(40)	N(41)	N(42)	N(43)	N(44)	N(45)	N(46)	N(47)	N(48)	N(49)	N(50)	N(51)	N(52)

To instruct the computer to set aside space in memory for these 52 boxes, we need a DIMENSION or DIM statement. We also need an initialization loop to put a zero in each box. Add lines 2, 3, 4, and 5 to the above four program steps.

```
 1 RANDOM
 2 DIM N(52)
 3 FOR I = 1 TO 52
 4 LET N(I) = 0
 5 NEXT I
10 FOR C = 1 TO 52
20 LET R = INT(52*RND(X)) + 1
30 NEXT C
```

Here's how the check list works. Each time a card is dealt, the zero in the corresponding box is replaced by the number 1. In short, the number 1 is a flag. When the computer generates a number, R, instruct it to look at

the contents of box N(R). If there's a 1 in the box N(R), the number R has already been "dealt." The computer must generate another number. If N(R) is not 1, then deal the card, after putting a 1 in the box. Add lines 21 and 22 to the steps above.

```
 1 RANDOM
 2 DIM N(52)
 3 FOR I = 1 TO 52
 4 LET N(I) = 0
 5 NEXT I
10 FOR C = 1 TO 52
20 LET R = INT(52*RND(X)) + 1
21 IF N(R) = 1 THEN 20
22 LET N(R) = 1
30 NEXT C
```

So much for preventing the same card being dealt twice. But what card does the number represent? The routine which assigns a card to the number requires quite a few steps and will be set up as a subroutine. Insert a GO SUB at line 23, leaving a blank after the GO SUB. Here is our partial program:

```
 1 RANDOM
 2 DIM N(52)
 3 FOR I = 1 TO 52
 4 LET N(I) = 0
 5 NEXT I
10 FOR C = 1 TO 52
20 LET R = INT (52*RND(X)) + 1
21 IF N(R) = 1 THEN 20
22 LET N(R) = 1
23 GO SUB _____
30 NEXT C
```

The organization of the program as given above is natural if your only purpose is to deal 52 cards. When the cards are dealt as part of a game, however, it is desirable to organize the program differently, treating the card dealing routine as a subroutine activated by a main game program.

In a real card game there are two steps that mark the start of the game: (1) gather the cards together in a complete deck and (2) shuffle the deck. Initializing the boxes in the check list parallels the first step of gathering the cards together. In the computer, the cards are shuffled by the random number generator. Let me place the initialization of the check list, lines 3, 4, and 5, in a subroutine beginning at line 500. The subroutine is activated by GO SUB 500 in line 5. Also, a STOP statement has been inserted at line 40 and an END statement at line 999. Our reorganized partial program now looks like this.

```
 1 RANDOM
 2 DIM N(52)
 5 GO SUB 500
10 FOR C = 1 TO 52
20 LET R = INT(52*RND(X)) + 1
21 IF N(R) = 1 THEN 20
22 LET N(R) = 1
23 GO SUB _____
30 NEXT C
40 STOP

500  FOR I = 1 TO 52
505  LET N(I) = 0
510  NEXT I
515  RETURN

999  END
```

Now let's build a subroutine which deals, not 52 cards, but just one card. Dealing one card consists of generating a number between 1 and 52, checking for duplicates, and then assigning a card to the number. These three operations are done in lines 20, 21, 22 and 23. I'll move these lines into a new subroutine beginning at line 600. The partial program now takes this revised form.

```
 1 RANDOM
 2 DIM N(52)
 5 GO SUB 500
10 FOR C = 1 TO 52
20 GO SUB 600
30 NEXT C
40 STOP

500 FOR I =1 TO 52
505 LET N(I) = 0
510 NEXT I
515 RETURN

600 LET R = INT(52*RND(X)) + 1
605 IF N(R) = 1 THEN 600
610 LET N(R) = 1
615 GO SUB 700
620 RETURN

999 END
```

Look at line 615. I've put 700 in the blank after the GO SUB. We're now ready to write the routine which assigns a card to each number, according to the following table.

CARD FACE

SUIT		2	3	4	5	6	7	8	9	10	J	Q	K	A
S	CLUB	1	2	3	4	5	6	7	8	9	10	11	12	13
U	DIAMOND	14	15	16	17	18	19	20	21	22	23	24	25	26
I	HEART	27	28	29	30	31	32	33	34	35	36	37	38	39
T	SPADE	40	41	42	43	44	45	46	47	48	49	50	51	52

Look at the table. Two steps are required to determine the card assigned to the number 10. First, read the corresponding row heading at the left edge of the table; the suit is clubs. Next, read the corresponding column heading at the top of the table; the card face is a jack. In short, a 10 is the jack of clubs. The same procedure will reveal that a 26 is the ace of diamonds, a 28 is the 3 of hearts, and a 47 is the 9 of spades.

So much for the human procedure for determining the card assigned to a number. What about a procedure for the computer? Look again at the table. Any number from 1 to 13 represents a club; any number from 14 to 26 represents a diamond; any number from 27 to 39 represents a heart; and any number from 40 to 52 represents a spade. These results are summarized below.

RANGE OF NUMBERS	SUIT
1 - 13	CLUBS
14 - 26	DIAMONDS
27 - 39	HEARTS
40 - 52	SPADES

Thus, for the computer to determine suit, it will simply use a sequence of IF-THEN statements to determine the range into which a given value of R falls. Now for the second operation, the procedure for determining the card face.

Look at the table again. In the first row all the numbers are between 1 and 13 inclusive. The correspondence between these numbers and the card face is given in the column heading above each number. The following list shows this association of number and card face.

NUMBER	CARD FACE
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	Jack
11	Queen
12	King
13	Ace

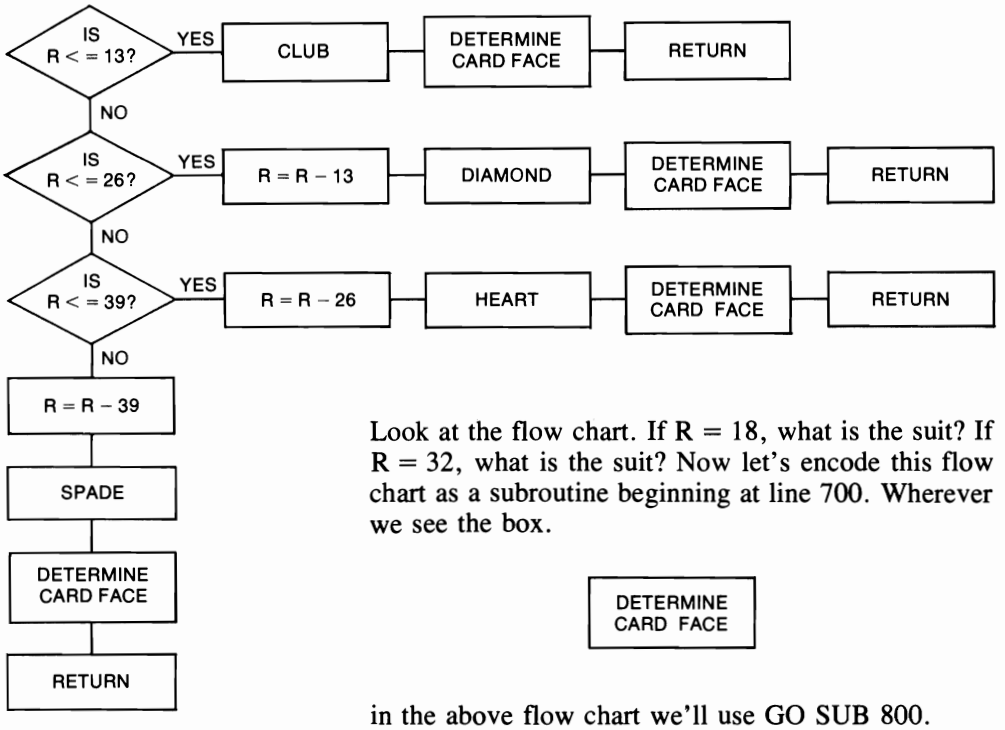
Note that the numbers 1 to 9 correspond to the card faces 2 to 10 and that the numbers 10 to 13 correspond to the jack, queen, king, and ace.

Now look at the second row in the table. All numbers are between 14 and 26 and designate diamonds. Subtract 13 from each number. The range of numbers is now 1 to 13. To determine the card face use the list given above.

Now look at the third row. All the numbers are between 27 and 39 and designate hearts. Subtract 26 from each number. The range of numbers is now 1 to 13. To determine the card face use the list given above.

Now look at the fourth row. All the numbers are between 40 and 52 and designate spades. Subtract 39 from each number. The range of numbers is now 1 to 13. To determine the card face use the list given above. Thus, after the suit has been determined, the computer can determine the card face by either subtracting 13, 26, or 39 and then using a sequence of IF-THEN statements to determine the card face according to the list given above. In short, the computer must perform two operations. The first operation is to determine suit and then subtract either 13, 26, or 29. Once the first operation has been completed, the second operation must be activated, that of determining the card face. Here is a flow chart of the first operation.

DETERMINE SUIT FLOW CHART



Look at the flow chart. If R = 18, what is the suit? If R = 32, what is the suit? Now let's encode this flow chart as a subroutine beginning at line 700. Wherever we see the box.

in the above flow chart we'll use GO SUB 800.

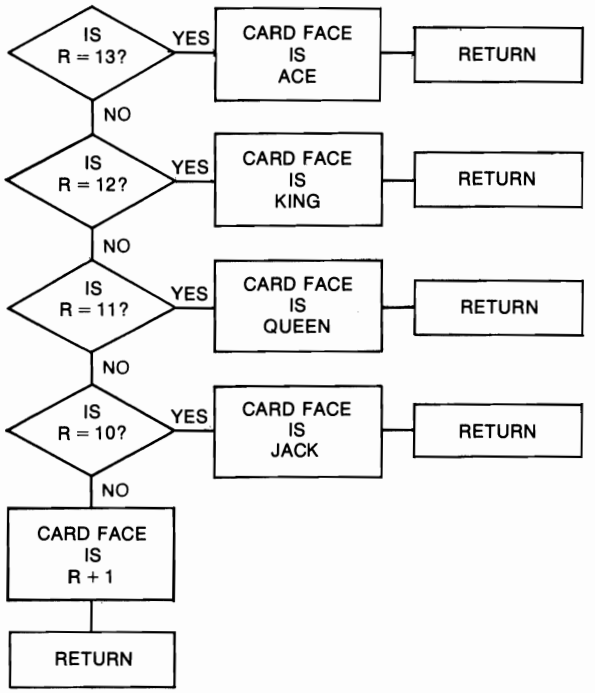
DETERMINE SUIT
SUBROUTINE

```
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 PRINT "SPADE ";
725 GO SUB 800
730 RETURN
735 LET R=R-26
740 PRINT "HEART ";
745 GO SUB 800
750 RETURN
755 LET R=R-13
760 PRINT "DIAMOND ";
765 GO SUB 800
770 RETURN
775 PRINT "CLUB ";
780 GO SUB 800
785 RETURN
```

Look at lines 720, 740, 760, and 775. The semicolon indicates there is more to come on each line.

So much for the first subroutine which designates suit. Now to activate the second operation, that of determining the card face. Here is a flow chart of that operation.

DETERMINE CARD FACE
FLOW CHART



If R = 8, what is the card face? If R = 1, what is the card face? Encode the above flow chart as a subroutine beginning at line 800.

DETERMINE CARD FACE
SUBROUTINE

```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1
825 RETURN
830 PRINT "JACK"
835 RETURN
840 PRINT "QUEEN"
845 RETURN
850 PRINT "KING"
855 RETURN
860 PRINT "ACE"
865 RETURN
```

The card face subroutine completes the two-part operation of associating a number with a card. Let's put the main program and all the subroutines together.

52 CARDS ***PROGRAM***

```
 1 RANDOM
 2 DIM N(52)
10 GO SUB 500 <------ ACTIVATE INITIALIZE CHECK LIST
20 FOR C=1 TO 52
30 GO SUB 600 <------ ACTIVATE GENERATE NUMBER
40 NEXT C
50 STOP
500 FOR I=1 TO 52
505 LET N(I)=0
510 NEXT I
515 RETURN
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700 <------ ACTIVATE DETERMINE SUIT
620 RETURN
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 PRINT "SPADE ";
725 GO SUB 800 <------ ACTIVATE DETERMINE CARD FACE
730 RETURN
735 LET R=R-26
740 PRINT "HEART ";
745 GO SUB 800 <------ ACTIVATE DETERMINE CARD FACE
750 RETURN
755 LET R=R-13
760 PRINT "DIAMOND ";
765 GO SUB 800 <------ ACTIVATE DETERMINE CARD FACE
770 RETURN
775 PRINT "CLUB ";
780 GO SUB 800 <------ ACTIVATE DETERMINE CARD FACE
785 RETURN
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1
825 RETURN
830 PRINT "JACK"
835 RETURN
840 PRINT "QUEEN"
845 RETURN
850 PRINT "KING"
855 RETURN
860 PRINT "ACE"
865 RETURN
999 END
```

MAIN PROGRAM (lines 10–50)

INITIALIZE CHECK LIST (lines 500–515)

GENERATE NUMBER (lines 600–620)

DETERMINE SUIT (lines 700–785)

DETERMINE CARD FACE (lines 800–865)

Here is a typical run.

```
HEART JACK          OUTPUT
HEART  7
DIAMOND  8
DIAMOND  3
CLUB JACK
HEART  10
DIAMOND  5
CLUB  2
DIAMOND KING
HEART  2
CLUB  9
DIAMOND  9
HEART QUEEN
CLUB  8
SPADE QUEEN
CLUB  6
CLUB  3
CLUB QUEEN
DIAMOND  10
SPADE  4
HEART ACE
SPADE  3
DIAMOND QUEEN
CLUB  7
HEART  6
DIAMOND ACE
DIAMOND  2
SPADE  5
DIAMOND  4
HEART KING
CLUB ACE
SPADE JACK
HEART  3
SPADE  6
HEART  8
DIAMOND JACK
CLUB KING
SPADE  8
HEART  5
SPADE ACE
SPADE KING
HEART  9
DIAMOND  7
SPADE  10
CLUB  10
HEART  4
CLUB  5
SPADE  7
CLUB  4
SPADE  9
DIAMOND  6
SPADE  2
```

Look at the output. The suit label comes first, followed by the card face. A more common way of listing cards is to print the card face first followed by the suit label. Most of us say "5 OF SPADES" instead of "SPADE 5." To humanize the output, let's change the

order of printing. Print the card face first and then the suit label. This requires interchanging two steps at four places in the program 52 CARDS. Lines 720 and 725, 740 and 745, 760 and 765, and 775 and 780 must be interchanged. Finally, the semicolon must be placed at the end of each PRINT statement in the DETERMINE CARD FACE subroutine. Here is the program 52 CARDS with these changes.

52 CARDS/FACE FIRST ***PROGRAM***

```
 1 RANDOM
 2 DIM N(52)
10 GO SUB 500                    MAIN PROGRAM
20 FOR C=1 TO 52
30 GO SUB 600
40 NEXT C
50 STOP
500 FOR I=1 TO 52                INITIALIZE CHECK LIST
505 LET N(I)=0
510 NEXT I
515 RETURN
600 LET R=INT(52*RND(X))+1       GENERATE NUMBER
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
700 IF R<=13 THEN 775            DETERMINE SUIT
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800                   INTERCHANGED
725 PRINT " OF SPADES"           INTERCHANGED
730 RETURN
735 LET R=R-26
740 GO SUB 800                   INTERCHANGED
745 PRINT " OF HEARTS"           INTERCHANGED
750 RETURN
755 LET R=R-13
760 GO SUB 800                   INTERCHANGED
765 PRINT " OF DIAMONDS"         INTERCHANGED
770 RETURN
775 GO SUB 800                   INTERCHANGED
780 PRINT " OF CLUBS"            INTERCHANGED
785 RETURN
800 IF R=13 THEN 860             DETERMINE CARD FACE
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
825 RETURN
830 PRINT "JACK";                SEMICOLON
835 RETURN
840 PRINT "QUEEN";               SEMICOLON
845 RETURN
850 PRINT "KING";                SEMICOLON
855 RETURN
860 PRINT "ACE";                 SEMICOLON
865 RETURN
999 END
```

Here is a typical run.

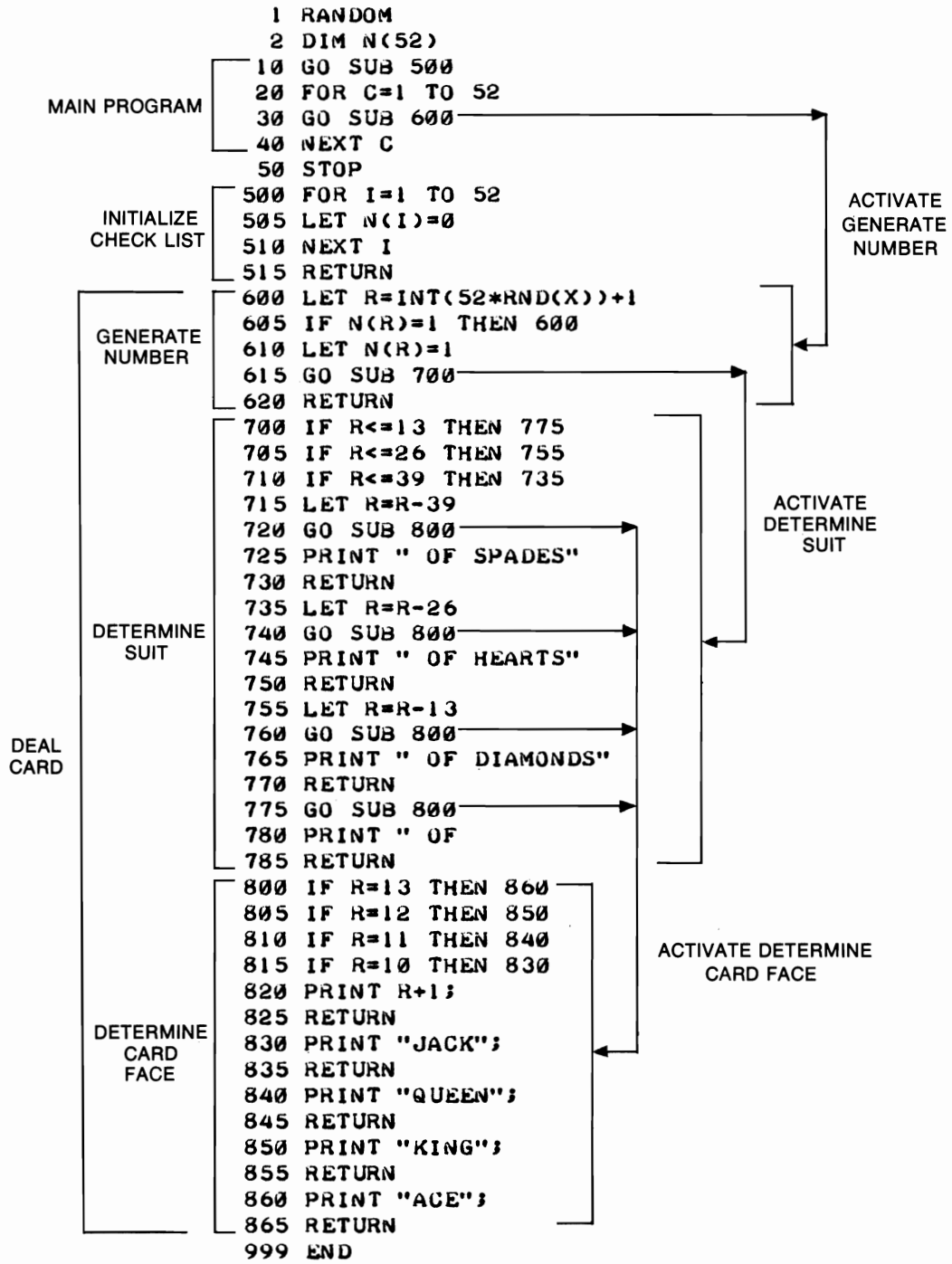
OUTPUT

```
 3  OF DIAMONDS
 10  OF DIAMONDS
 6  OF CLUBS
 9  OF DIAMONDS
 4  OF HEARTS
 3  OF SPADES
 10  OF HEARTS
 5  OF CLUBS
QUEEN OF DIAMONDS
 8  OF HEARTS
 2  OF CLUBS
 7  OF SPADES
 6  OF SPADES
 7  OF CLUBS
 4  OF DIAMONDS
QUEEN OF CLUBS
KING OF SPADES
KING OF DIAMONDS
ACE OF CLUBS
 5  OF SPADES
ACE OF HEARTS
 6  OF DIAMONDS
 9  OF CLUBS
JACK OF CLUBS
JACK OF HEARTS
JACK OF DIAMONDS
 6  OF HEARTS
 4  OF SPADES
 2  OF SPADES
 7  OF DIAMONDS
 9  OF SPADES
 10  OF CLUBS
 7  OF HEARTS
JACK OF SPADES
 8  OF DIAMONDS
KING OF HEARTS
ACE OF DIAMONDS
 5  OF DIAMONDS
ACE OF SPADES
 8  OF SPADES
QUEEN OF SPADES
 10  OF SPADES
 2  OF HEARTS
 5  OF HEARTS
KING OF CLUBS
 2  OF DIAMONDS
 3  OF CLUBS
 9  OF HEARTS
 4  OF CLUBS
 3  OF HEARTS
 8  OF CLUBS
QUEEN OF HEARTS
```

It is most important that you have a clear picture of the organization of the program, 52 CARDS/FACE FIRST, and the interrelationship between the main program and its subroutines. Just as loops can be nested, one subroutine may activate another, which in turn may

activate another. At line 30, the main program activates the GENERATE NUMBER subroutine beginning at line 600, and that in turn activates the DETERMINE SUIT subroutine beginning at line 700, and that in turn activates the DETERMINE CARD FACE subroutine beginning at line 800. Here is a schematic diagram of these nested subroutines.

52 CARDS/FACE FIRST



EXERCISE SET 7.1

On-line:

1. Take the program 52 CARDS/FACE FIRST and fix it up to produce an output similar to the following. Then save lines 500 through 865 for use in the next 3 exercise sets.

```
I CAN DEAL YOU ANY NUMBER OF CARDS. WHEN YOU DON'T
WANT ANYMORE, TYPE 99.

HOW MANY CARDS WOULD YOU LIKE DEALT? 4
 8  OF SPADES
KING OF CLUBS
QUEEN OF CLUBS
JACK OF HEARTS

HOW MANY CARDS WOULD YOU LIKE DEALT? 0
NOT TOO BRIGHT. IF YOU DON'T WANT ANY TYPE 99.

HOW MANY CARDS WOULD YOU LIKE DEALT? 84
STUPID! A DECK HAS ONLY 52 CARDS.

HOW MANY CARDS WOULD YOU LIKE DEALT? 7.9
WISE GUY! WHAT DO YOU WANT ME TO DO? TEAR CARDS
INTO LITTLE PIECES.

HOW MANY CARDS WOULD YOU LIKE DEALT? -6
I'D LIKE TO SEE YOU DEAL THAT NUMBER OF CARDS.

HOW MANY CARDS WOULD YOU LIKE DEALT? 6
 10  OF DIAMONDS
 5 OF DIAMONDS
 3 OF DIAMONDS
 6 OF SPADES
 4 OF SPADES
 10 OF HEARTS

HOW MANY CARDS WOULD YOU LIKE DEALT? 99
HOPE YOU LIKED MY FAST DEAL.
```

7.2 SUM OF FIVE CARDS

Here's a card game in which the computer asks the player to guess into which of four ranges, 6 to 20, 21 to 30, 31 to 40, or 41 to 50, the sum of five cards, to be dealt by the computer, will fall. The value of an ace is 1. The value of a king, queen, or jack is 10. The value of any other card is its face value. The minimum sum of five cards, 6, occurs if the four aces and a 2 are dealt. The maximum sum, 50, occurs if any five cards with value 10 are dealt. The overall range of numbers for the sums is therefore 6 to 50, which is subdivided into four outcomes as follows.

OUTCOME	RANGE OF SUMS
1	6-20
2	21-30
3	31-40
4	41-50

If the sum falls in the range specified by the player, the player wins. Otherwise he loses.

Suppose a player wants to back up his guess with a little wager, say a dollar. How much should he win if his guess is correct? Should he win a dollar? That is, should the payoff be 1 to 1 for all winning bets? Recall that the payoff for a given outcome is related to the probability of that outcome. In BIG WHEEL the player could bet on one of four numbers. The payoff table looked like this:

NUMBER	WAYS TO WIN	WAYS TO LOSE	PAYOFF FACTOR
1	3	6	2
2	3	6	2
3	2	7	3.5
4	1	8	8

The corresponding table for the new game, 5 CARD SUM, looks like this.

RANGE	WAYS TO WIN	WAYS TO LOSE	PAYOFF FACTOR
6-20			
21-30			
31-40			
41-50			

We could easily fill in the payoff table for BIG WHEEL because we knew from the geometry of the wheel that the probability of a 1 was 1/3, of a 2 was 1/3, of a 3 was 2/9 and of a 4 was 1/9. There are two methods by which we can determine the probability of each of the four outcomes in 5 CARD SUM. One method is called the analytical method and the other the experimental method. Recall that both methods were used in Chapter 6 to calculate the probability of sums when two dice are rolled. The analytical method required four steps:

(1) Draw a tree diagram as an aid to listing all possible outcomes, 36 in the case of two dice.
(2) Insert at the end of each branch the sum of the two dice.
(3) Make a table showing the eleven different sums from 2 to 12 and the corresponding frequency of occurrence of each sum.
(4) Calculate the probability of each sum or outcome by dividing the frequency by 36.

The foregoing analytical method gave us the theoretical probabilities.

In the second method, the experimental method, we calculated the probability of each of the sums from 2 to 12 by rolling two dice 3600 times and recording the frequency of each different outcome. Instead of rolling the dice ourselves, however, we taught the computer to do this in the program TWO DICE/3600/EXPER. This

program "rolled" two dice, computed the sum and tallied the result. After 3600 rolls, a frequency table was published along with the probability of occurence of each sum. The probabilities obtained experimentally were not exactly the same as those obtained with the analytical method, but the more times the computer rolled the dice, the closer the experimental values would come to the theoretical values.

Each method has its advantage. The analytical method yields exact values for the probabilities. If it is impossible to specify all the outcomes, however, we must resort to the experimental method.

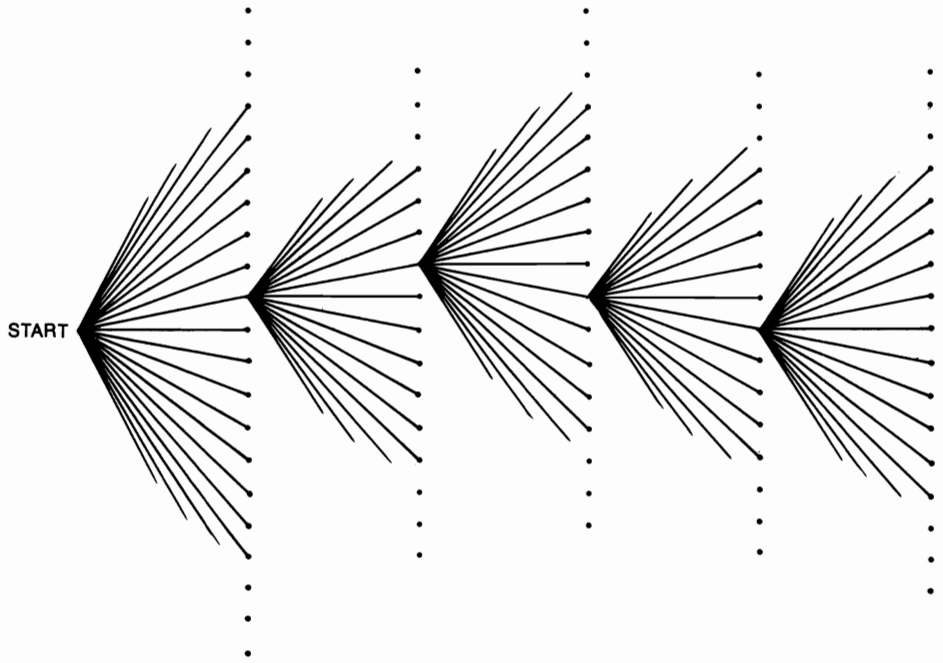
So much for our review of the two methods of determining probabilities, as used in Chapter 6. Now let's use them to determine the probabilities of each of the four outcomes of the game 5 CARD SUM. First, the analytical method. The first step is to draw a tree diagram as an aid in listing all the possible outcomes when five cards are dealt. It's apparent that there will be more branches on the 5 CARD tree than there were on the 2 DICE tree. Here is the way to figure out the number of branches on the 5 CARD tree. The first card dealt can be any one of the 52 cards in the deck. The second card dealt can be any one of the 51 remaining cards. The third card dealt can be any one of the 50 remaining cards. The fourth card dealt can be any one of the 49 remaining cards. The fifth card dealt can be any one of the 48 remaining cards. Thus, the total number of outcomes or branches in the tree diagram is the product of 52*51*50*49*48 or 311,875,200. A large number! However, this number can be reduced by eliminating duplicate outcomes. For example,

this hand	ace of spades 2 of diamonds 9 of hearts 10 of hearts 3 of clubs	has the same sum as
this hand	9 of hearts 2 of diamonds 3 of clubs 10 of hearts ace of spades	even though the

order in which the cards were dealt is different. Altogether, there are 120 possible arrangements of these five cards, all with the same sum. Any one of these five cards could be dealt first. Then, once the first card has been selected, any one of the four remaining cards could be second. Then, once the second card has been selected, any one of the three remaining cards could be third. Then, once the third card has been selected, any one of the two remaining cards could be fourth. Finally,

after the fourth card has been selected, the fifth card can be selected in only one way. If you multiply the number of ways in which each of the cards can be selected, 5*4*3*2*1, you get 120. Thus, each set of five cards can be drawn in 120 different sequences. Each arrangement has the same sum, however. To eliminate the sequence effect divide 311,875,200 by 120. The result, 2,598,960, is the number of distinct five card hands.

Even if you consider only the 2,598,960 distinct five-card combinations, the resulting tree diagram is too much to ask a human to draw. Part of the tree diagram might look something like this, however.



If it were possible to draw the whole tree, you would write the sum of the five cards at the end of each branch. Then you would set up a frequency table like this:

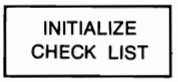
RANGE	FREQUENCY
6-20	
21-30	
31-40	
41-50	

To fill in the frequency table, you would start at the top of the tree diagram and work down, putting a tally mark opposite the range into which each sum falls. Next you would divide the frequency accumulated in each range by the total frequency, 2,598,960.

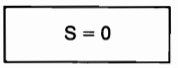
Even though we can easily visualize the analytical procedure, it would take too long to execute using paper and pencil. As we did in computing the probabilities of the outcomes of rolling two dice, we could program the computer to generate the tree.

The alternative to the analytical method is to employ the experimental method, programming the computer to simulate 10,000 deals of five cards each. After the computer completes the 10,000 deals, it prints out the frequency table. From this table we can calculate the probabilities of each of the four outcomes. These probabilities then allow us to construct a payoff table.

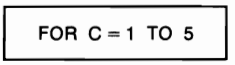
Since either method, analytical or experimental, will take quite a bit of computer time, I'll limit myself to the experimental procedure. We start by teaching the computer to deal five cards and compute the sum. If the program works correctly, we'll modify it to repeat the procedure 10,000 times. The flow chart for the program is drawn as follows. First, the check list for the card dealing routine is initialized.



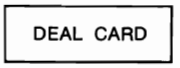
Next, a variable S is used to hold a place for the sum. The value of S at the start of a deal is zero.



The next step is to build a loop using the FOR command. Five cards are to be dealt, so the FOR statement looks like this.



The next step is to deal a card.



This step is encoded by inserting a GO SUB command in the main program which activates the card dealing subroutine developed earlier in this chapter. Here again

is the card dealing subroutine.

DEAL CARD
SUBROUTINE

GENERATE NUMBER

```
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
```

DETERMINE SUIT

```
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800
725 PRINT " OF SPADES"
730 RETURN
735 LET R=R-26
740 GO SUB 800
745 PRINT " OF HEARTS"
750 RETURN
755 LET R=R-13
760 GO SUB 800
765 PRINT " OF DIAMONDS"
770 RETURN
775 GO SUB 800
780 PRINT " OF CLUBS"
785 RETURN
```

DETERMINE CARD FACE

```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
825 RETURN
830 PRINT "JACK";
835 RETURN
840 PRINT "QUEEN";
845 RETURN
850 PRINT "KING";
855 RETURN
860 PRINT "ACE";
865 RETURN
```

When the computer returns from the card-dealing subroutine to the main program, the value of the card dealt must be added to S, the running total. As yet there is no statement in the subroutine that stores in the computer's memory the value of the card dealt. The computer only prints out the name of the card. We must modify the card-dealing subroutine to tell the computer the value of the card. The place to do this is in the DETERMINE CARD FACE section between lines 800 and 865.

DETERMINE CARD FACE
SUBROUTINE

```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
825 RETURN
830 PRINT "JACK";
835 RETURN
840 PRINT "QUEEN";
845 RETURN
850 PRINT "KING";
855 RETURN
860 PRINT "ACE";
865 RETURN
```

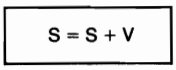
In the game of 5 CARD SUM, an ace is worth one point, a king, queen, or jack ten points, and all other cards are worth their face value. Now for the modifications. Look at line 800. If R, the output of the suit-designating subroutine, is 13, the computer transfers control to line 860 and executes PRINT "ACE";. After executing this statement, the value of an ace, 1, should be stored in memory. This requires a new variable, V, to hold a place for the value of the card dealt. Insert 861 LET V = 1. Now look at lines 805, 810, and 815. If R is 12, 11, or 10, the card dealt is a king, queen or jack. The value of each of these cards is 10. To store a 10 for any of these cards, insert 851 LET V = 10, 841 LET V = 10, and 831 LET V = 10. So much for telling the computer the values of an ace, king, queen, and jack. Now to store the values of cards with face values 2 through 10.

If R is not 13, 12, 11, or 10, the computer drops to 820 PRINT R + 1;. Values of R from 1 through 9 correspond to the face cards 2 through 10. The card face is always one more than the corresponding value of R. After 820 PRINT R + 1;, insert 821 LET V = R + 1. Here is the DETERMINE CARD FACE routine with all these modifications.

DETERMINE CARD FACE/ 5 CARD SUM *SUBROUTINE*

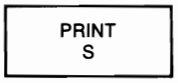
```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
821 LET V=R+1
825 RETURN
830 PRINT "JACK";
831 LET V=10
835 RETURN
840 PRINT "QUEEN";
841 LET V=10
845 RETURN
850 PRINT "KING";
851 LET V=10
855 RETURN
860 PRINT "ACE";
861 LET V=1
865 RETURN
```

On returning from the card dealing subroutine to the main program, we want the computer to add V to the sum. Draw the box:



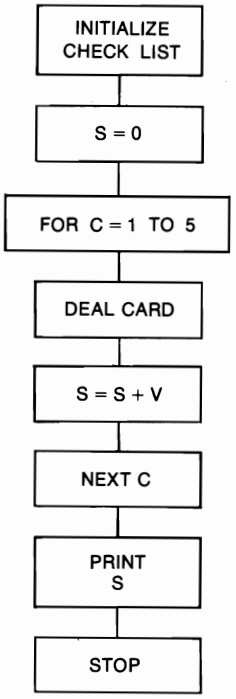
Having added the value of the last card to the sum, deal the next card.

After five cards have been dealt and the loop is satisfied, have the sum printed.



Here is the complete flow chart of the main program.

5 CARD SUM FLOW CHART



Encode this flow chart as follows. Don't forget to include the RANDOM and DIM statements in lines 1 and 2. Although remotely located, they are required for the INITIALIZE CHECK LIST and GENERATE NUMBER subroutines.

5 CARD SUM
PROGRAM

```
 1 RANDOM
 2 DIM N(52)
10 GO SUB 500 <---- ACTIVATE INITIALIZE CHECK LIST
20 LET S=0
30 FOR C=1 TO 5
40 GO SUB 600 <---- ACTIVATE DEAL CARD
50 LET S=S+V
60 NEXT C
70 PRINT "SUM IS" S
80 STOP
500 FOR I=1 TO 52
505 LET N(I)=0
510 NEXT I
515 RETURN
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800
725 PRINT " OF SPADES"
730 RETURN
735 LET R=R-26
740 GO SUB 800
745 PRINT " OF HEARTS"
750 RETURN
755 LET R=R-13
760 GO SUB 800
765 PRINT " OF DIAMONDS"
770 RETURN
775 GO SUB 800
780 PRINT " OF CLUBS"
785 RETURN
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
821 LET V=R+1
825 RETURN
830 PRINT "JACK";
831 LET V=10
835 RETURN
840 PRINT "QUEEN";
841 LET V=10
845 RETURN
850 PRINT "KING";
851 LET V=10
855 RETURN
860 PRINT "ACE";
861 LET V=1
865 RETURN
999 END
```

DEAL 5 CARDS

INITIALIZE CHECK LIST

DEAL CARD

Here are three typical runs.

```
RUN 1 QUEEN OF CLUBS
       9  OF CLUBS
       4  OF CLUBS
      ACE OF CLUBS
       7  OF CLUBS
      SUM IS 31

RUN 2  7  OF DIAMONDS
       8  OF HEARTS
      QUEEN OF DIAMONDS
      ACE OF DIAMONDS
       6  OF SPADES
      SUM IS 32

RUN 3  2  OF DIAMONDS
       7  OF DIAMONDS
       9  OF HEARTS
       3  OF CLUBS
      ACE OF DIAMONDS
      SUM IS 22
```

Convinced that the program operates correctly, let's (1) eliminate the printing of the cards, (2) set up a tally list to record the range into which the sum of each hand falls, and (3) have the computer repeat the five-card deal 10,000 times.

The printing of the cards can be eliminated by deleting the PRINT statements.

Delete:

```
725 PRINT "OF SPADES"
745 PRINT "OF HEARTS"
765 PRINT "OF DIAMONDS"
780 PRINT "OF CLUBS"

820 PRINT R +1;
830 PRINT "JACK";
840 PRINT "QUEEN";
850 PRINT "KING";
860 PRINT "ACE";
```

Here is the present DETERMINE CARD FACE subroutine.

DETERMINE CARD FACE/ 5 CARD SUM *SUBROUTINE*

```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
821 LET V=R+1
825 RETURN
830 PRINT "JACK";
831 LET V=10
835 RETURN
840 PRINT "QUEEN";
841 LET V=10
845 RETURN
850 PRINT "KING";
851 LET V=10
855 RETURN
860 PRINT "ACE";
861 LET V=1
865 RETURN
```

If you delete lines 830, 840, 850, and 860, however, an error will occur when the program is running. There will be no lines for the computer to transfer to in lines 800 – 815. This is remedied by changing the line numbers after each of the THENs to an existing line. Change 860 to 861, 850 to 851, 840 to 841, and 830 to 831. The subroutine for dealing cards now looks like this.

DEAL CARD/5 CARD SUM
SUBROUTINE

```
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800
730 RETURN
735 LET R=R-26
740 GO SUB 800
750 RETURN
755 LET R=R-13
760 GO SUB 800
770 RETURN
775 GO SUB 800
785 RETURN
800 IF R=13 THEN 861
805 IF R=12 THEN 851
810 IF R=11 THEN 841
815 IF R=10 THEN 831
821 LET V=R+1
825 RETURN
831 LET V=10
835 RETURN
841 LET V=10
845 RETURN
851 LET V=10
855 RETURN
861 LET V=1
865 RETURN
```

So much for eliminating the PRINT statements. Now to set up the tally list.

Four boxes will be used to tally the range into which a sum falls. The boxes look like this:

F(1)	F(2)	F(3)	F(4)

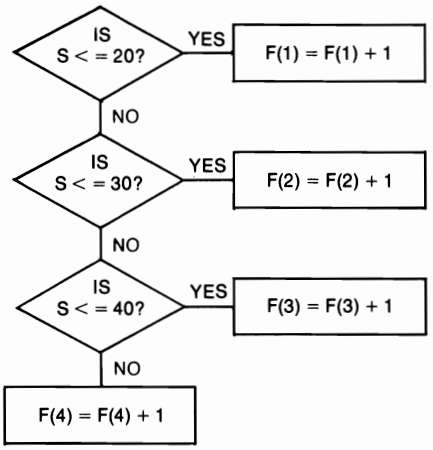
F(1) will tally all sums which are 6-20.
F(2) will tally all sums which are 21-30.
F(3) will tally all sums which are 31-40.
F(4) will tally all sums which are 41-50.
The tally list will be used in the main program after the box

NEXT C

and in place of

The box PRINT S can be eliminated because we do not need to see the sum after each deal. In place of PRINT S we'll put the statements to operate the tally list. First the computer must determine into which range the sum falls. This is done by using three IF-THEN statements. Next the computer must execute the corresponding tally statement. Here is a flow chart of the tally operation.

TALLY OPERATION



5 CARD SUM/10K FLOW CHART

So much for the tally list. The third revision is to have the computer repeat the five-card deal 10,000 times. This is done by putting the entire routine for dealing and tallying five cards in a FOR-NEXT loop, using the variable N to count off the number of repetitions. After the loop has been satisfied, the results should be printed. Here is the complete flow chart for the main program of 5 CARD SUM/10K

The following program is written from the above flow chart. Add the subroutines for the initialization of the check list and card dealing at lines 500 and 600.

PROGRAM

```
MAIN PROGRAM / 10K DEALS
    1 RANDOM
    2 DIM N(52)
    3 DIM F(4)
    5 FOR N=1 TO 10000
   10 GO SUB 500
   20 LET S=0
   30 FOR C=1 TO 5                 5 CARDS
   40 GO SUB 600
   50 LET S=S+V
   60 NEXT C
   80 IF S<=20 THEN 170            TALLY
   90 IF S<=30 THEN 150
  100 IF S<=40 THEN 130
  110 LET F(4)=F(4)+1
  120 GO TO 180
  130 LET F(3)=F(3)+1
  140 GO TO 180
  150 LET F(2)=F(2)+1
  160 GO TO 180
  170 LET F(1)=F(1)+1
  180 NEXT N
  190 PRINT "RANGE","FREQUENCY"      OUTPUT TOTALS
  200 FOR I=1 TO 4
  210 PRINT I,F(I)
  220 NEXT I
  230 STOP

INITIALIZE CHECK LIST
  500 FOR I=1 TO 52
  505 LET N(I)=0
  510 NEXT I
  515 RETURN

DEAL CARD
  600 LET R=INT(52*RND(X))+1
  605 IF N(R)=1 THEN 600
  610 LET N(R)=1
  615 GO SUB 700
  620 RETURN
  700 IF R<=13 THEN 775
  705 IF R<=26 THEN 755
  710 IF R<=39 THEN 735
  715 LET R=R-39
  720 GO SUB 800
  730 RETURN
  735 LET R=R-26
  740 GO SUB 800
  750 RETURN
  755 LET R=R-13
  760 GO SUB 800
  770 RETURN
  775 GO SUB 800
  785 RETURN
  800 IF R=13 THEN 861
  805 IF R=12 THEN 851
  810 IF R=11 THEN 841
  815 IF R=10 THEN 831
  821 LET V=R+1
  825 RETURN
  831 LET V=10
  835 RETURN
  841 LET V=10
  845 RETURN
  851 LET V=10
  855 RETURN
  861 LET V=1
  865 RETURN
  999 END
```

Here is a typical run of the program. The run time on my computer, a time-sharing system, was several hours!

```
OUTPUT  RANGE          FREQUENCY
          1              508
          2             3208
          3             4909
          4             1375
```

Now for the payoff table. In 10,000 games you win 508 games if you bet on a sum in the range 6-20, 3,208 games if you bet on a sum in the range 21-30, 4,909 games if you bet on a sum in the range 31-40, and 1,375 games if you bet on a sum in the range 41-50. Here is the table. The payoff factor for each outcome has been calculated to ten significant figures by dividing the ways to lose by the ways to win. A desk calculator helps with the arithmetic!

OUTCOME	WAYS TO WIN	WAYS TO LOSE	PAYOFF FACTOR
1	508	9492	18.68503957
2	3208	6792	2.117206982
3	4909	5091	1.037074760
4	1375	8625	6.272727272

To make the payoffs whole numbers, round each number off to the nearest whole number.

OUTCOME	PAYOFF FACTOR
1	19
2	2
3	1
4	6

These payoff factors can be used to compute the player's winnings in the game of 5 CARD SUM.

EXERCISE SET 7.2

Off-line:

1. Take the 5 CARD SUM flow chart and modify it to:
   (1) Start the player with an initial amount of money,
   (2) Permit the player to pick one of the four outcomes,
   (3) Permit the player to wager an amount of money,
   (4) Determine if the player has won,
   (5) Pay off a winning bet using the payoff factors as calculated from the experimental data in this section,
   (6) Include the replay option FLAG 99.

   In short, develop a flow chart which plays a game known as 5 CARD SUM/FLAG 99.

On-line:

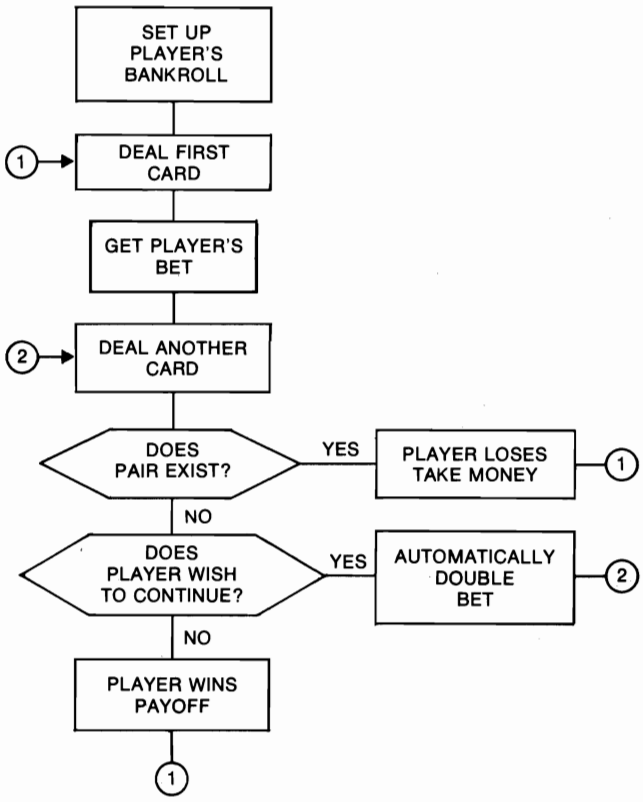
2. Write a program from your flow chart for 5 CARD SUM/FLAG 99. Test your program. You may then

wish to add to your 5 CARD SUM/FLAG 99 program subroutines for BET OK, BUSTED, VALID INPUT, LOSING PERSONALITY, WINNING PERSONALITY, INSTRUCTIONS, RING BELL, and 50 CENT TIP. Refer to YOUR BIG WHEEL program which you developed in Chapter 5 for the construction and function of each of these subroutines.

7.3 BET AGAINST A PAIR

Here's another card game, one in which the computer deals a series of cards, one at a time. After each card is dealt, the player is asked if he wants to see another one. If he says "yes," the player is gambling that the new card will not make a pair with any previously dealt card and a bet is automatically made for him. The amount bet is doubled each time he elects to look at another card. If at any time he thinks the next card may create a pair, he can say "I quit," and pick up his winnings that have accumulated to that point. If, on the other hand, he looks at the card and it does make a pair, he loses everything. To sum up, the player must decide when it's a good time to quit. Here is a flow chart of the game, which we call PAIR.

PAIR/ELEMENTARY FLOW CHART

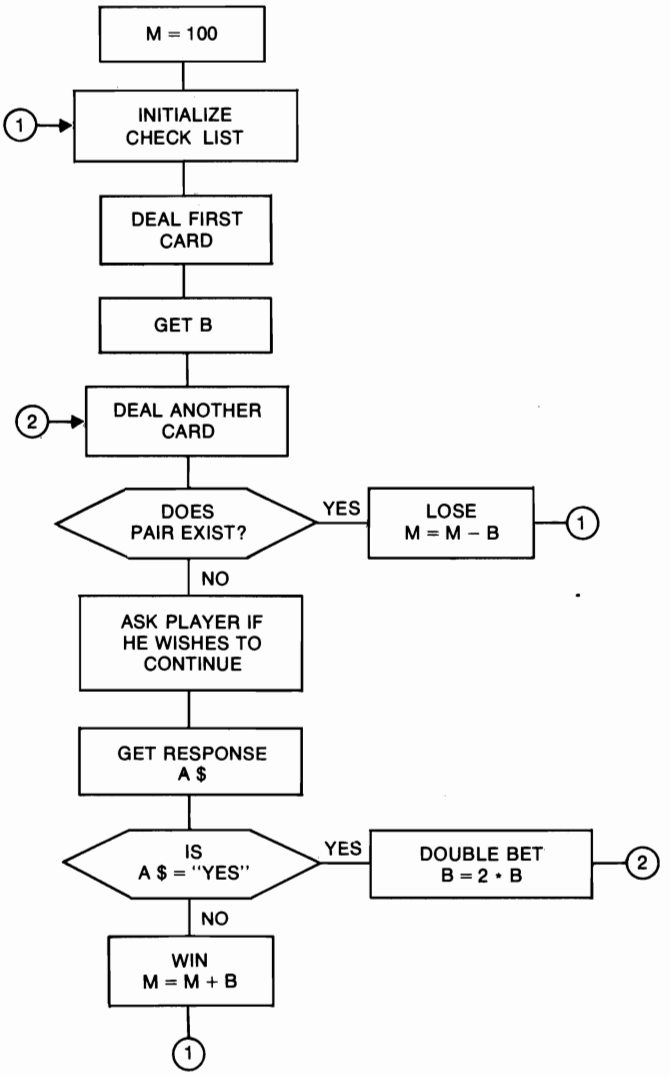


As a step toward coding a program in BASIC, redraw the flow chart substituting the following variables for some of the functional descriptions.

1. Let M hold a place for the amount of money in the

player's bankroll. The initial value of M will be 100.
2. Let B hold a place for the initial bet.
3. Let A$ hold a place for the player's response, YES or NO. Here is the new flow chart with these variables inserted.

PAIR/REVISED FLOW CHART



Look at the flow chart. The first box, M = 100, can be encoded as LET M = 100. The next two boxes can be encoded using the GO SUB command. The INITIALIZE CHECK LIST box will be translated as GO SUB 500. The DEAL FIRST CARD box will be translated as GO SUB 600. Here is a listing of the two subroutines.

INITIALIZE CHECK LIST

```
500 FOR I=1 TO 52
505 LET N(I)=0
510 NEXT I
515 RETURN
```

DEAL CARD

```
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800
725 PRINT " OF SPADES"
730 RETURN
735 LET R=R-26
740 GO SUB 800
745 PRINT " OF HEARTS"
750 RETURN
755 LET R=R-13
760 GO SUB 800
765 PRINT " OF DIAMONDS"
770 RETURN
775 GO SUB 800
780 PRINT " OF CLUBS"
785 RETURN
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
825 RETURN
830 PRINT "JACK";
835 RETURN
840 PRINT "QUEEN";
845 RETURN
850 PRINT "KING";
855 RETURN
860 PRINT "ACE";
865 RETURN
```

The next box, GET B, is encoded as INPUT B. Following the GET B box is DEAL ANOTHER CARD, and this is encoded as GO SUB 600. So much for the first five boxes.

Now we come to the diamond:

This cannot be encoded as one statement. We need to teach the computer a method or algorithm to determine if a pair exists. We'll teach the computer essentially the same method we would use when playing the game in real life. When a new card is dealt, the computer will compare it to each of the previously dealt cards. But first

we must teach it to save the cards which have been dealt.

Cards will be stored in the computer in boxes using a subscripted variable. We are not going to store cards in the form of 5 OF SPADES or 6 OF DIAMONDS, but as numbers between 1 and 52. How many boxes must we reserve in the computer's memory to hold the cards after they are dealt? To answer this question we need to know the maximum number of cards that have to be dealt before a pair occurs. There are thirteen different cards in a suit, so it is possible to deal thirteen cards without having a pair occur. The fourteenth card, however, must match one of the previous cards. Thus, we should set aside thirteen boxes in the computer's memory to hold cards until a pair occurs. To do this we'll use the variable D for card dealt. The boxes look like this:

D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)	D(9)	D(10)	D(11)	D(12)	D(13)

This array is neither a check list nor a tally list. It will be used merely to store numbers. Although there is no need to initialize the array by putting a zero in each box, a DIM statement is still required to reserve space in the computer's memory.

Let's work through an example to see how the array of boxes is used by the computer to determine if a pair exists. We need the following table for reference.

CARD FACE

		2	3	4	5	6	7	8	9	10	J	Q	K	A
S	CLUB	1	2	3	4	5	6	7	8	9	10	11	12	13
U	DIAMOND	14	15	16	17	18	19	20	21	22	23	24	25	26
I	HEART	27	28	29	30	31	32	33	34	35	36	37	38	39
T	SPADE	40	41	42	43	44	45	46	47	48	49	50	51	52

The computer deals the first card by activating the card dealing subroutine beginning at line 600. Suppose the first value of R is 45. Look at the above table. The number 45 corresponds to the 7 of spades. But, lacking the above table, the computer identifies the suit and face value of the card by the following two steps. The computer first determines the suit by comparing the value of R to the intervals 1 to 13, 14 to 26, 27 to 39, or 40 to 52. In this case, the suit is spades. After subtracting 39 from 45 to obtain a new value for R, it goes to DETERMINE CARD FACE routine where an R of 6 is found to correspond to a card with a face value of 7. After typing 7 of spades, the computer returns to the main program with the value of 6 for R. Put 6 in the first box.

D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)	D(9)	D(10)	D(11)	D(12)	D(13)
6												

The next card is dealt. Suppose the next value of R is 13. Since R is between 1 and 13 inclusive it must be a club. Furthermore, because it's between 1 and 13, no subtraction is performed. The number 13 corresponds to an ace. When the computer returns to the main program, it compares R = 13 with each number stored in the D array. R = 13 does not match D(1) = 6, so there is no pair. The computer simply stores 13 in box D(2).

D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)	D(9)	D(10)	D(11)	D(12)	D(13)
6	13											

Another card is dealt. R is 30. The suit is hearts. Subtraction of 26 yields R = 4, corresponding to a card face of 5. On return to the main program the computer compares R = 4 with D(1). There is no match. It then compares R = 4 with D(2). Again there is no match. Place 4 in box D(3).

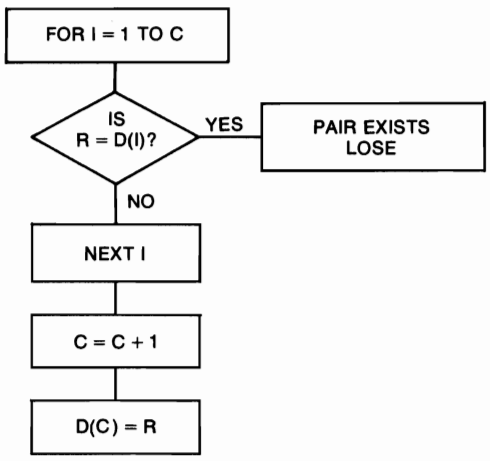
D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)	D(9)	D(10)	D(11)	D(12)	D(13)
6	13	4										

Another card is dealt. R is 38. Subtraction of 26 yields 12, corresponding to a king. There is no match with D(1), D(2), or D(3). The computer stores 12 in box D(4).

D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)	D(9)	D(10)	D(11)	D(12)	D(13)
6	13	4	12									

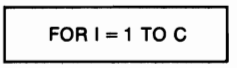
The above example illustrates the way in which the computer answers the question DOES PAIR EXIST?, the question which occupies the diamond in the middle of the PAIR flow chart at the beginning of this section. Here is the flow chart for this routine.

FLOW CHART

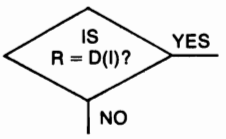


Later we shall substitute this fragment of a flow chart for the DOES PAIR EXIST? diamond in the overall

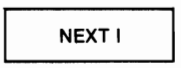
PAIR flow chart, but first an explanation of the steps in this fragment of a flow chart. The first box



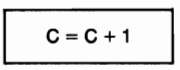
sets up a loop which directs the computer to examine all boxes which have numbers stored in them. The diamond



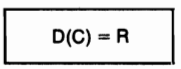
compares the value of R with the contents of one of the boxes. If R is equal to D(I), the computer takes the YES exit and tells the player that a pair exists and that he loses. If R is not equal to D(I), the computer drops down to the box



If there is no NEXT I, the loop is satisfied and the computer drops to the box



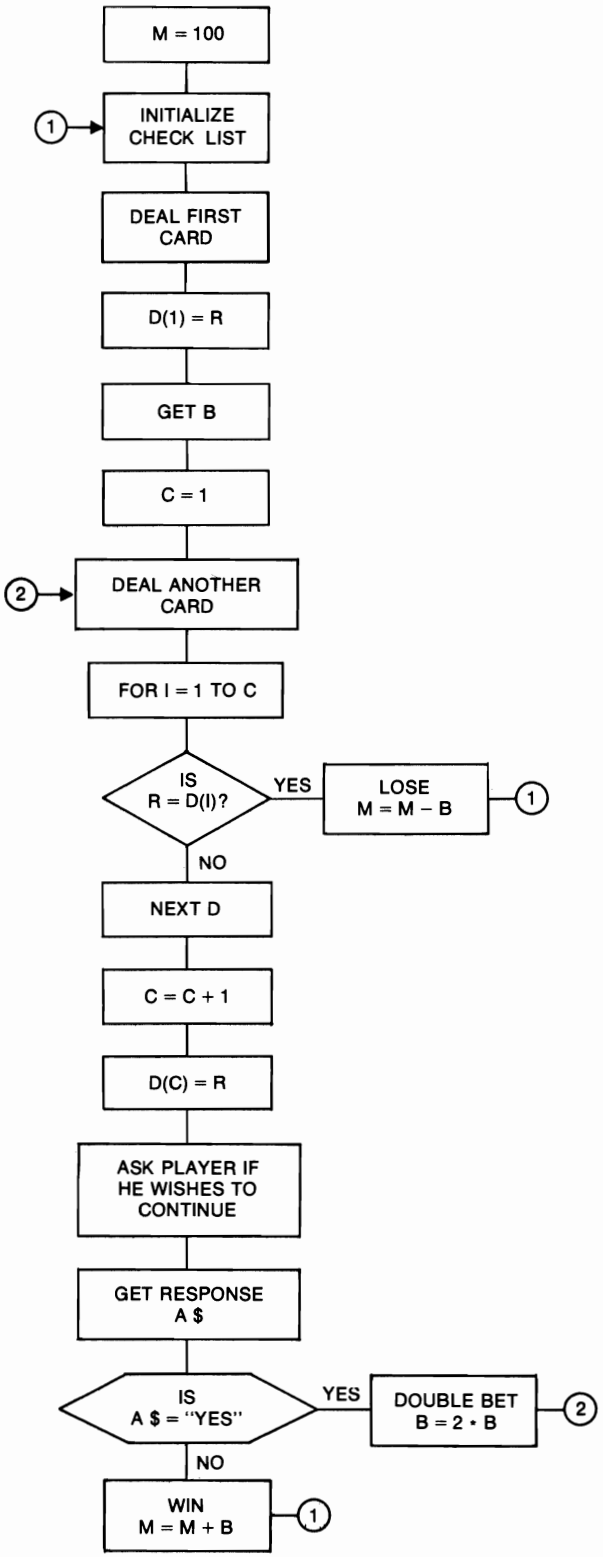
and increases the counter which is keeping a total of the number of cards that have been dealt and stored. Then the computer executes



which stores the most recent value of R in the next available box.

Here is the flow chart fragment inserted in the overall PAIR flow chart. C is a counter which keeps track of the number of cards dealt.

PAIR
FLOW CHART



From this flow chart the following elementary program is encoded.

PROGRAM

```
MAIN PROGRAM
    1 RANDOM
    2 DIM N(52),D(13)      RESERVE SPACE FOR AN ARRAY OF 52 BOXES
                           AND AN ARRAY OF 13 BOXES
   10 LET M=100
   20 GO SUB 500
   30 LET C=1
   40 GO SUB 600           DEAL FIRST CARD
   50 LET D(1)=R           SAVE FIRST CARD
   60 PRINT "BET";
   70 INPUT B
   80 GO SUB 600           DEAL NEXT CARD
   90 FOR I=1 TO C
  100 IF R=D(I) THEN 240   DOES PAIR EXIST?
  110 NEXT I
  120 LET C=C+1
  130 LET D(C)=R           SAVE NEXT CARD
  140 PRINT "ANOTHER CARD";
  150 INPUT A$
  160 IF A$="YES" THEN 210
  170 PRINT "YOU WIN $"B
  180 LET M=M+B
  190 PRINT "YOU NOW HAVE" M "DOLLARS."
  195 PRINT                SEPARATE GAMES
  200 GO TO 20
  210 LET B=2*B
  220 PRINT "NOW BETTING $"B
  230 GO TO 80
  240 PRINT "YOU LOSE $"B
  250 LET M=M-B
  260 PRINT "YOU NOW HAVE" M "DOLLARS."
  265 PRINT                SEPARATE GAMES
  270 GO TO 20
INITIALIZE CHECK LIST
  500 FOR I=1 TO 52
  505 LET N(I)=0
  510 NEXT I
  515 RETURN
```

```
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800
725 PRINT " OF SPADES"
730 RETURN
735 LET R=R-26
740 GO SUB 800
745 PRINT " OF HEARTS"
750 RETURN
755 LET R=R-13
760 GO SUB 800
765 PRINT " OF DIAMONDS"
770 RETURN
775 GO SUB 800
780 PRINT " OF CLUBS"
785 RETURN
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
825 RETURN
830 PRINT "JACK";
835 RETURN
840 PRINT "QUEEN";
845 RETURN
850 PRINT "KING";
855 RETURN
860 PRINT "ACE";
865 RETURN
999 END
```

DEAL CARD

Here is a typical run.

OUTPUT

GAME 1

```
 10  OF DIAMONDS
BET? 20
 6  OF SPADES
ANOTHER CARD? YES
NOW BETTING $ 40
 7  OF DIAMONDS
ANOTHER CARD? YES
NOW BETTING $ 80
 10  OF CLUBS
YOU LOSE $ 80
YOU NOW HAVE 20 DOLLARS.
```

GAME 2

```
 7  OF SPADES
BET? 10
QUEEN OF HEARTS
ANOTHER CARD? NO
YOU WIN $ 10
YOU NOW HAVE 30 DOLLARS.
```

GAME 3

```
KING OF DIAMONDS
BET? 20
JACK OF HEARTS
ANOTHER CARD? YES
NOW BETTING $ 40
 6  OF HEARTS
ANOTHER CARD? NO
YOU WIN $ 40
YOU NOW HAVE 70 DOLLARS.
```

Look at the output. In game 1 I lost $80 after four cards. In game 2 I elected to quit after two cards and won $10. In game 3 I quit after 3 cards and won $40. When would you quit? What is the chance of winning after five cards have been dealt?

EXERCISE SET 7.3

On-line:

1. Take the program PAIR and modify it to play 1000 games automatically. Use the program to determine the frequency with which a player loses after two cards are dealt, after three cards are dealt, etc. up to fourteen cards. To modify the program (1) delete all unnecessary statements and (2) implement a tally list. Your modified program should produce an output which looks like this:

NUMBER CARDS DEALT	LOSS FREQUENCY	CUMULATIVE LOSS FREQUENCY
2	56	56
3	114	170
4	148	318
5	162	480
6	158	638
7	146	784
8	95	879
9	65	944
10	34	978
11	18	996
12	3	999
13	0	999
14	1	1000

   A) Draw a graph of the above output with number of cards dealt along the horizontal axis, and cumulative loss frequency along the vertical axis.
   B) Now construct from your frequency table a payoff table.

2. Take the program PAIR and instead of automatically doubling the player's bet each time he elects to continue, have him bet one amount at the beginning of the game. If he elects to quit, then pay him off using the payoff factor for the number of cards dealt. You will find it helpful to store the payoff factors in boxes which look like this:

P(2)	P(3)	P(4)	P(5)	P(6)	P(7)	P(8)	P(9)	P(10)	P(11)	P(12)	P(13)

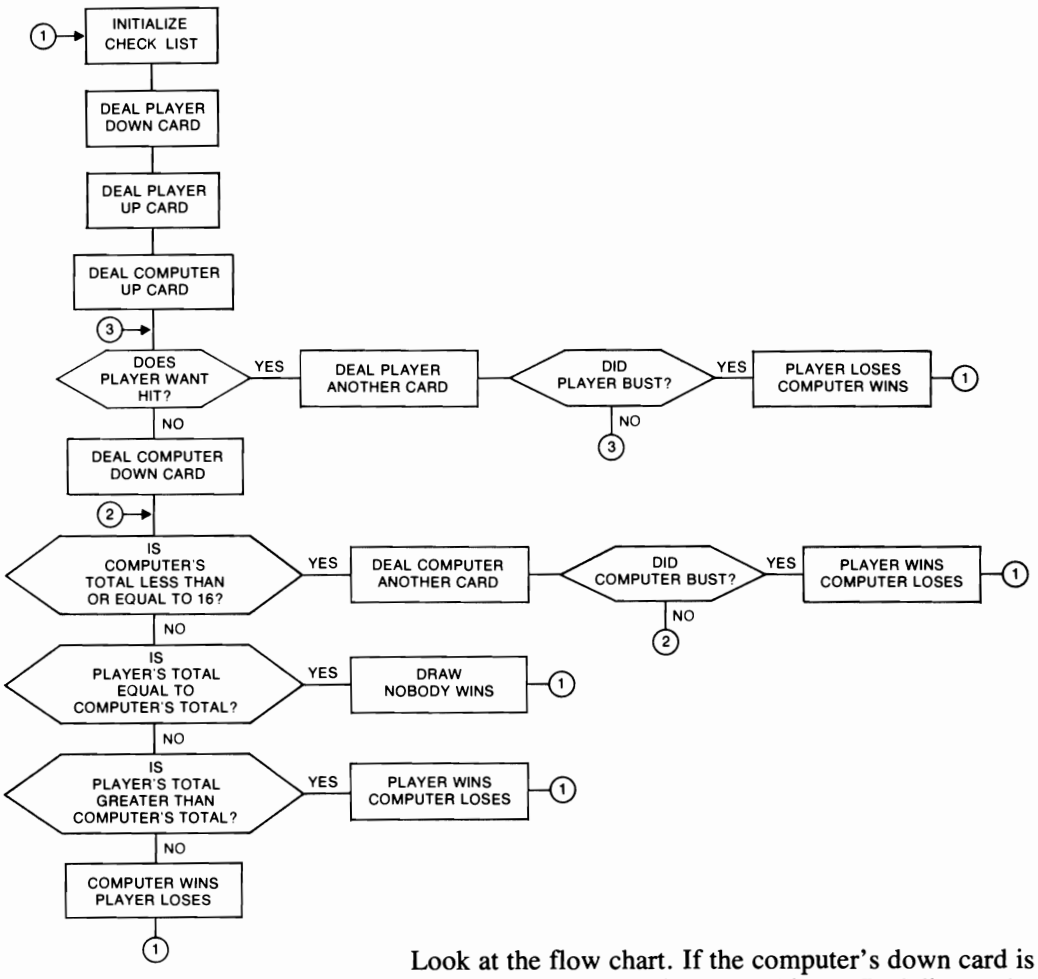
7.4 BLACKJACK

Blackjack or Twenty-one is a popular card game. Let's teach the computer to play an elementary version of the game[1] in which the user plays against the computer which is known as the "dealer." Cards are dealt to the player and to the dealer; the hand that totals 21 or is closer to 21 without going over 21 wins. Ties are considered a draw. The face cards, king, queen, jack, are worth 10 points; 10, 9, 8, 7, 6, 5, 4, 3, 2 are worth their face value, and aces are worth 11. The play of the game will be as follows.

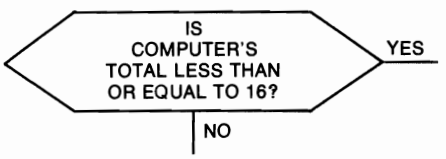
The player is asked to bet and then receives two cards. The first one is called the "down" card and the second one the "up" card. Then the computer deals itself an "up" card. The player is then asked if he wants another card, or "hit." If he does, he is dealt an additional card. This process continues until he declines a "hit" or until he "busts," that is, the total of the cards in his hand goes over 21. If the player "busts," he automatically loses. If he hasn't busted and declines additional cards, then he is said to "stand." When the player stands, it is time for the computer to play its hand. It deals itself a second card, the down card. It continues to deal itself additional cards as long as the total of its hand is less than or equal to 16. Thus, if its total is 16 and it deals itself a 6 of any suit, it goes over 21, that is the computer busts. If the computer busts, then it obviously loses. But if the computer eventually stands, then the totals of the two hands are compared and the hand with the greater total wins. If the player wins, the payoff is 1 to 1. That is, if he wagered one dollar, he picks up his original dollar plus another one. A tie is considered a draw. Here is a flow chart of Blackjack. For the sake of simplicity, I will not consider the betting routine at this time. The player is merely told whether he wins or loses.

[1] For a more detailed description of Twenty-one see Donald D. Spencer, *Game Playing with Computers*, Hayden. This book describes many games which can be programmed.

BLACKJACK/ELEMENTARY FLOW CHART



Look at the flow chart. If the computer's down card is the ace of spades and its up card is the 7 of diamonds, which exit does it take from the diamond?



As a step toward coding the above flow chart in BASIC, let's introduce a variable, P, to represent the total of the player's cards, and another variable, C, to represent the total of the computer's cards. Furthermore, as in the game of 5 CARD SUM, numerical values from 2 through 11 will have to be assigned to the cards as they are dealt using the variable V. Here is a copy of the DETERMINE CARD FACE subroutine.

DETERMINE CARD FACE
SUBROUTINE

```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
825 RETURN
830 PRINT "JACK";
835 RETURN
840 PRINT "QUEEN";
845 RETURN
850 PRINT "KING";
855 RETURN
860 PRINT "ACE";
865 RETURN
```

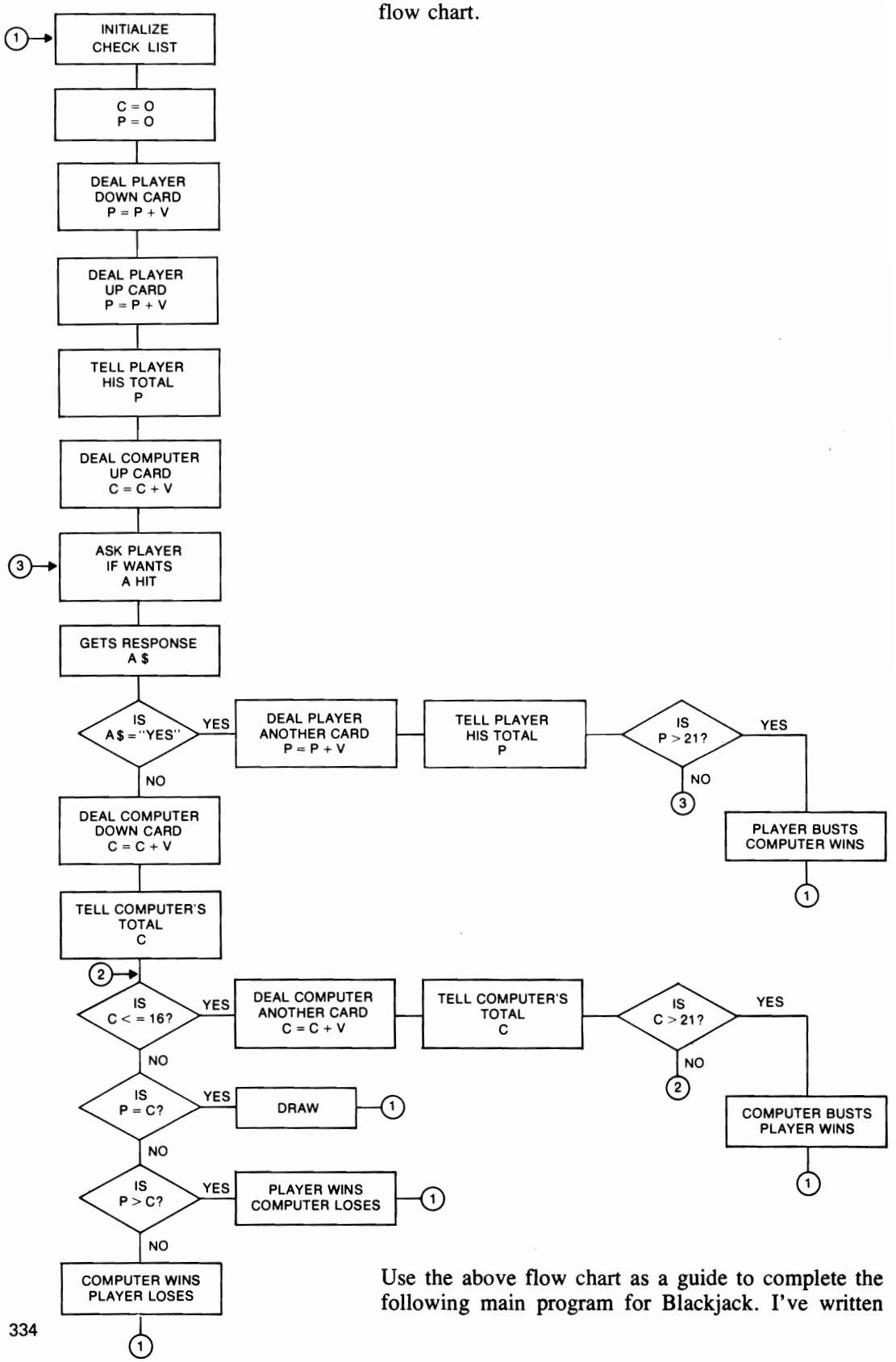
When an ace is dealt, the value of the card is 11. After 860 PRINT "ACE";, insert 861 LET V = 11. Likewise insert 851 LET V = 10 after the king is dealt, insert 841 LET V = 10 after the queen is dealt, and insert 831 LET V = 10 after the jack is dealt. When the computer executes PRINT R + 1; in line 820, the value of the card is R + 1. Insert 821 LET V = R + 1. The card face subroutine now looks like:

DETERMINE CARD FACE/21
SUBROUTINE

```
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
821 LET V=R+1
825 RETURN
830 PRINT "JACK";
831 LET V=10
835 RETURN
840 PRINT "QUEEN";
841 LET V=10
845 RETURN
850 PRINT "KING";
851 LET V=10
855 RETURN
860 PRINT "ACE";
861 LET V=11
865 RETURN
```

Therefore, on returning to the main program, the value of V must be added to either P or C depending upon whether the player or computer was dealt a card. If the player is dealt the card, the computer must execute LET C = C + V. To permit the player to respond with YES or NO to the question HIT? when it is asked by the

BLACKJACK FLOW CHART



computer, use the string variable A$. Here is the revised flow chart.

Use the above flow chart as a guide to complete the following main program for Blackjack. I've written

some of the statements and have left others blank or incomplete. Fill in the empty blanks and lines.

```
  1 RANDOM
  2 DIM N(52)
 10 GO SUB 500
 20 LET C = _____
 30 LET P = _____
 40 PRINT "YOUR DOWN CARD IS  ";
 50 GO SUB _____
 60 LET P = P + _____
 70 PRINT "YOUR UP CARD IS  ";
 80 _____
 90 LET P = _____
100 PRINT "YOUR TOTAL IS  ";
110 PRINT "MY UP CARD IS  ";
120 GO SUB _____
130 LET C = C + _____
140 PRINT "HIT";
150 INPUT A$
160 IF A$ = "YES" THEN 380
170 PRINT "MY DOWN CARD IS  ";
180 GO SUB _____
190 _____
200 PRINT "MY TOTAL IS"C
210 IF C< = 16 THEN 300
220 IF P = C THEN _____
230 IF P > C THEN _____
240 PRINT "I WIN."
250 GO TO _____
260 PRINT "_____."
270 GO TO _____
280 PRINT "DRAW."
290 GO TO _____
300 PRINT "I TAKE ANOTHER CARD  ";
310 _____
320 LET C = _____
330 PRINT "MY TOTAL IS" C
340 IF C _____ 21 THEN _____
350 GO TO _____
360 PRINT "I BUST. YOU WIN."
370 _____
380 PRINT "YOUR NEXT CARD IS  ";
390 _____
400 _____
410 PRINT "YOUR TOTAL IS" P
420 IF P _____ 21 THEN _____
430 GO TO _____
440 PRINT "YOU BUST. _____."
450 GO TO 10
```

Here is how I filled in the blanks.

BLACKJACK
MAIN PROGRAM

```
  1 RANDOM
  2 DIM N(52)
 10 GO SUB 500
 20 LET C=0
 30 LET P=0
 40 PRINT "YOUR DOWN CARD IS ";
 50 GO SUB 600
 60 LET P=P+V
 70 PRINT "YOUR UP CARD IS ";
 80 GO SUB 600
 90 LET P=P+V
100 PRINT "YOUR TOTAL IS "P
110 PRINT "MY UP CARD IS ";
120 GO SUB 600
130 LET C=C+V
140 PRINT "HIT";
150 INPUT A$
160 IF A$="YES" THEN 380
170 PRINT "MY DOWN CARD IS ";
180 GO SUB 600
190 LET C=C+V
200 PRINT "MY TOTAL IS "C
210 IF C<=16 THEN 300
220 IF P=C THEN 280
230 IF P>C THEN 260
240 PRINT "I WIN."
250 GO TO 10
260 PRINT "YOU WIN."
270 GO TO 10
280 PRINT "DRAW."
290 GO TO 10
300 PRINT "I TAKE ANOTHER CARD ";
310 GO SUB 600
320 LET C=C+V
330 PRINT "MY TOTAL IS "C
340 IF C>21 THEN 360
350 GO TO 210
360 PRINT "I BUST. YOU WIN."
370 GO TO 10
380 PRINT "YOUR NEXT CARD IS ";
390 GO SUB 600
400 LET P=P+V
410 PRINT "YOUR TOTAL IS "P
420 IF P>21 THEN 440
430 GO TO 140
440 PRINT "YOU BUST. I WIN."
450 GO TO 10
```

To complete the program, add the subroutines to initialize the check list and deal a card.

BLACKJACK
SUBROUTINES

```
500 FOR I=1 TO 52
505 LET N(I)=0
510 NEXT I
515 RETURN
600 LET R=INT(52*RND(X))+1
605 IF N(R)=1 THEN 600
610 LET N(R)=1
615 GO SUB 700
620 RETURN
700 IF R<=13 THEN 775
705 IF R<=26 THEN 755
710 IF R<=39 THEN 735
715 LET R=R-39
720 GO SUB 800
725 PRINT " OF SPADES"
730 RETURN
735 LET R=R-26
740 GO SUB 800
745 PRINT " OF HEARTS"
750 RETURN
755 LET R=R-13
760 GO SUB 800
765 PRINT " OF DIAMONDS"
770 RETURN
775 GO SUB 800
780 PRINT " OF CLUBS"
785 RETURN
800 IF R=13 THEN 860
805 IF R=12 THEN 850
810 IF R=11 THEN 840
815 IF R=10 THEN 830
820 PRINT R+1;
821 LET V=R+1
825 RETURN
830 PRINT "JACK";
831 LET V=10
835 RETURN
840 PRINT "QUEEN";
841 LET V=10
845 RETURN
850 PRINT "KING";
851 LET V=10
855 RETURN
860 PRINT "ACE";
861 LET V=11
865 RETURN
999 END
```

Here is a typical run of this Blackjack program.

OUTPUT

```
GAME 1
YOUR DOWN CARD IS  6  OF CLUBS
YOUR UP CARD IS  7  OF SPADES
YOUR TOTAL IS  13
MY UP CARD IS  9  OF CLUBS
HIT? YES
YOUR NEXT CARD IS  4  OF HEARTS
YOUR TOTAL IS  17
HIT? NO
MY DOWN CARD IS  3  OF CLUBS
MY TOTAL IS  12
I TAKE ANOTHER CARD KING OF CLUBS
MY TOTAL IS  22
I BUST. YOU WIN.

GAME 2
YOUR DOWN CARD IS  3  OF CLUBS
YOUR UP CARD IS  7  OF CLUBS
YOUR TOTAL IS  10
MY UP CARD IS  7  OF HEARTS
HIT? YES
YOUR NEXT CARD IS  6  OF DIAMONDS
YOUR TOTAL IS  16
HIT? YES
YOUR NEXT CARD IS  9  OF CLUBS
YOUR TOTAL IS  25
YOU BUST. I WIN.

GAME 3
YOUR DOWN CARD IS  4  OF CLUBS
YOUR UP CARD IS  10  OF DIAMONDS
YOUR TOTAL IS  14
MY UP CARD IS  6  OF SPADES
HIT? YES
YOUR NEXT CARD IS  3  OF CLUBS
YOUR TOTAL IS  17
HIT? NO
MY DOWN CARD IS  7  OF DIAMONDS
MY TOTAL IS  13
I TAKE ANOTHER CARD  7  OF HEARTS
MY TOTAL IS  20
I WIN.

YOUR DOWN CARD IS  6  OF CLUBS
YOUR UP CARD IS  3  OF CLUBS

↑C
```

Look at the output. How many times did the computer win?

EXERCISE SET 7.4

On-line:

1. Take the program BLACKJACK and add steps (1) to permit the player to bet, (2) to permit the player to signal when he wants to quit, (3) to improve the layout, and (4) to give the game more personality.
2. The present BLACKJACK program does not recognize a BLACKJACK, that is, any ace-10 combination. Modify the BLACKJACK program so that if

the player's first two cards total 21, he automatically wins. If the dealer also has BLACKJACK, the hand is a draw. Make the payoff 1.5 to 1 on BLACKJACK if the player wins.

3. The present BLACKJACK program assigns the value of eleven to an ace. Modify the program so that if the player is dealt one or more aces, he has the option of telling the computer whether to count each ace as 1 or 11. The player should be given this option each time before his hand is evaluated by the computer.

LOGOUT

You have learned how to make the computer work for you, although the games you have taught it are valuable mainly as entertainment. The computer, however, does a great deal of important work in modern society and shows promise of doing even more work in the future. At this point, to further your understanding of computers, two kinds of books will be useful to you. One kind of book is the book that does not require access to a computer. The other kind of book is like this one, it assumes you have a computer to program.

Books of the first kind, requiring little or no knowledge of programming, describe computer hardware and its operation, or the application of computers to medicine, law, science and engineering, education, humanities, business, machine control, transportation, etc. Books of the second kind require or teach you a computer language which you then use to solve problems in mathematics, science, or business.

There are now many hundreds of books in the computer field. The books we have listed below are simply representative of the books you will find in your book store. We have listed these at the risk of omitting other, possibly better books, on the pragmatic ground that some help is better than none.

While many of the publishers listed below will be familiar to you, more and more materials related to computer education, particularly materials based on the assumption you have a computer, are being made available by the computer manufacturers themselves.

NO COMPUTER REQUIRED

General Applications

Martin, J. and Norman, A.R.D. **The Computerized Society.** Prentice-Hall, 1970.

Sanders, D. H. **Computers in Society: An Introduction to Information Processing.** McGraw-Hill, 1973.

Spence, D. D. **Computers in Society: The Wheres, Whys, and Hows of Computer Use.** Hayden, 1974.

Business Applications

Fiori, W. M. **Introduction to the Computer: The Tool of Business.** Prentice-Hall, 1973.

COMPUTER REQUIRED OR HELPFUL

More About BASIC

Kemeny, J. G. and Kurtz, T. E. **BASIC Programming.** Wiley, 1971.

More About Games

Spencer, D. D. **Game Playing with Computers.** Hayden, 1968.

Ahl, D. **101 BASIC Computer Games.** Digital Equipment Corp., 1973.

Computer Science

Forsythe, A. I., Keenan, T. A., Organick, E. I. and Stenberg, W. **Computer Science: A First Course.** Wiley, 1969.

Knuth, D. E. **The Art of Computer Programming** (7 vols.). Addison-Wesley, 1968 **et seq.**

Math Applications

Sage, E. R. **Problem-Solving with the Computer.** ENTELEK, 1969.

Dorn, W. S., Greenburg, H. J., and Keller, M. K. **Mathematical Logic and Probability with BASIC Programming.** Prindle, Weber and Schmidt, 1972.

Dorn, W. S., Bitter, G. G., and Hector, D. L. **Computer Applications for Calculus.** Prindle, Weber & Schmidt, 1972.

Gruenberger, F. and Jaffrey, G. **Problems for Computer Solution.** Wiley, 1965.

Applications Packages

Braun, L. (Director) **Huntington Two Project.** Materials available from:

(1) Data General, Southboro, Mass. 01772
(2) Digital Equipment Corp., Maynard, Mass. 01754
(3) Hewlett-Packard, Cupertino, Calif. 95014
(4) Wang Laboratories, Tewksbury, Mass. 01876

INDEX

ABOUT THE AUTHOR

Edwin R. Sage is head of the Mathematics Department at Middlesex, a private secondary school in Concord, Massachusetts. He has been a member of the faculty there since 1959, except for the academic year 1965-1966 when he served as head of the Mathematics Department at Wayland Academy in Beaver Dam, Wisconsin. Mr. Sage is also the author of the widely used ENTELEK text *Problem-Solving with the Computer,* which presents 40 carefully graded problems drawn from the traditional secondary school math curriculum. He has participated in numerous seminars, panels, and programs on the use of computers in education. He was graduated from Williams College and received an MA in mathematics from Boston College. He is a member of the National Council of Teachers of Mathematics and other professional bodies.