CHRISTOPHER LAMPTON

How to Create Adventure Games

A COMPUTER-AWARENESS FIRST BOOK

HOW TO CREATE ADVENTURE GAMES

CHRISTOPHER LAMPTON

HOW TO CREATE ADVENTURE GAMES

A Computer-Awareness
First Book

Franklin Watts
New York | London | Toronto
Sydney | 1986

Diagram courtesy of Vantage Art, Inc.

Library of Congress Cataloging in Publication Data

Lampton, Christopher.
How to create adventure games.

(A Computer-awareness first book)
Includes index.
Summary: Provides instructions for writing a computer program for an adventure game using BASIC.
1. Computer adventure games—Juvenile literature.
2. BASIC (Computer program language)—Juvenile literature. [1. Computer games. 2. BASIC (Computer program language) 3. Programming languages (Computers) 4. Programming (Computers)] I. Title. II. Series.
GV1469.22.L35 1986 794.8'2 85-26511
ISBN 0-531-10119-3

Printed in the United States of America
5 4 3 2 1

CONTENTS

HOW TO CREATE ADVENTURE GAMES

CHAPTER ONE

THE IMAGINATION MACHINE

A computer is an imagination machine. It can take you places where you've never been before, where *nobody* has ever been.

A computer is a magic wand. It can make things happen: exciting things, wonderful things.

If you have ever played games on a computer, you know about these wonderful places and things. The most wonderful thing about the computer, however, is that you can use it to create your *own* wonderful places, where you and your friends can go. You can create strange worlds, exotic lands. You can even have adventures in these new worlds that you have created.

To create these new worlds, though, you must know how to program a computer, using a computer programming language. If you don't know what programming is, or what a programming language is, then you should do a little research before you read this book. In the pages that follow, we'll show you how to create a kind of computer game called an *adventure game*, using the programming language known as BASIC. Although we will explain step by step how an adventure program is written, you will need to know some things about writing programs in BASIC before you start. That doesn't mean that you should be an expert program-

mer, just that you should know something about how to type a BASIC program on your computer. You should be familiar with such BASIC commands as PRINT and GOTO and IF-THEN and FOR-NEXT. You should know how to use variables. You should know what a string is, and a number, and how to perform simple arithmetic in a computer program.

An adventure game isn't the easiest kind of program to write, though it is far from the hardest. We will write some parts of the program for you. At the end of this book, we'll even give you an Adventure Game Skeleton, with blank spaces left for the routines that you have to write yourself. To write an adventure game using this skeleton, all you have to do—literally—is fill in the blanks.

There is one problem that you may encounter in using the program routines that we give you in this book. They may not work properly on a few computers. The truth is, there are many different versions of the BASIC programming language, and different computers understand different versions. This makes it difficult to write a BASIC program on one computer and then transfer it to another. The programs in this book were written on computers from Radio Shack and Commodore, but the computer that you are using may be manufactured by IBM or Apple or any of a hundred other computer companies. Fortunately, all but a few of these computers understand a special dialect of BASIC called Microsoft BASIC (written by the Microsoft Corporation of Washington State), which is the version of BASIC that we will use in this book. Computers that use Microsoft BASIC, such as those made by Apple, IBM, Radio Shack, and Commodore, should have no trouble running these programs.

What is an adventure game? If you have never played one, then you have missed an exciting experience. An adventure game is like a novel, except that *you* are the main character and can make decisions about what will happen in the story. Unlike a novel, an adventure game is different every time you play it, because the course of the story is determined by what you choose to do.

The first adventure game was written in the mid-1970s by a

programmer named William Crowther. It was written in a programming language called FORTRAN and was intended to be used on very large computers. (Today, you can buy versions of this program to run on home computers.) It was not given a name—after all, it was the only program of its sort at the time—but it quickly became known as Adventure or the Colossal Cave Adventure or, after other adventure games were written, the Original Adventure. Shortly after Crowther unleashed the program on the world, it was rewritten and expanded by a second programmer named Don Woods, and so it is sometimes also known as the Crowther and Woods Adventure. The program became so popular that it can be found on computers at universities all over the world.

Like most of the adventure programs that have followed, the Crowther and Woods Adventure begins by telling the player where he or she is and what is visible. The opening description in the Original Adventure goes something like this:

YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK BUILDING. AROUND YOU IS A FOREST. A SMALL STREAM FLOWS OUT OF THE BUILDING AND DOWN A VALLEY.

The game then asks the player what he or she would like to do next. The player must respond by typing a one- or two-word sentence. The words in this sentence must be words that the game program will recognize. (The game will inform the player if it doesn't recognize a word or words.) For instance, if a player at the beginning of the Original Adventure types "GO BUILDING," the program will move your character into the building, then describe what the building looks like on the inside:

YOU ARE INSIDE A BUILDING, A WELL HOUSE FOR A LARGE SPRING.
THERE IS A BOTTLE OF WATER HERE.

THERE IS A SHINY BRASS LAMP HERE.
THERE IS A SET OF KEYS HERE.
THERE IS FOOD HERE.

Notice that the game describes a number of objects that are present in the current location. This usually means that the player can pick up one or more of these objects and carry them about. Therefore, if you type "GET LAMP" or "TAKE LAMP," the game will respond:

OK.

meaning that the lamp has been taken, and is now in your possession. To double-check that the lamp is in your possession, you can type "INVENTORY." This tells the game to print out a complete list of everything that you are carrying, like this:

YOU HAVE:
LAMP

If you then pick up another object, it will also be included in the list next time you type "INVENTORY." Should you want to get rid of an object, such as the lamp, you can type "DROP LAMP." The lamp will be placed in whatever room you are standing in when you drop it. (In adventure language, any location in the game is referred to as a *room*, even if it is out of doors.)

Eventually, the player of the Crowther and Woods Adventure finds his or her way into a huge underground cavern. In that cavern are deadly knife-throwing dwarfs, fire-breathing dragons, and valuable treasures. Every time you find a valuable treasure, you are given a certain number of points. Once you have achieved the maximum number of points, you have won the game. Of course, getting the treasures isn't easy. In fact, it requires a great deal of ingenuity. And you will probably get killed a few times before you have finished playing the game. Fortunately, every time you

are killed, you are automatically brought back to life outside the cave, to start the adventure over again.

Most of the adventure games that have followed the Original Adventure have been patterned after it in some way. The first adventures written for microcomputers were produced by a young programmer named Scott Adams, who founded a company called Adventure International to publish a series of twelve adventure games, with names like Adventureland and Voodoo Castle and Pyramid of Doom, which are still available for several brands of microcomputers. And in the late 1970s, a group of programmers at MIT wrote their own version of the Original Adventure, called Zork, which is now available, along with several sequels, on microcomputers. Unlike Adventure, Zork does not restrict the player to one- and two-word commands. In fact, Zork can understand entire sentences, as long as they are constructed according to a loose set of rules. The microcomputer version of Zork is published by a company called Infocom, along with a series of adventure games regarded by some as the finest computer adventures ever written.

Other adventure game writers have introduced other innovations. Many adventures include computer-generated pictures, or *graphics*, so that you can *see* the places described in the text. Although the games in this book use only text, not pictures, an ingenious programmer can figure out ways to incorporate graphics into an adventure game.

CHAPTER TWO

MAPPING YOUR IMAGINATION

The first thing you need when you write an adventure game is an idea. The second thing you need is a map.

Getting an idea is easy. Ideas are all around you; you just have to look for them. You can base an adventure game on a favorite movie, or on a novel you've read, or on a comic book. Or you can think up a completely new idea for a story, if you've got a sharp imagination.

Getting a map is harder. You can't walk into a gas station and buy a map of your adventure. You have to make one yourself. We recommend that you set aside several sheets of paper and a sharp pencil for map making. And make sure that the pencil has a good eraser!

The typical adventure game takes the form of a *quest*. A quest is a story in which the hero is searching for something, usually an object or a person. The object could be a jewel or a magic book or just about anything of value. The person could be a beautiful princess, or a lost child, or an important scientist—or a master criminal.

In order to find the object or objects of a quest, powerful obstacles must first be overcome. And in order to overcome these

obstacles, certain devices must be obtained, or certain tasks must be accomplished.

The adventure that we will create in this book will be a straightforward quest adventure. The object will be a magic jewel. Once the player has the magic jewel, the game will be over. Getting the magic jewel, of course, will not be easy. It will require that the player travel a great distance, learn to perform certain tricks, and obtain certain objects.

Here is the plot of the game:

You (the player) have been left a mysterious diary and a box by your late uncle. When you read the diary, you learn that your uncle had discovered a gateway into another world! He has left you instructions for reaching that other world, and a magic formula that will help you in your journey, so that you can continue his search for a fabled magic jewel.

According to the diary, you can reach the other world by mixing the magic formula (which is in the box) with sodium chloride and rainwater. If you don't know what sodium chloride is, look it up in the dictionary (which is also in the game, in a nearby room). The dictionary will tell you that sodium chloride is ordinary table salt. In your kitchen, you find a shaker of salt. In the garage, you discover a barrel filled with rainwater. When you pour the formula and the salt into the barrel—*poof*! You are transported to the other world.

In the new world, you find an open field and a tree. If you try to climb the tree, you are told that the branches are too high. If you try to use a ladder—surprise!—it sinks into the soft ground below the tree and disappears!

There is a way to climb the tree, however: you must JUMP. On the lowest branch of the tree, you will find a MAGIC FAN. You can GET this fan and add it to your INVENTORY, if you so wish. You can get back out of the tree by typing GO DOWN.

There is also an object in the open field, but you cannot see it at first because it is buried. If you have a shovel in your inventory,

you can DIG for it. It is a GOLDEN SWORD, and it will come in handy later.

To the north, you find a boat by a river. You can enter the boat by typing GO BOAT. Making the boat move to the other side of the river is more difficult, but if you type WAVE FAN, the magic fan will create a breeze that blows the boat across. You can then leave the boat by typing LEAVE BOAT or EXIT BOAT.

Further north, you come to a large castle. If you try to GO NORTH again and enter the castle, a nasty-looking guard stops you—unless you have the sword, in which case the guard retreats inside the castle.

Inside, you find a glass case that contains a jewel—the magic jewel! But first you must OPEN the case—and you cannot, because it is electrified! (You might ask how glass can be electrified. Nobody ever claimed that this adventure was perfectly logical. How often do you find magic fans in trees?)

How can you open an electrified case? The solution is that . . . you overlooked an object earlier. When you climbed the tree, you only JUMPed to the first limb. However, there was another limb above that one. With a second JUMP command, you would have ended up in the top of the tree, where you would have found a pair of rubber gloves.

You go back and get the gloves. Now, the command WEAR GLOVES will make you immune to the electrified case. You can OPEN CASE, GET JEWEL—and win the game!

That's the adventure game we are going to write. Now we need a map. There are two steps to making a map. First, you must make a list of all the rooms in your game. A *room*, in an adventure game, is a location that the player can go to. It can be indoors or outdoors. In this game, there are nineteen rooms. When you make a list of these rooms, you should write their names exactly as they will appear in the game, minus the words "YOU ARE," which will always appear in front of the description of the room. Here is a list of the rooms in our adventure:

1. IN YOUR LIVING ROOM.
2. IN THE KITCHEN.
3. IN THE LIBRARY.
4. IN THE FRONT YARD.
5. IN THE GARAGE.
6. IN AN OPEN FIELD.
7. AT THE EDGE OF A FOREST.
8. ON A BRANCH OF A TREE.
9. ON A LONG, WINDING ROAD.
10. ON A LONG, WINDING ROAD.
11. ON A LONG, WINDING ROAD.
12. ON THE SOUTH BANK OF A RIVER.
13. INSIDE THE WOODEN BOAT.
14. ON THE NORTH BANK OF A RIVER.
15. ON A WELL-TRAVELED ROAD.
16. IN FRONT OF A LARGE CASTLE.
17. IN A NARROW HALL.
18. IN A LARGE HALL.
19. ON THE TOP OF A TREE.

Notice that every room has a number. This is *very* important. The adventure game program uses these numbers to identify what room you are in.

Now, we need a way to get this list of rooms into our game program. For that, we will need a *variable array*.

A variable array is a method of putting lists of things in a BASIC program. In Microsoft BASIC, there are *numeric arrays*, for keeping lists of numbers, and *string arrays*, for keeping lists of strings (that is, words and sentences).

We will store the names of our rooms in a string array, because these names are strings of characters. But we will also use numeric arrays in our adventure program, to keep lists of numbers.

An array has a name, just like an ordinary variable. Unlike the name of a normal variable, the name of an array variable must be followed by a pair of parentheses—"( )". These parentheses

should contain a number—or a variable or arithmetic operation equal to a number. This number is called the *subscript* of the variable and should be between 0 and 32767. A typical array variable might look like this:

```
AB(17)
```

The name of this array is AB. The subscript of this array variable is 17.

Before we can use an array variable, however, we must *dimension* the array—that is, we must use the BASIC command DIM, followed by the name of the variable, and a number in parentheses indicating how many items are in the list we want to store. For the list of room names, we will use an array called R$. (Notice the dollar sign after the name. This tells us that R$ is a string array.) Since there are nineteen room names, we would dimension this array like this:

```
DIM R$(19)
```

There are nineteen *elements* in this array. That means that this array is actually made up of nineteen different array variables. Here are their names:

R$(1)	R$(6)	R$(11)	R$(16)
R$(2)	R$(7)	R$(12)	R$(17)
R$(3)	R$(8)	R$(13)	R$(18)
R$(4)	R$(9)	R$(14)	R$(19)
R$(5)	R$(10)	R$(15)	

Actually, there is a twentieth element in this array, called R$(0). We are going to ignore this element for now, since there is no room 0 in our game—and we are going to use each element in the array to store the name of one room.

Here is a subroutine that will set each element of array R$

equal to the name of one room. This subroutine will eventually become part of our adventure game program:

```
27000 R$(1)="IN YOUR LIVING ROOM."
27010 R$(2)="IN THE KITCHEN."
27020 R$(3)="IN THE LIBRARY."
27030 R$(4)="IN THE FRONT YARD."
27040 R$(5)="IN THE GARAGE."
27050 R$(6)="IN AN OPEN FIELD."
27060 R$(7)="AT THE EDGE OF A FOREST."
27070 R$(8)="ON A BRANCH OF A TREE."
27080 R$(9)="ON A LONG, WINDING ROAD."
27090 R$(10)="ON A LONG, WINDING ROAD."
27100 R$(11)="ON A LONG, WINDING ROAD."
27110 R$(12)="ON THE SOUTH BANK OF A RIVER."
27120 R$(13)="INSIDE THE WOODEN BOAT."
27130 R$(14)="ON THE NORTH BANK OF A RIVER."
27140 R$(15)="ON A WELL-TRAVELED ROAD."
27150 R$(16)="IN FRONT OF A LARGE CASTLE."
27160 R$(17)="IN A NARROW HALL."
27170 R$(18)="IN A LARGE HALL."
27180 R$(19)="ON THE TOP OF A TREE."
29900 RETURN
```

If you use this subroutine in a program, you must also include the instructions DIM R$(NR) and GOSUB 27000 very early in the program, in that order. (NR should be equal to the number of rooms in your program.)

Type this subroutine and save it on a disk or tape. We'll be using it in a minute.

If you want to adapt this subroutine for your own adventure game programs, you must substitute the names of your own rooms. Add or subtract lines from this routine as needed.

In our finished adventure game, there will be a variable called R. This is a numeric variable, which means that it will

always be equal to a number. The number that it will be equal to is the number of the room that the player is in currently. When the description of a room needs to be printed on the screen of the computer, the program will consult variable R to find out what room is to be described. It will then look at array R$, which we created in the subroutine above, to find the proper description. How will it know which description goes with room R? Because R will be equal to the subscript (the number in parentheses) of the array element containing the description.

For instance, if variable R is equal to 6, that means that the player is in room 6. What is the description of room 6? If we look at the subroutine above, we see that room 6 is described like this:

"IN AN OPEN FIELD."

Now we are going to give you a subroutine that will print out a description of the current room. When the adventure program needs to describe the current room (room R), it can call this subroutine with a GOSUB 700 instruction:

```
691 REM ** ROOM DESCRIPTION ROUTINE
696 REM
700 PRINT : PRINT "YOU ARE";R$(R)
710 RETURN
```

This is a very simple subroutine. All it does is print the words "YOU ARE," followed by the description of the room that you have put in array R$.

To demonstrate this subroutine, add it to the earlier subroutine and then add these lines:

```
20 NR=19
30 DIM R$(20)
50 GOSUB 27000
```

```
60 PRINT "WHAT ROOM WOULD YOU LIKE DESCRIBED
(1-19)";
65 IF R<1 OR R>19 THEN 30
70 INPUT R
90 GOSUB 700
100 GOTO 60
```

Also, you should add a line 10 that will clear the video display of your computer. If you are using an Apple II series computer, you would write:

```
10 HOME
```

If you are using a Radio Shack or IBM computer, you would write:

```
10 CLS
```

And if you are using a Commodore computer, you would write:

```
10 PRINT CHR$(147);
```

If you are using a computer other than those named above, you will need to consult the computer's manual to learn how to clear the display—or you will simply have to leave the display uncleared.

(If you are using a Radio Shack Model I, III, or 4, add this instruction to line 10:

```
CLEAR 2000
```

to open up storage space for all the text strings we will be handling.)

Once you have all these routines together in a single program, type RUN and press RETURN.

The program will ask "WHAT ROOM WOULD YOU LIKE DESCRIBED (1-19)?" In response, type a number from 1 to 19. The computer will print a description of that room, just as though we were playing the adventure game and had entered that room.

For instance, if you type 15, the computer will print:

YOU ARE ON A WELL-TRAVELED ROAD.

Once you have satisfied yourself that this program works, make sure that you have saved it to disk or tape. Later, we will delete lines 10 through 70, because they are not part of the adventure program. For now, however, we will leave them in the program. With each chapter, we will add more lines to this program—until we have a complete adventure game.

CHAPTER THREE

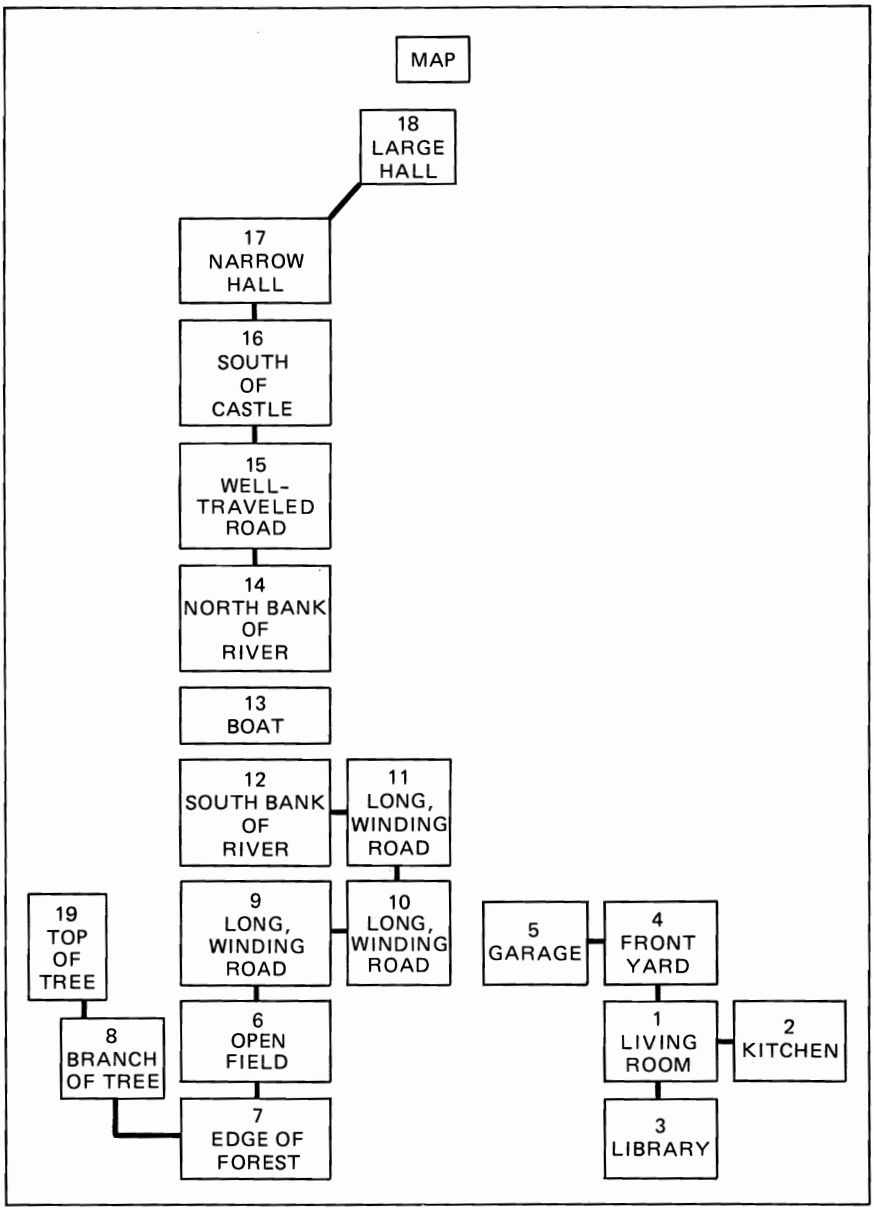
GETTING YOUR BEARINGS

The second step in making a map is to draw it on paper. Get the pencil that we suggested you set aside—and start drawing.

How you draw the map is your own business. We recommend that you draw a separate box for each room in the game. Each box should be big enough that you can write the name and number of the room inside it, but small enough that you can fit lots of them on a piece of paper.

You should also draw lines between the boxes to show paths that lead from one room to another. As on a road map, a line that goes left means that you must go west to get from one room to the next, a line that goes right means that you must go east, a line that goes up means that you must go north, and a line that goes down means that you must go south. How do you represent paths that go up and down between rooms? We recommend that you use diagonal or even curved lines. If the line slants downward toward a room, then you must go down. If the line slants upward, you must go up.

On page 18 you will see a map for our adventure game. Notice that some rooms are not connected to other rooms by lines. The


MAP
18 LARGE HALL
17 NARROW HALL
16 SOUTH OF CASTLE
15 WELL-TRAVELED ROAD
14 NORTH BANK OF RIVER
13 BOAT
12 SOUTH BANK OF RIVER
11 LONG, WINDING ROAD
19 TOP OF TREE
9 LONG, WINDING ROAD
10 LONG, WINDING ROAD
5 GARAGE
4 FRONT YARD
8 BRANCH OF TREE
6 OPEN FIELD
1 LIVING ROOM
2 KITCHEN
7 EDGE OF FOREST
3 LIBRARY

boat (room 13), for instance, has no lines connecting it to other rooms. This is because you can only get to the boat by saying GO BOAT and you can only leave the boat by saying LEAVE BOAT. You cannot reach the boat by going north, south, east, west, up, or down.

In the same way, the garage is not connected to the open field, because you can only get between these two places by mixing the magic formula, not by going in a direction.

The way in which rooms are connected on the map is important. It tells us how we can move around through the imaginary world of the adventure. We must also tell the *computer* about this imaginary world. That is, we must somehow put this map into our program, much like we put the list of room names into the program in the last chapter. This requires that we use a *two-dimensional array.*

A map has two dimensions: length and width. You can move north and south on the map and you can move east and west on it. (On the adventure game map, you can also move up and down, but we'll pretend for now that these fit in with the other two dimensions.)

To describe a map to a computer, we must use a *two-dimensional array.* A two-dimensional array, like the one-dimensional arrays we discussed in the last chapter, can store lists of numbers or strings. Every element in a two-dimensional array has *two* subscripts, instead of just one, like this:

```
AB(15,6)
```

Each subscript represents one of the two dimensions. We will call the first subscript "dimension one" and the second "dimension two." We create two-dimensional arrays with a DIM statement, just as we create one-dimensional arrays, except that we must tell the DIM statement how many elements we will have in *both* dimensions.

For our map, we will use a two-dimensional array called MA.

This will be a numeric array—that is, an array that will store a list of numbers. We can dimension it like so:

```
DIM MA(19,5)
```

This tells the computer that the first subscript can have any value between 0 and 19 and that the second subscript can have any value between 0 and 5. Thus, the total number of elements in this two-dimensional array is 19 times 6, or 114. We won't list all of those elements, but here are a few examples:

```
MA(0,0)
MA(6,3)
MA(14,5)
MA(19,0)
MA(8,2)
MA(3,3)
```

And so forth.

We'll use the first of the two subscripts to represent room numbers. (Once again, the 0 subscript will be ignored, as in the last chapter.) We'll use the second subscript to represent directions.

There are six different directions—north, south, east, west, up, and down—and we will use the numbers 0 to 5 to represent them, like this:

```
0 - North
1 - South
2 - East
3 - West
4 - Up
5 - Down
```

Why are we doing this? We can use the values of the elements in this two-dimensional array to tell the computer whether we can go in a certain direction from a certain room and, if so, where we will end up if we go that way. For instance, suppose that we set this element:

```
MA(3,1)
```

equal to 0 with this statement:

```
MA(3,1)=0
```

This will tell the computer that we cannot go south from the library. How does it tell the computer that? The first subscript, as noted above, is the room number. In this case, it is a 3, which is the library. The second subscript is the direction. In this case, it is a 1, which is south. The fact that this element has been set equal to 0 means "you can't go that way!"

On the other hand, if we set an element equal to a number other than 0, like this:

```
MA(7,0)=6
```

it means that you *can* go in a certain direction. In this case, it means that you can go north (direction 0) from the edge of the forest (room 7). And where do you end up if you go north from the edge of the forest? You end up in room 6, the open field. Which is why we set this element equal to 6.

To sum up, each element of the two-dimensional array, MA, represents a room and a direction, as identified by the two subscripts. If the value of the element is 0, then you cannot go in that direction from that room. If the value of the element is not 0, then it represents the room you will end up in if you go in that direction from that room.

In a few special cases, you might not want the player to move between rooms in an ordinary manner. Perhaps something or someone, such as a dragon or a guard, is blocking the passage. In this case, put a number between 128 and 255 for the value of that direction and room. Remember what number you use. In Chapter Six, we'll show you what to do with it.

At the start of our adventure game program, we must call a subroutine that will fill in all the values of this array, according to the map on page 18. We could design this subroutine as a series of assignment statements, like this:

```
25000 MA(0,0)=4
25010 MA(0,1)=3
25020 MA(0,2)=2
25030 MA(0,3)=0
```

And so forth. But the subroutine would contain 114 separate assignment statements, one for each element of the array, so it would be very large. To save space (and typing), we will use DATA statements instead. We will also need a subroutine to read the numbers out of the DATA statements and into array MA, using the READ command.

For each room on the map of your adventure game, write six numbers on a piece of paper. Each of these numbers should correspond to one of the six map directions—north, south, east, west, up, down—in that order. If you can't move in that direction from the room, write the number 0. If you can move in that direction, write the number of the room that you will end up in. If you can normally move in that direction, but there is a special difficulty (such as a dragon blocking the way), use a number between 128 and 255. Since it is unlikely that you will ever have more than 127 rooms in a game, this number will not be mistaken for an ordinary room number.

For room 1 in the map on page 18, you would write:

4,3,2,0,0,0 (room 1)

This indicates that if you go north from room 1, you will end up in room 4 (the front yard). If you go south, you will end up in room 3 (the library). If you go east, you will end up in room 2 (the kitchen). You can't go west, up, or down. Notice that we have given the number of the room in parentheses, so that we won't forget which room it is.

These are the *map numbers* for the game. A complete list of map numbers for our game looks like this:

RM #	Name	N	S	E	W	U	D
1.	Living Room	4	3	2	0	0	0
2.	Kitchen	0	0	0	1	0	0
3.	Library	1	0	0	0	0	0
4.	Front Yard	0	1	0	5	0	0
5.	Garage	0	0	4	0	0	0
6.	Open Field	9	7	0	0	0	0
7.	Edge of Forest	6	0	0	0	0	0
8.	Branch of Tree	0	0	0	0	0	7
9.	Long, Winding Road (1)	0	6	10	0	0	0
10.	Long, Winding Road (2)	11	0	0	9	0	0
11.	Long, Winding Road (3)	0	10	0	12	0	0
12.	South Bank of River	0	0	11	0	0	0
13.	Boat	0	0	0	0	0	0
14.	North Bank of River	15	0	0	0	0	0
15.	Well-Traveled Road	16	14	0	0	0	0
16.	South of Castle	128	15	0	0	0	0
17.	Narrow Hall	0	0	0	0	18	0
18.	Large Hall	0	0	0	0	0	17
19.	Top of Tree	0	0	0	0	0	8

Compare these map numbers to the map on page 18. Make sure that you understand how that map was used to create these numbers, using the instructions above. Notice that "SOUTH OF CASTLE" includes the number 128 in its list, for the first direction (north). This means that there is something unusual about this direction. In this case, it means that there is a guard blocking the castle door to the north.

Once you have the list of map numbers for your adventure, you must turn them into DATA statements. To do this, you must add a BASIC line number plus the word "DATA" in front of each list of six numbers, and put the name of the room into a REM statement at the end of the line, separated from the DATA statement by a colon (:). Here are the DATA statements for the above map numbers.

```
25100 DATA 4,3,2,0,0,0 : REM LIVING ROOM
25110 DATA 0,0,0,1,0,0 : REM KITCHEN
25120 DATA 1,0,0,0,0,0 : REM LIBRARY
25130 DATA 0,1,0,5,0,0 : REM FRONT YARD
25140 DATA 0,0,4,0,0,0 : REM GARAGE
25150 DATA 9,7,0,0,0,0 : REM OPEN FIELD
25160 DATA 6,0,0,0,0,0 : REM EDGE OF FOREST
25170 DATA 0,0,0,0,0,7 : REM BRANCH OF TREE
25180 DATA 0,6,10,0,0,0 : REM LONG, WINDING ROAD (1)
25190 DATA 11,0,0,9,0,0 : REM LONG, WINDING ROAD (2)
25200 DATA 0,10,0,12,0,0 : REM LONG, WINDING ROAD (3)
25210 DATA 0,0,11,0,0,0 : REM SOUTH BANK OF RIVER
25220 DATA 0,0,0,0,0,0 : REM BOAT
25230 DATA 15,0,0,0,0,0 : REM NORTH BANK OF RIVER
25240 DATA 16,14,0,0,0,0 : REM WELL-TRAVELED ROAD
25250 DATA 128,15,0,0,0,0 : REM SOUTH OF CASTLE
25260 DATA 0,0,0,0,18,0 : REM NARROW HALL
25270 DATA 0,0,0,0,0,17 : REM LARGE HALL
25280 DATA 0,0,0,0,0,8 : REM TOP OF TREE
```

You should now type these lines and add them to our adventure program. Save the result to tape or disk.

You will also need a subroutine that will read this data into the map array, MA. Here it is:

```
24991 REM The following routine reads the map data into
24992 REM the map array, MA(ROOM, DIRECTION).
24993 REM
25000 IF NR=0 THEN RETURN
25010 DI$(0)="NORTH":DI$(1)="SOUTH":DI$(2)="EAST"
25020 DI$(3)="WEST":DI$(4)="UP":DI$(5)="DOWN"
25030 FOR I=1 TO NR
25040 FOR J=0 TO ND-1
25050 READ MA(I,J)
25060 NEXT J
25070 NEXT I
25080 RETURN
```

Add this subroutine to the adventure game program. You can use this subroutine exactly in this form in an adventure program of your own. Earlier in the program, you must set the variable NR equal to the number of rooms in the game with a statement like this: NR=19. You must also call this subroutine with the statement GOSUB 25000.

This subroutine also creates a string array called DI$, which has six elements, each of which is set equal to the name of one of the six directions. This array must be dimensioned at the beginning of the program with the statement

```
DIM DI$(6)
```

Notice that we have numbered the DATA statements so that they will come right after this subroutine in the program. This helps us keep track of the order of these statements. Later, we will

put other subroutines into the program that will read DATA statements. It is *very* important that we execute these subroutines in order according to line number, because DATA statements must always be READ in the order that they appear in the program. (But you knew that, didn't you?)

Finally, we will need a subroutine that prints out a list of the directions that you can go from the current room—that is, the room with its number in variable R. Here is that subroutine:

```
490 REM *** DIRECTIONS
495 REM
500 PRINT "YOU CAN GO: ";
510 FOR I=0 TO 5
520 IF MA(R,I)>0 THEN PRINT DI$(I) ;" ";
530 NEXT I
540 PRINT
550 RETURN
```

Add this subroutine to our adventure program. To see how our most recent subroutines work, make the following changes to the program that you have put together so far:

```
20 NR=19 : ND=6 : REM NUMBER OF ROOMS AND
DIRECTIONS
30 DIM R$(NR),DI$(6)
50 GOSUB 25000 : GOSUB 27000
90 GOSUB 700 : GOSUB 500
```

When you have all of this typed, RUN the program. Once again, it will ask you "WHAT ROOM WOULD YOU LIKE DESCRIBED (1-19)?" Type a room number from 1 to 19. The program will not only print the description of the room, but it will print "YOU CAN GO," followed by a list of the directions in which you can go from that room. For instance, if you type 10, it will print:

YOU ARE ON A LONG, WINDING ROAD
YOU CAN GO: NORTH WEST

We now have our map in the computer. The next step is to fill that map with objects, such as swords and magic fans, that we can actually pick up and use.

CHAPTER FOUR

FILLING THE WORLD

Putting objects in our imaginary world, such as swords and books and magic formulas, is easy. First, we must create three new arrays, which we will call OB$, O2$, and OB.

OB$ is a string array. We will use it to store the names of the objects.

O2$ is a second string array. We will use it to store a three-letter "tag" by which the computer can identify each object. More about the tag in a moment.

OB is a numeric array. It will identify the number of the room in which each object can be found.

These three arrays must be dimensioned with the statement

```
DIM OB$(NO),O2$(NO),OB(NO)
```

where the variable NO is equal to the number of objects in the game.

To create these arrays, you must first make a list of all the objects in the game. In this list, you should first identify each object by name. After the name, write the three-letter tag. The tag should consist of the first three letters in the name of the object.

For instance, the tag for a BOOK would be BOO. The tag for a SWORD would be SWO. Don't include any adjectives in this tag. For instance, if the object is called a GOLDEN SWORD, the tag would still be SWO, not GOL. If the book is called an ANCIENT BOOK, the tag would still be BOO, not ANC. Each of these tags must be unique—that is, you can't have two objects with the same tag, even if this means changing the name of an object. For instance, if there is a CASE and a CASTLE in the adventure, they would both have the tag CAS. Therefore you would have to change the CASTLE to a PALACE or the CASE to a BOX.

Finally, write the number of the room in which the object can be found at the beginning of the game. For instance, if the object is in room 2 (the kitchen), then write the number 2 after the name and the tag. If the object is not visible at the beginning of the game, write a 0 after the name and tag. If the object is in the player's inventory, write the number −1 after the tag. (In case you haven't taken any courses in math, this number is "negative one," and is actually lower than 0. The computer will use this number to recognize which objects are in the player's inventory.)

We need a special way to indicate objects that cannot be picked up and put into the player's inventory. We indicate these objects by adding 128 to the number of the room in which the object can be found. For instance, if the room number of the object is 14 and the object cannot be picked up, then you would give the object a room number of 142, which is 14 plus 128. This trick lets the computer know when it should not let the player pick up an object.

A list of objects for our adventure game looks like this:

OBJ #	Name	Tag	Room
1.	An old diary	DIA	1
2.	A small box	BOX	1

3.	Cabinet	CAB	130
4.	A salt shaker	SAL	0
5.	A dictionary	DIC	3
6.	Wooden barrel	BAR	133
7.	A small bottle	BOT	0
8.	A ladder	LAD	4
9.	A shovel	SHO	5
10.	A tree	TRE	135
11.	A golden sword	SWO	0
12.	A wooden boat	BOA	140
13.	A magic fan	FAN	8
14.	A nasty-looking guard	GUA	144
15.	A glass case	CAS	146
16.	A glowing ruby	RUB	0
17.	A pair of rubber gloves	GLO	19

Six objects in this list—the cabinet, the wooden barrel, the tree, the wooden boat, the nasty-looking guard, and the glass case—have room numbers larger than 127. This means that these objects cannot be picked up. The guard, for instance, has a room number of 144. If we subtract 128 from this number, we learn that the guard is in room 16. In the same way, we can tell that the cabinet is in room 2.

For the sake of the program, we must change this list of objects into a series of DATA statements. We do this the same way we turned the map numbers into DATA statements: by adding line numbers and the word DATA in front of each object name. Here is a series of DATA statements made from this list of objects:

```
26100 DATA AN OLD DIARY, DIA, 1 : REM OBJECT #0
26110 DATA A SMALL BOX, BOX, 1 : REM OBJECT #1
26120 DATA CABINET, CAB, 130 : REM OBJECT #2
26130 DATA A SALT SHAKER, SAL, 0 : REM OBJECT #3
26140 DATA A DICTIONARY, DIC, 3 : REM OBJECT #4
26150 DATA WOODEN BARREL, BAR, 133 : REM OBJECT #5
```

```
26160 DATA A SMALL BOTTLE, BOT, 0 : REM OBJECT #6
26170 DATA A LADDER, LAD, 4 : REM OBJECT #7
26180 DATA A SHOVEL, SHO, 5 : REM OBJECT #8
26190 DATA A TREE, TRE, 135 : REM OBJECT #9
26200 DATA A GOLDEN SWORD, SWO, 0 : REM OBJECT
#10
26210 DATA A WOODEN BOAT, BOA, 140 : REM OBJECT
#11
26220 DATA A MAGIC FAN, FAN, 8 : REM OBJECT #12
26230 DATA A NASTY-LOOKING GUARD, GUA, 144 : REM
OBJECT #13
26240 DATA A GLASS CASE, CAS, 146 : REM OBJECT #14
26250 DATA A GLOWING RUBY, RUB, 0 : REM OBJECT #15
26260 DATA A PAIR OF RUBBER GLOVES, GLO, 19 : REM
OBJECT #17
```

At the end of each line, we have included an object number in a REM statement. These numbers will come in handy later.

Next, we need a subroutine that will read these data items into arrays OB$, O2$, and OB. Here is that subroutine:

```
25910 REM The following subroutine reads the object data
25920 REM into the three object arrays, OB(X), OB$(X), and
25930 REM O2$(X).
25940 REM
26000 IF NO=0 THEN RETURN
26010 FOR I=0 TO NO-1
26020 READ OB$(I),O2$(I),OB(I)
26030 NEXT I
26040 RETURN
```

Before calling this subroutine, you must set variable NO equal to the number of objects in the game, like this: NO=18. (Notice, incidentally, that the first object is object #0. If you wish, you may start with object #1, just as we did with the numbers of the

rooms.) We can execute this subroutine with the statement GOSUB 26000. The DATA statements holding the object DATA should come right after this routine.

Next, we need a subroutine that will print out all of the objects in the current room. Here is that subroutine:

```
600 PRINT "YOU CAN SEE: "
610 FL=0:FOR I=0 TO NO-1
620 IF (OB(I) AND 127)=R THEN PRINT " ";OB$(I) : FL=1
630 NEXT I
640 IF FL=0 THEN PRINT " NOTHING OF INTEREST"
650 RETURN
```

Add all of these DATA statements and subroutines to our rapidly growing program. Then make the following changes and additions:

```
20 NR=19 : NO=18 : ND=6 : REM NUMBER OF ROOMS,
OBJECTS, AND DIRECTIONS
30 DIM R$(NR),DI$(ND),OB$(NO),O2$(NO),OB(NO)
50 GOSUB 25000 : GOSUB 26000 : GOSUB 27000
90 GOSUB 700 : GOSUB 500 : GOSUB 600
```

Run the program. Once again, you will be asked what room you would like to have described. Answer with a number between 1 and 19. The program will print a description of the room, a list of the directions in which you can go from that room, and a list of the objects visible in that room. For instance, if you type a number 1, it will print:

```
YOU ARE IN YOUR LIVING ROOM
YOU CAN GO: NORTH  SOUTH  EAST
YOU CAN SEE:
      AN OLD DIARY
      A SMALL BOX
```

If there are no objects in a room, it will print:

```
YOU CAN SEE:
      NOTHING OF INTEREST
```

And that pretty much completes our adventure program's ability to describe a room. It will now print the description of the room, the directions in which we can go from that room, and a list of objects visible in that room.

Next, we need to give commands to the game. This means that our adventure program must actually be able to understand what we have typed, in a limited sense. It can do this with the aid of a very special program routine called a *parser*.

CHAPTER FIVE

THE PARSER

Computers don't understand the English language. In fact, the only language that computers understand is machine language, which is nothing more than a bunch of electronic bloops and bleeps (or would be, if you could hear it aloud).

Yet in an adventure game the player must give commands to the computer using English words. How is this possible?

Actually, it is a trick. The adventure program doesn't really understand English. However, we can teach it to look for certain combinations of letters and words, and to do certain things when it finds those combinations, provided all other conditions are in agreement.

Some adventure games understand more combinations of words than others. A lot of the adventures you can buy in stores will accept commands that resemble complete English sentences, though even these games don't really understand English. They just understand certain kinds of sentences—and they don't really "understand" these sentences in the way that a human being would.

In our game, we only let the player create two-word commands. Although this limits the number and type of commands

that the player can use, it makes the program a lot easier to write. And it doesn't make the playing of the game as difficult as you might think. Most of the early adventure game programs, such as the Original Adventure and the Scott Adams adventures, required two-word commands.

The first word in a command must always be a verb—that is, a word that tells the program what to do. The second word must be a noun—that is, a word that tells the program what to do it to. A very few commands are made up of one word, and this word should always be a verb. INVENTORY, for instance, is a one-word command.

When the player types a command, the program must analyze the command to figure out what the player is asking it to do. The first step in this analysis is to break the command into two parts, the verb and the noun. This is done by a routine called a *parser*. The parser that we will use in this book is printed below. It is a complicated routine, and we will not explain how it works in detail. However, you can easily use it in your adventure programs if you follow some simple rules, which we will explain immediately after the parser. Here is the routine:

```
92 REM *** PARSER
99 REM
100 PRINT : CM$="" : INPUT "WHAT NOW";CM$ : IF CM$="" THEN 100
110 C=0 : V$="" : N$=""
120 C=C+1 : IF C>LEN(CM$) THEN 150
130 W$=MID$(CM$,C,1) : IF W$=" " THEN 150
140 V$=V$+W$ : GOTO 120
150 C=C+1 : IF C>LEN(CM$) THEN 180
160 W$=MID$(CM$,C,1) : IF W$=" " THEN 180
170 N$=N$+W$ : GOTO 150
180 IF V$="" THEN 100
190 IF LEN(V$)>3 THEN V$=LEFT$(V$,3)
200 IF LEN(N$)>3 THEN N$=LEFT$(N$,3)
```

Notice that this routine is *not* a subroutine. It does not end with the word "RETURN." We do not call it with the word "GOSUB." We simply plunk it into the program near the beginning, where it will be executed before each move of the game.

What does it do? First, it prints the words "WHAT NOW?" on the video display. Then, it waits for the player to type a command. When the command has been typed, it breaks it into a verb and a noun. For instance, if you type the command DROP SWORD, it will break it into the verb "DROP" and the noun "SWORD." When the parser is finished, the verb will be stored in the string variable V$. The noun will be stored in the string variable N$.

Well, actually only the first three letters of the verb and noun will be stored. This is generally enough to recognize the words by, and will make the rest of the program a little easier to write. Therefore, the command DROP SWORD will actually become the verb "DRO" and the noun "SWO."

To see the parser in action, add it to our growing adventure program. Then delete these lines from the program:

```
60, 65, 70
```

We don't need these lines anymore. They'll only get in the way.

Finally, add these lines:

```
60 R=1
210 PRINT "V$ = ";V$
220 PRINT "N$ = ";N$
230 GOTO 100
```

Line 60 establishes that we are in room 1. When you run this program, it will print out the description of the first room, just as though we were actually starting the game.

Run the program. It will print the words "WHAT NOW?" on

the display. Type a two-word command. It will show you the three-letter verb and noun stored in V$ and N$.

For instance, if you type "FIGHT DRAGON," this program will print:

```
V$ = FIG
N$ = DRA
```

If one of the two words is three letters or less in length, the parser will set N$ or V$ equal to the entire word. If you type GO NORTH, then this program will print:

```
V$ = GO
N$ = NOR
```

If you type more than two words, the extra words will be ignored. If you type GET SALT SHAKER, this program will print:

```
V$ = GET
N$ = SAL
```

If you only type one word, N$ will be set equal to the *null string*. This is the string without any characters in it at all. We represent it symbolically as "". We can check to see if N$ is equal to the null string like this:

```
IF N$="" THEN [rest of statement]
```

If you type INVENTORY, this program will print:

```
V$ = INV
N$ =
```

Now that we can talk to our adventure program, what should we say? All sorts of things—as we shall see in the next chapter.

CHAPTER SIX

VERBS AND NOUNS

The largest parts of an adventure game program are the *verb routines*. These are the parts of the program that check to see what verb the player typed, then take appropriate action.

There must be a verb routine in the program for every verb that the program will understand. In fact, every time that we think of a new verb that we want the program to understand, we must write a new verb routine to go with it. In almost every game, we will want to include verb routines for GO, GET, DROP, INVENTORY, EXAMINE, and LOOK.

Most of the verb routines will be written in exactly the same manner. The first line of the routine checks to see if V$ is equal to the first three letters of a certain verb. If not, the program jumps to the next verb routine, which checks for a different verb.

If no verb routine recognizes V$, the program prints the words

"I DON'T KNOW HOW TO DO THAT"

on the video display, and jumps back to the parser on line 100.

The verb routine then checks to see what noun is stored in N$. If the noun in N$ doesn't make sense when used with the verb in

V$, the program prints an error message, something like "YOU CAN'T DO THAT WITH THAT!"

On the other hand, if the noun goes with the verb, the verb routine must then check several other things. In some cases, the player must be in the right room for performing the action. Any necessary objects must be in the same room or in the player's inventory, and so forth. The exact process depends on the particular verb.

Almost every verb routine ends with the instruction GOTO 100. This takes the program back to the first line of the parser, which asks for a new command. A few verb routines end with the line GOTO 90. This takes the program back to line 90, which is just before the beginning of the parser. Line 90 will print out a full description of the current room. This line is useful if the player has just moved to a new room, or has asked to LOOK at the current room. As soon as this description is printed, the parser will ask for a new command.

Most of the remainder of this book will be concerned with verb routines. We will give you several verb routines in this chapter. These routines are for verbs such as "GO" and "GET" that are common to all adventure games. You may use these routines in your own adventure programs. (They are included in the skeleton program at the end of the book.) In the next chapter, we will write some verb routines that are special to this program. You may want to rewrite some of these routines so that they will work in your own games.

For instance, here is a routine for the verb "GO":

```
1998 REM ROUTINE FOR THE VERB 'GO.'
1999 REM
2000 IF V$<>"GO" THEN 2500
2010 IF N$="NOR" THEN DI=0 : GOTO 2400
2020 IF N$="SOU" THEN DI=1 : GOTO 2400
2030 IF N$="EAS" THEN DI=2 : GOTO 2400
```

```
2040 IF N$="WES" THEN DI=3 : GOTO 2400
2050 IF N$="UP" THEN DI=4 : GOTO 2400
2060 IF N$="DOW" THEN DI=5 : GOTO 2400
2070 REM PUT ADDITIONAL 'GO' DIRECTIONS HERE.
2390 PRINT "YOU CAN'T GO THERE!" : GOTO 100
2399 REM
2400 IF MA(R,DI)>0 AND MA(R,DI)<128 THEN R=MA(R,DI) :
GOTO 90
2410 REM CHECK FOR SPECIAL (GREATER THAN 128)
DIRECTIONS
2480 GOTO 2390
```

The first line (2000) of the GO routine checks to see if the verb (V$) is "GO." If not, it jumps to the next verb routine, which will begin on line 2500. If the verb is "GO," it checks to see if the noun (N$) is one of the four directions: north, south, east, west, up, or down. If so, it assigns a value between 0 and 5 to variable DI, then jumps to the routine on line 2400. This routine checks the element of the map array, MA, that is equal to the current room and the requested direction. If this element equals 0, it does nothing. You can't move that way. If it is greater than 127, it does nothing. Otherwise, it changes variable R to the value of the new room, and jumps back to line 90, which prints a description of this room before executing the parser.

If it turns out that the requested move is illegal, the message "YOU CAN'T GO THERE!" is printed. Notice, however, that there is a lot of blank space left in this routine, where you can add instructions of your own. Lines 2070 through 2380 are reserved for routines that check for unusual directions that the player can go, such as GO BUILDING or GO HOLE. There is one such direction in our adventure game: GO BOAT. We can add a short routine that checks for this command at line 2070, like this:

```
2070 IF N$="BOA" AND OB(11)=R+128 THEN R=13 : GOTO
90
```

The BOAT is considered by the program to be both an object and a room. (This is allowed!) It is room 13 and object 11. Line 2070 checks to see if the noun (N$) is equal to "BOA" and if OB(11) is present in the current room (R). (It adds 128 to the room number because the boat is an object that cannot be picked up and therefore has 128 added to its room number to indicate this.) If all of these conditions are true, the room number is changed to 13 (the boat) and the program jumps back to line 90, which prints a description of room 13.

Lines 2410 to 2470 are reserved for map elements with numbers larger than 128. Since there is only one of these in our program, we can put it on line 2410:

```
2410 IF MA(R,DI)=128 THEN PRINT "THE GUARD WON'T LET YOU!" : GOTO 100
```

If the map element is equal to 128, then we know that we are at the entrance to the castle, which is blocked by the guard. (We set this up back in Chapter Four, remember?) The message "THE GUARD WON'T LET YOU!" is printed, and the program jumps back to the parser.

Let's try out this routine. Add all of the lines above relating to the GO routine to our program. Then add the following line:

```
480 GOTO 2000
```

This line follows the parser. It jumps to the first of the verb routines—the GO routine. (The lines between this line and the end of the parser are reserved for any special routines that need to be performed before the verb routines are executed.)

Also, add this line:

```
2500 REM
```

This holds the place of the next verb routine, until we get around to writing it.

Run the program. You will now be able to move around among the first five rooms of the adventure, with the commands GO NORTH, GO SOUTH, GO EAST, and GO WEST. You cannot move beyond room 5, however, without first mixing the magic formula, and you cannot do this yet.

Another very important verb is "GET." Before we create a GET routine, however, we need a subroutine that will check a room to see if a certain object exists in the game. (We will put this in a subroutine because it will be used by other verb routines.) Here is that subroutine:

```
1000 IF NO=0 THEN RETURN
1010 FOR I=0 TO NO-1
1020 IF O2$(I)=N$ THEN FL=1 : RO=OB(I) : GOTO 1050
1030 NEXT I
1040 FL=0 : RETURN
1050 RO=OB(I) : IF RO>127 THEN RO=RO-128
1060 RETURN
```

This subroutine does several things for us. The key is a special variable called FL. The name is short for "flag," because it is a *flag variable*. A flag variable is a variable that carries information about the results of a routine. In this case, FL tells us whether an object exists in the game. When this subroutine executes, it will give FL one of two values. If FL equals 0, then there is no such object in the game. If FL=1 then there is such an object in the game. In this case, the variable I is equal to the number of the object in the object list. The variable RO is set equal to the number of the room where the object can be found. (If 128 was added to the room number of the object to indicate that it can't be picked up, this subroutine automatically subtracts 128 from RO, so that it

represents the actual room number.) If RO equals −1, then the object is in the player's inventory.

Here is the GET routine:

```
2499 REM *** 'GET' ROUTINE
2500 IF V$ <> "GET" AND V$ <>"TAK" THEN 2600
2510 GOSUB 1000
2520 IF FL=0 THEN PRINT "YOU CAN'T GET THAT!" : GOTO
100
2530 IF RO=−1 THEN PRINT "YOU ALREADY HAVE IT!" :
GOTO 100
2540 IF OB(I)>127 THEN PRINT "YOU CAN'T GET THAT!" :
GOTO 100
2550 IF RO<>R THEN PRINT "THAT'S NOT HERE!" : GOTO
100
2570 IF IN>NI THEN PRINT "YOU CAN'T CARRY ANY MORE."
: GOTO 100
2580 IN=IN+1 : OB(I)=−1 : PRINT "TAKEN." : GOTO 100
```

This routine is written in such a way (see line 2500) that it will also accept the verb "TAKE" as a synonym for "GET." It is a good idea to include synonyms for most of the major verbs in your program, so that the player can phrase important sentences in more than one way.

Before this routine can be used, you must create a variable called NI and set it equal to the maximum number of objects that the player can carry at one time in his or her inventory. You must also create a variable called IN, which should initially be set equal to 0. This variable will always be equal to the current number of objects in the player's inventory.

The first line (2500) checks to make sure that the verb is "GET." If not, it skips to the next verb routine, at line 2600. Then it calls the subroutine at 1000, to see if the noun represents one of the objects in the game.

Line 2520 checks to see if the object exists. If not, it prints "YOU CAN'T GET THAT!" and jumps back to the parser.

Line 2530 checks to see if the object is in the player's inventory. If so, it prints "YOU ALREADY HAVE IT!" and jumps back to the parser.

Line 2540 checks to see if the object can be picked up. If not (that is, if its room number is greater than 127), it says "YOU CAN'T GET THAT!" and jumps back to the parser.

Line 2550 checks to see if the object is in the current room. If not, it prints "THAT'S NOT HERE!" and jumps back to the parser.

Line 2570 checks to see if the player's inventory is full. If so, it prints "YOU CAN'T CARRY ANY MORE!" and jumps back to the parser.

If everything checks out properly, line 2580 increases the value of IN. Then it sets the room number for the object (which is in variable OB(I)) equal to −1. The −1 means that the object is now in the player's inventory.

There is a special line that we are going to put into this GET routine. Since the game is over when the player GETs the magic jewel, we need to check to see if the noun is RUB (short for RUBY) and if the player successfully GETs it. We can do this by adding this line:

```
2575 IF R=18 AND N$="RUB" THEN PRINT "CONGRATULA
TIONS! YOU'VE WON!" : GOTO 3430
```

The routine on line 3430, which we will write in the next chapter, asks the player if he or she wants to play again.

We also need a DROP routine, to get rid of objects that we have picked up and don't want to carry anymore. Here is the DROP routine:

```
2599 REM *** 'DROP' ROUTINE
2600 IF V$<>"DRO" AND V$<>"THR" THEN 2700
2610 GOSUB 1000
2620 IF FL=0 OR RO<>-1 THEN PRINT "YOU DON'T HAVE
THAT!" : GOTO 100
2650 IN=IN-1 : OB(I)=-1 : PRINT "DROPPED." : GOTO
100
```

After making sure that the verb is "DROP" (or "THROW," which is usually allowed in adventures as a synonym for "DROP"), this routine calls the subroutine at 1000 to see if the noun is a real object. If it isn't a real object (in the game) or if it is but the player doesn't have it, the routine prints "YOU DON'T HAVE THAT!" and jumps back to the parser.

Otherwise, it subtracts 1 from IN and sets OB(I) (the room number of the object) equal to the current room number. Now, the object will appear in the description of the current room.

Finally, we need an INVENTORY routine. Here it is:

```
2699 REM *** 'INVENTORY' ROUTINE
2700 IF V$ <>"INV" AND V$ <>"I" THEN 2800
2710 FL=0 : PRINT "YOU ARE CARRYING:"
2720 FOR I=0 TO NO-1
2730 IF OB(I)=-1 THEN PRINT " ";OB$(I) : FL=1
2740 NEXT I
2750 IF FL=0 THEN PRINT " NOTHING"
2760 GOTO 100
```

This is a fairly simple routine. It prints the words "YOU ARE CARRYING:", then checks the room number of every object in the object list. If the room number of an object is -1, then it prints the name of that object. If no object has a room number of -1, it prints "NOTHING."

To use these last three routines, add them to our program (starting with the subroutine at line 1000) and then make the following change:

```
60 R=1 : IN=0 : NI=0
```

Now you can use the words "GET," "DROP," and "INVENTORY," as you wander around among the first five rooms using the GO command.

These routines can be included, in pretty much the same form, in all adventure games. In the next chapter, we'll give you some verb routines that are unique to this program.

CHAPTER SEVEN

VERBS, VERBS, AND MORE VERBS

For every verb that the player is able to use in playing the game, there must be a verb routine in the finished program. In this chapter, we present the verb routines that will make our game unique. The routines in the last chapter were essential; the routines in this chapter are, we hope, fun.

The "EXAMINE" verb, for instance, is used to take a closer look at objects, to learn details that might be essential in completing the quest.

An important synonym for "EXAMINE" is "LOOK." Instead of "EXAMINE BOTTLE," the player may say "LOOK BOTTLE"—short for "look at bottle." However, the verb "LOOK" has a special meaning if used without a noun. It tells the program to print out a description of the current room. Therefore the EXAMINE routine, printed below, also contains a special routine for the verb "LOOK." If the verb "LOOK" is used by itself, it jumps back to line 90, which prints a description of the current room. If "LOOK" is used with a noun, it jumps to the EXAMINE routine. ("LOOK" can be abbreviated as L.)

Here is the EXAMINE/LOOK routine:

```
2799 REM *** 'LOOK' ROUTINE
2800 IF V$<>"LOO" AND V$<>"L" THEN 2900
2810 IF N$<>"" THEN 2910
2820 GOTO 90
2899 REM *** 'EXAMINE' ROUTINE
2900 IF V$<>"EXA" THEN 3400
2910 IF N$<>"GRO" THEN 2940
2920 IF R<>6 THEN PRINT "IT LOOKS LIKE GROUND!" :
GOTO 100
2930 PRINT "IT LOOKS LIKE SOMETHING'S BURIED HERE." :
GOTO 100
2940 REM
3000 GOSUB 1000
3010 IF RO<>R AND RO<>-1 THEN PRINT "IT'S NOT
HERE!" : GOTO 100
3020 IF N$="BOT" THEN PRINT "THERE'S SOMETHING
WRITTEN ON IT!" : GOTO 100
3030 IF N$="CAS" THEN PRINT "THERE'S A JEWEL INSIDE!"
: GOTO 100
3040 IF N$="BAR" THEN PRINT "IT'S FILLED WITH RAIN
WATER." : GOTO 100
3390 PRINT"YOU SEE NOTHING UNUSUAL." : GOTO 100
```

In line 3000, this routine looks to see if the object to be examined is present, by calling the subroutine at 1000. If the object is not present, it responds "IT'S NOT HERE!"

Sometimes, the player may need to examine something that is not in the object list. You must check for this *before* line 3000. (The lines after line 2900 and before 3000 are reserved for this.) In the routine above, the player can say "EXAMINE GROUND" while standing in the open field (room 6). We check for this in lines 2910–2930.

Another important verb is "QUI," which lets the player end the game early. The QUIT routine first asks the player if he or she really wants to quit, in case the command was typed accidentally.

QUIT" requires no noun. Here is the QUIT routine:

```
3400 IF V$<>"QUI" THEN 3500
3410 PRINT "ARE YOU SURE YOU WANT TO QUIT (Y/N)"; :
INPUT QU$
3420 IF QU$="N" THEN GOTO 100
3430 PRINT "WOULD YOU LIKE TO PLAY AGAIN (Y/N)"; :
INPUT QU$
3440 IF QU$="Y" THEN RUN
3450 IF QU$="N" THEN END
3460 GOTO 3430
```

Sometimes the player will need to read something, either a book or an object (such as the bottle of formula). Here is a routine for the "READ" verb:

```
3500 IF V$<>"REA" THEN 3700
3510 IF N$<>"DIA" THEN 3560
3520 IF OB(0)<>R AND OB(0)<>-1 THEN PRINT "THERE'S
NO DIARY HERE!" : GOTO 100
3530 PRINT "IT SAYS: 'ADD SODIUM CHLORIDE PLUS THE"
3540 PRINT "FORMULA TO RAINWATER, TO REACH THE"
3550 PRINT "OTHER WORLD.' " : GOTO 100
3560 IF N$<>"DIC" THEN 3590
3570 IF OB(4)<>R AND OB(4)<>-1 THEN PRINT "YOU
DON'T SEE A DICTIONARY!" : GOTO 100
3580 PRINT "IT SAYS: SODIUM CHLORIDE IS" : PRINT "COM
MON TABLE SALT." : GOTO 100
3590 IF N$ <>"BOT" THEN 3620
3600 IF OB(6) <>R AND OB(6)<>-1 THEN PRINT "THERE'S
NO BOTTLE HERE!" : GOTO 100
3610 PRINT "IT READS: 'SECRET FORMULA'." : GOTO 100
3620 REM
3690 PRINT "YOU CAN'T READ THAT!" : GOTO 100
```

Like most of the routines that follow, this routine checks to make sure that the object to be read is either in the same room with the player or in the player's inventory. We can check if an object is in the same room as the player by writing:

```
IF OB(I)=R THEN [do this]
```

where I is the number of the object in the object list and R is the current room. We can check if the object is in the player's inventory with the statement:

```
IF OB(I)=-1 THEN [do this]
```

We can combine these two checks in the statement:

```
IF OB(I)=R OR OB(I)=-1 THEN [do this]
```

Usually, however, we do it the other way around:

```
IF OB(I)<>R AND OB(I)<>-1 THEN [give error message]
```

This says: If the object *isn't* in the same room or in the player's inventory, a mistake has been made.

A few objects in this game, such as the BOX and the CASE, can be OPENed. Here is the routine for the "OPEN" verb:

```
3700 IF V$<>"OPE" THEN 3900
3710 IF N$<>"BOX" THEN 3740
3720 IF OB(1)<>R AND OB(1)<>-1 THEN PRINT "THERE'S
NO BOX HERE!" : GOTO 100
3730 OB(6)=R : PRINT "SOMETHING FELL OUT!" : GOTO
100
3740 IF N$<>"CAB" THEN 3770
3750 IF R<>2 THEN PRINT "THERE'S NO CABINET HERE!" :
GOTO 100
```

```
3760 PRINT "THERE'S SOMETHING INSIDE!" : OB(3)=2 :
GOTO 100
3770 IF N$<>"CAS" THEN 3820
3780 IF R<>18 THEN PRINT "THERE'S NO CASE HERE!" :
GOTO 100
3790 IF GF<>1 THEN PRINT "THE CASE IS ELECTRIFIED!" :
GOTO 100
3800 PRINT "THE GLOVES INSULATE AGAINST THE"
3810 PRINT "ELECTRICITY! THE CASE OPENS!"
3820 OB(15)=18 : GOTO 100
3890 PRINT "YOU CAN'T OPEN THAT!" : GOTO 100
```

This routine includes a special variable called GF, short for "glove flag." This variable tells us whether the player is wearing the gloves. If GF equals 0, the player is not wearing the rubber gloves and cannot open the electrified case. If GF equals 1, then the player is wearing the gloves and can open the case.

To reach the magic land, it is necessary to POUR the salt and the formula into the rainwater. Here is the POUR routine:

```
3900 IF V$<>"POU" THEN 4100
3910 IF N$<>"SAL" THEN 3960
3920 IF OB(3)<>R AND OB(3)<>-1 THEN PRINT "YOU
DON'T HAVE THE SALT!" : GOTO 100
3930 IF SF=1 THEN PRINT "THE SHAKER IS EMPTY!" : GOTO
100
3940 IF R=5 THEN MX=MX+1
3950 SF=1 : PRINT "POURED!" : GOTO 4010
3960 IF N$<>"BOT" THEN PRINT "YOU CAN'T POUR THAT!"
: GOTO 100
3970 IF OB(6)<>R AND OB(3)<>-1 THEN PRINT "YOU
DON'T HAVE THE BOTTLE!" : GOTO 100
3980 IF FF=1 THEN PRINT "THE BOTTLE IS EMPTY!" : GOTO
100
3990 IF R=5 THEN MX=MX+1
```

```
4000 FF=1 : PRINT "POURED!"
4010 IF MX<3 THEN 100
4020 PRINT "THERE IS AN EXPLOSION!"
4030 PRINT "EVERYTHING GOES BLACK!"
4040 PRINT "SUDDENLY YOU ARE. . ."
4050 PRINT ". . . SOMEWHERE ELSE!"
4060 R=6 : GOTO 90
```

This routine uses three special variables: SF (short for "salt flag"), FF ("formula flag"), and MX ("mixture"). SF tells us if the salt has been poured yet. If SF equals 0 the salt has not been poured; if it equals 1 then it has been. In the same way, FF tells us if the formula has been poured. MX tells us how many ingredients are in the magic mixture. To reach the magic land, all three ingredients—rainwater, salt, and formula—must be in the barrel. When the game starts, MX equals 1, because one ingredient, the rainwater, is in the barrel. For every additional ingredient poured into the barrel, the value of MX is increased by 1. When it reaches 3, the player is transported to the magic land!

The player may want to climb either the tree (at the edge of the forest, room 7) or the ladder (object #7). Neither of these tactics will do the player any good, but we include a CLIMB routine anyway:

```
4100 IF V$ <>"CLI" THEN 4300
4110 IF N$ <>"TRE" THEN 4140
4120 IF R <>7 THEN PRINT "THERE'S NO TREE HERE!" :
GOTO 100
4130 PRINT "YOU CAN'T REACH THE BRANCHES!" : GOTO
100
4140 IF N$ <>"LAD" THEN 4280
4145 IF OB(7) <>R AND OB(7) <>-1 THEN PRINT "YOU
DON'T HAVE THE LADDER!" : GOTO 100
4150 IF R<>7 THEN 4180
4160 PRINT "THE LADDER SINKS UNDER YOUR WEIGHT!"
```

```
4170 PRINT "IT DISAPPEARS INTO THE GROUND!" : OB(7)=0
: GOTO 100
4180 IF R<>20 THEN PRINT "WHATEVER FOR?" : GOTO
100
4280 PRINT "IT WON'T DO ANY GOOD."
```

The way to get to the top of the tree, of course, is to jump. Here is the JUMP routine:

```
4300 IF V$<> "JUM" THEN 4400
4310 IF R<>7 AND R<>8 THEN PRINT "WHEE! THAT WAS
FUN!" : GOTO 100
4315 IF R=8 THEN 4350
4320 PRINT "YOU GRAB THE LOWEST BRANCH OF THE"
4330 PRINT "TREE AND PULL YOURSELF UP. . . ."
4340 R=8 : GOTO 90
4350 PRINT "YOU GRAB A HIGHER BRANCH ON THE"
4360 PRINT "TREE AND PULL YOURSELF UP. . . ."
4370 R=19 : GOTO 90
```

To find the sword (object #10), the player must dig a hole in the open field (room 6). But before the player can dig, he or she must have the shovel (object #8). Here is the DIG routine:

```
4400 IF V$<>"DIG" THEN 4500
4410 IF N$<>"HOL" AND N$<>"GRO" AND N$<>"" THEN
PRINT "YOU CAN'T DIG THAT!" : GOTO 100
4415 IF OB(8)<>R AND OB(8)<>-1 THEN PRINT "YOU
DON'T HAVE A SHOVEL!" : GOTO 100
4420 IF R<>6 THEN PRINT "YOU DON'T FIND ANYTHING." :
GOTO 100
4430 IF OB(10)<>0 THEN PRINT "THERE'S NOTHING ELSE
THERE!" : GOTO 100
4440 PRINT "THERE'S SOMETHING THERE!" : OB(10)=6 :
GOTO 100
```

When the player gets to the boat (room 13), he or she will try to make it move. Rowing doesn't work, but we include a "ROW" verb to make things interesting:

```
4500 IF V$<>"ROW" THEN 4600
4510 IF N$<>"BOA" AND N$<>"" THEN PRINT "HOW CAN
YOU ROW THAT?" : GOTO 100
4520 IF R<>13 PRINT "YOU'RE NOT IN THE BOAT!" : GOTO
100
4530 PRINT "YOU DON'T HAVE AN OAR!" : GOTO 100
```

The *real* way to move the boat is to wave the magic fan (object #12). Here is the "WAVE" verb routine:

```
4600 IF V$<>"WAV" THEN 4700
4610 IF N$<>"FAN" THEN PRINT "YOU CAN'T WAVE THAT!"
: GOTO 100
4615 IF OB(12)<>R AND OB(12)<>-1 THEN PRINT "YOU
DON'T HAVE THE FAN!" : GOTO 100
4620 IF R<>13 THEN PRINT "YOU FEEL A REFRESHING
BREEZE!" : GOTO 100
4630 PRINT "A POWERFUL BREEZE PROPELS THE BOAT"
4640 PRINT "TO THE OPPOSITE SHORE!"
4650 IF OB(11)=140 THEN OB(11)=142 : GOTO 100
4660 OB(11)=140 : GOTO 100
```

Once the boat has been WAVEd to the opposite shore (room 14), the player will need to get out. Here is the LEAVE (or EXIT) routine:

```
4700 IF V$<>"LEA" AND V$<>"EXI" THEN 4800
4710 IF R<>13 THEN PRINT "PLEASE GIVE A DIRECTION!" :
GOTO 100
4720 IF N$ <>"BOA" AND N$ <>"" THEN PRINT "HUH?" :
```

```
GOTO 100
4730 R=OB(11)-128 : GOTO 90
```

Finally, the player will come to the castle. A nasty guard (object #13) stands in the door. The player must FIGHT the guard, but the fight will be useless without the sword (object #10). Here is the FIGHT routine:

```
4800 IF V$ <>"FIG" THEN 4900
4810 IF N$="" THEN PRINT "WHOM DO YOU WANT TO
FIGHT?" : GOTO 100
4820 IF N$ <>"GUA" THEN PRINT "YOU CAN'T FIGHT HIM!" :
GOTO 100
4830 IF R <>16 THEN PRINT "THERE'S NO GUARD HERE!" :
GOTO 100
4840 IF OB(10) <>-1 THEN PRINT "YOU DON'T HAVE A
WEAPON!" : GOTO 100
4850 PRINT "THE GUARD, NOTICING YOUR SWORD,"
4860 PRINT "WISELY RETREATS INTO THE CASTLE."
4870 MA(16,0)=17 : OB(13)=0 : GOTO 100
```

At long last, the player is inside the large hall (room 18) facing the glass case (object #14) that contains the jewel (object #15). But the case is electrified! The gloves (object #17) are needed here, but to be effective the player must WEAR them. Here is the "WEAR" verb:

```
4900 IF V$ <>"WEA" THEN 24900
4910 IF N$ <>"GLO" THEN PRINT "YOU CAN'T WEAR
THAT!" : GOTO 100
4920 IF OB(16) <>R AND OB(16) <>-1 THEN PRINT "YOU
DON'T HAVE THE GLOVES." : GOTO 100
4930 PRINT "YOU ARE NOW WEARING THE GLOVES." :
GF=1 : GOTO 100
```

If the player types WEAR GLOVES while carrying the gloves, the variable GF ("glove flag") will be set to 1, which tells the OPEN routine that the gloves are being worn. The case can then be opened. The jewel may then be extracted, and the game won!

And that's all of the verbs that this game understands. If the player types a verb that the game doesn't know, the program must indicate the error, like this:

```
24900 PRINT "I DON'T KNOW HOW TO DO THAT" : GOTO
100
```

And now the game is almost ready to be played. We'll take care of the final details in the next chapter.

CHAPTER EIGHT

FINISHING TOUCHES

And now that our adventure program is at an end, we must write its beginning. Many adventure programs, including this one, open with a short text message, explaining the premise of the adventure. You don't have to include such an introductory message in your programs, but it is a pleasant touch. Here is the introductory section of our adventure:

```
30000 PRINT "ALL YOUR LIFE YOU HAD HEARD THE STO
RIES"
30010 PRINT "ABOUT YOUR CRAZY UNCLE SIMON. HE WAS
AN"
30020 PRINT "INVENTOR, WHO KEPT DISAPPEARING FOR"
30030 PRINT "LONG PERIODS OF TIME, NEVER TELLING"
30040 PRINT "ANYONE WHERE HE HAD BEEN."
30050 PRINT
30060 PRINT "YOU NEVER BELIEVED THE STORIES, BUT"
30070 PRINT "WHEN YOUR UNCLE DIED AND LEFT YOU
HIS"
30080 PRINT "DIARY, YOU LEARNED THAT THEY WERE
TRUE."
```

```
30090 PRINT "YOUR UNCLE HAD DISCOVERED A MAGIC"
30100 PRINT "LAND, AND A SECRET FORMULA THAT
COULD"
30110 PRINT "TAKE HIM THERE. IN THAT LAND WAS A"
30120 PRINT "MAGIC RUBY, AND HIS DIARY CONTAINED"
30130 PRINT "THE INSTRUCTIONS FOR GOING THERE TO"
30140 PRINT "FIND IT."
30150 INPUT A
31999 RETURN
```

The INPUT statement in line 30150 will keep the text on the display until the player presses RETURN. You might want to add a message telling the player to press the RETURN key, if there is room for such a message at the bottom of the display. (Some computers have larger displays than others.) You can include several screens of text in your introduction, by separating them with INPUT statements like this.

The program is pretty much complete now. If you have been typing all of the lines and routines as we went along, you now have the complete program—or just about. Make these changes and additions:

```
20 NR=1 : NO=17 : ND=6 : NI=5
40 PRINT "PLEASE STAND BY . . . ." : PRINT : PRINT
60 R=1 : IN=0 : SF=0 : FF=0 : MX=1 : GF=0
70 GOSUB 30000 : REM INTRODUCTORY SEQUENCE
80 REM ADD A STATEMENT HERE TO CLEAR THE VIDEO DIS
PLAY
```

There are a couple of objects in this program that can be referred to by more than one name. The salt shaker, for instance, might be called SALT or it might be called SHAKER. The BOTTLE might also be referred to, under certain circumstances, as the FORMULA. Objects can have only one tag, but we can put special

routines directly after the parser that look for possible synonyms and correct them. The lines from 210 to 470 are reserved for this. Here are two lines that you should add:

```
210 IF N$="SHA" THEN N$="SAL"
220 IF N$="FOR" THEN N$="BOT"
```

This corrects possible problems with the words "FORMULA" and "SHAKER."

If you haven't been typing along, or you aren't sure that you have everything right, the complete program is printed in Appendix A. If you have problems in making the program run correctly, proofread it against this version, and see if you can find any mistakes in typing, or sections that you might have left out.

In the second appendix, Appendix B, you will find the Skeleton Adventure Program. This skeleton program contains most of the routines you need to get your own adventure game program up and running. It also includes a lot of REM statements, which tell you where you should add your own routines and DATA statements, to create your own adventure.

If you have read all of the chapters leading up to this one, you should have no trouble using the skeleton adventure. It can be the source of an unlimited number of adventures. Just fill in the blanks.

Of course, knowing how to program isn't everything there is to writing an adventure. The most important element isn't programming—it's imagination.

Nobody can teach you how to be imaginative. That's the part you have to do all by yourself.

But, of course, you have an imagination machine to help you. . . .

APPENDIX A

THE QUEST

```
1 REM ** THE QUEST **
2 REM **
3 REM ** An adventure game
4 REM
10 REM Put a statement here to clear the screen. If you are using a
Radio Shack Model I, III, or 4, add a CLEAR statement. (See text.)
20 NR=19 : NO=17 : ND=6 : NI=5
30 DIM R$(NR),OB(NO),OB$(NO),02$(NO),MA(NR,ND)
40 PRINT "Please stand by. . . ." : PRINT : PRINT
50 GOSUB 25000 : GOSUB 26000 : GOSUB 27000 : REM initialize
arrays
60 R=1 : IN=0 : SF=0 : FF=0 : MX=1 : GF=0
70 GOSUB 30000 : REM Execute introductory sequence, if any.
80 REM Put a statement here to clear the screen.
90 GOSUB 700 : GOSUB 500 : GOSUB 600
91 REM
92 REM *** PARSER
99 REM
100 PRINT : CM$="" : INPUT "WHAT NOW";CM$ : IF CM$="" THEN
100
```

```
110 C=0 : V$="" : N$=""
120 C=C+1 : IF C>LEN(CM$) THEN 150
130 W$=MID$(CM$,C,1) : IF W$=" " THEN 150
140 V$=V$+W$ : GOTO 120
150 C=C+1 : IF C>LEN(CM$) THEN 180
160 W$=MID$(CM$,C,1) : IF W$=" " THEN 180
170 N$=N$+W$ : GOTO 150
180 IF V$="" THEN 100
190 IF LEN(V$)>3 THEN V$=LEFT$(V$,3)
200 IF LEN(N$)>3 THEN N$=LEFT$(N$,3)
210 IF N$="SHA" THEN N$="SAL"
220 IF N$="FOR" THEN N$="BOT"
259 REM
290 REM
480 GOTO 2000 : REM Execute verb routines.
490 REM
491 REM *** DIRECTIONS
496 REM
500 PRINT "YOU CAN GO: ";
510 FOR I=0 TO 5
520 IF MA(R,I)>0 THEN PRINT DI$(I) ;" ";
530 NEXT I
540 PRINT
550 RETURN
590 REM
596 REM
600 PRINT "YOU CAN SEE: "
610 FL=0 : FOR I=0 TO NO-1
620 IF (OB(I) AND 127)=R THEN PRINT " ";OB$(I) : FL=1
630 NEXT I
640 IF FL=0 THEN PRINT " NOTHING OF INTEREST"
650 RETURN
690 REM
691 REM ** ROOM DESCRIPTION
```

```
696 REM
700 PRINT : PRINT "YOU ARE ";R$(R)
710 RETURN
1000 IF NO=0 THEN RETURN
1010 FOR I=0 TO NO-1
1020 IF 02$(I)=N$ THEN FL=1 : RO=OB(I) : GOTO 1050
1030 NEXT I
1040 FL=0 : RETURN
1050 RO=OB(I) : IF RO>127 THEN RO=RO-128
1060 RETURN
1904 REM
1998 REM Routine for the verb 'GO'.
1999 REM
2000 IF V$<>"GO" THEN 2500
2010 IF N$="NOR" THEN DI=0 : GOTO 2400
2020 IF N$="SOU" THEN DI=1 : GOTO 2400
2030 IF N$="EAS" THEN DI=2 : GOTO 2400
2040 IF N$="WES" THEN DI=3 : GOTO 2400
2050 IF N$="UP" THEN DI=4 : GOTO 2400
2060 IF N$="DOW" THEN DI=5 : GOTO 2400
2070 IF N$="BOA" AND OB(11)=R+128 THEN R=13 : GOTO 90
2390 PRINT "YOU CAN'T GO THERE!" : GOTO 100
2391 REM
2399 REM
2400 IF MA(R,DI)>0 AND MA(R,DI)<128 THEN R=MA(R,DI) : GOTO
90
2409 REM
2410 IF MA(R,DI)=128 THEN PRINT "THE GUARD WON'T LET YOU!"
: GOTO 100
2480 GOTO 2390
2490 REM
2499 REM *** 'GET' ROUTINE
2500 IF V$<>"GET" AND V$<>"TAK" THEN 2600
2510 GOSUB 1000
```

```
2520 IF FL=0 THEN PRINT "YOU CAN'T GET THAT!" : GOTO 100
2530 IF RO=-1 THEN PRINT "YOU ALREADY HAVE IT!" : GOTO
100
2540 IF OB(I)>127 THEN PRINT "YOU CAN'T GET THAT!" : GOTO
100
2550 IF RO<>R THEN PRINT "THAT'S NOT HERE!" : GOTO 100
2570 IF IN>NI THEN PRINT "YOU CAN'T CARRY ANY MORE." :
GOTO 100
2575 IF R=18 AND N$="RUB" THEN PRINT "CONGRATULATIONS!
YOU'VE WON!" : GOTO 3430
2580 IN=IN+1 : OB(I)=-1 : PRINT "TAKEN." : GOTO 100
2599 REM *** 'DROP' ROUTINE
2600 IF V$<>"DRO" AND V$<>"THR" THEN 2700
2610 GOSUB 1000
2620 IF FL=0 OR RO<>-1 THEN PRINT "YOU DON'T HAVE
THAT!" : GOTO 100
2650 IN=IN-1 : OB(I)=-1 : PRINT "DROPPED." : GOTO 100
2699 REM *** 'INVENTORY' ROUTINE
2700 IF V$<>"INV" AND V$<>"I" THEN 2800
2710 FL=0 : PRINT "YOU ARE CARRYING:"
2720 FOR I=0 TO NO-1
2730 IF OB(I)=-1 THEN PRINT " ";OB$(I) : FL=1
2740 NEXT I
2750 IF FL=0 THEN PRINT " NOTHING"
2760 GOTO 100
2799 REM *** 'LOOK' ROUTINE
2800 IF V$<>"LOO" AND V$<>"L" THEN 2900
2810 IF N$<>"" THEN 2910
2820 GOTO 90
2899 REM *** 'EXAMINE' ROUTINE
2900 IF V$<>"EXA" THEN 3400
2910 IF N$<>"GRO" THEN 2940
2920 IF R<>6 THEN PRINT "IT LOOKS LIKE GROUND!": GOTO
100
```

```
2930 PRINT "IT LOOKS LIKE SOMETHING'S BURIED HERE." : GOTO
100
2940 REM
3000 GOSUB 1000
3010 IF RO<>R AND RO<>-1 THEN PRINT "IT'S NOT HERE!" :
GOTO 100
3020 IF N$="BOT" THEN PRINT "THERE'S SOMETHING WRITTEN
ON IT!" : GOTO 100
3030 IF N$="CAS" THEN PRINT "THERE'S A JEWEL INSIDE!":
GOTO 100
3040 IF N$="BAR" THEN PRINT "IT'S FILLED WITH RAINWATER." :
GOTO 100
3390 PRINT "YOU SEE NOTHING UNUSUAL." : GOTO 100
3400 IF V$<>"QUI" THEN 3500
3410 PRINT "ARE YOU SURE YOU WANT TO QUIT (Y/N)"; : INPUT
QU$
3420 IF QU$="N" THEN GOTO 100
3430 PRINT "WOULD YOU LIKE TO PLAY AGAIN (Y/N)"; : INPUT
QU$
3440 IF QU$="Y" THEN RUN
3450 IF QU$="N" THEN END
3460 GOTO 3430
3500 IF V$<>"REA" THEN 3700
3510 IF N$<>"DIA" THEN 3560
3520 IF OB(0)<>R AND OB(0)<>-1 THEN PRINT "THERE'S NO
DIARY HERE!" : GOTO 100
3530 PRINT "IT SAYS: 'ADD SODIUM CHLORIDE PLUS THE"
3540 PRINT "FORMULA TO RAINWATER, TO REACH THE"
3550 PRINT "OTHER WORLD.' " : GOTO 100
3560 IF N$<>"DIC" THEN 3590
3570 IF OB(4)<>R AND OB(4)<>-1 THEN PRINT "YOU DON'T SEE
A DICTIONARY!" : GOTO 100
3580 PRINT "IT SAYS: SODIUM CHLORIDE IS" : PRINT "COMMON
TABLE SALT." : GOTO 100
```

```
3590 IF N$<>"BOT" THEN 3620
3600 IF OB(6)<>R AND OB(6)<>-1 THEN PRINT "THERE'S NO
BOTTLE HERE!" : GOTO 100
3610 PRINT "IT READS: 'SECRET FORMULA'." : GOTO 100
3620 REM
3690 PRINT "YOU CAN'T READ THAT!" : GOTO 100
3700 IF V$<>"OPE" THEN 3900
3710 IF N$<>"BOX" THEN 3740
3720 IF OB(1)<>R AND OB(1)<>-1 THEN PRINT "THERE'S NO
BOX HERE!" : GOTO 100
3730 OB(6)=R : PRINT "SOMETHING FELL OUT!" : GOTO 100
3740 IF N$<>"CAB" THEN 3770
3750 IF R<>2 THEN PRINT "THERE'S NO CABINET HERE!" : GOTO
100
3760 PRINT "THERE'S SOMETHING INSIDE!" : OB(3)=2 : GOTO
100
3770 IF N$<>"CAS" THEN 3820
3780 IF R<>18 THEN PRINT "THERE'S NO CASE HERE!" : GOTO
100
3790 IF GF<>1 THEN PRINT "THE CASE IS ELECTRIFIED!" : GOTO
100
3800 PRINT "THE GLOVES INSULATE AGAINST THE"
3810 PRINT "ELECTRICITY! THE CASE OPENS!"
3820 OB(15)=18 : GOTO 100
3890 PRINT "YOU CAN'T OPEN THAT!" : GOTO 100
3900 IF V$<>"POU" THEN 4100
3910 IF N$<>"SAL" THEN 3960
3920 IF OB(3)<>R AND OB(3)<>-1 THEN PRINT "YOU DON'T
HAVE THE SALT!" : GOTO 100
3930 IF SF=1 THEN PRINT "THE SHAKER IS EMPTY!" : GOTO 100
3940 IF R=5 THEN MX=MX+1
3950 SF=1 : PRINT "POURED!" : GOTO 4010
3960 IF N$<>"BOT" THEN PRINT "YOU CAN'T POUR THAT!" :
GOTO 100
3970 IF OB(6)<>R AND OB(3)<>-1 THEN PRINT "YOU DON'T
```

```
HAVE THE BOTTLE!" : GOTO 100
3980 IF FF=1 THEN PRINT "THE BOTTLE IS EMPTY!" : GOTO 100
3990 IF R=5 THEN MX=MX+1
4000 FF=1 : PRINT "POURED!"
4010 IF MX<3 THEN 100
4020 PRINT "THERE IS AN EXPLOSION!"
4030 PRINT "EVERYTHING GOES BLACK!"
4040 PRINT "SUDDENLY YOU ARE. . ."
4050 PRINT ". . .SOMEWHERE ELSE!"
4060 R=6 : GOTO 90
4100 IF V$<>"CLI" THEN 4300
4110 IF N$<>"TRE" THEN 4140
4120 IF R<>7 THEN PRINT "THERE'S NO TREE HERE!" : GOTO
100
4130 PRINT "YOU CAN'T REACH THE BRANCHES!" : GOTO 100
4140 IF N$<>"LAD" THEN 4290
4145 IF OB(7)<>R AND OB(7)<>-1 THEN PRINT "YOU DON'T
HAVE THE LADDER!" : GOTO 100
4150 IF R<>7 THEN 4180
4160 PRINT "THE LADDER SINKS UNDER YOUR WEIGHT!"
4170 PRINT "IT DISAPPEARS INTO THE GROUND!" : OB(7)=0 :
GOTO 100
4180 PRINT "WHATEVER FOR?" : GOTO 100
4290 PRINT "IT WON'T DO ANY GOOD." : GOTO 100
4300 IF V$<>"JUM" THEN 4400
4310 IF R<>7 AND R<>8 THEN PRINT "WHEE! THAT WAS FUN!" :
GOTO 100
4315 IF R=8 THEN 4350
4320 PRINT "YOU GRAB THE LOWEST BRANCH OF THE"
4330 PRINT "TREE AND PULL YOURSELF UP. . . ."
4340 R=8 : GOTO 90
4350 PRINT "YOU GRAB A HIGHER BRANCH ON THE"
4360 PRINT "TREE AND PULL YOURSELF UP. . . ."
4370 R=19 : GOTO 90
4400 IF V$<>"DIG" THEN 4500
```

```
4410 IF N$<>"HOL" AND N$<>"GRO" AND N$<>"" THEN
PRINT "YOU CAN'T DIG THAT!" : GOTO 100
4415 IF OB(8)<>R AND OB(8)<>-1 THEN PRINT "YOU DON'T
HAVE A SHOVEL!" : GOTO 100
4420 IF R<> 6 THEN PRINT "YOU DON'T FIND ANYTHING." : GOTO
100
4430 IF OB(10)<>0 THEN PRINT "THERE'S NOTHING ELSE
THERE!" : GOTO 100
4440 PRINT "THERE'S SOMETHING THERE!" : OB(10)=6 : GOTO
100
4500 IF V$<>"ROW" THEN 4600
4510 IF N$<>"BOA" AND N$<>"" THEN PRINT "HOW CAN YOU
ROW THAT?" : GOTO 100
4520 IF R<>13 PRINT "YOU'RE NOT IN THE BOAT!" : GOTO 100
4530 PRINT "YOU DON'T HAVE AN OAR!" : GOTO 100
4600 IF V$<>"WAV" THEN 4700
4610 IF N$<>"FAN" THEN PRINT "YOU CAN'T WAVE THAT!" :
GOTO 100
4615 IF OB(12)<>R AND OB(12)<>-1 THEN PRINT "YOU DON'T
HAVE THE FAN!" : GOTO 100
4620 IF R<>13 THEN PRINT "YOU FEEL A REFRESHING BREEZE!"
: GOTO 100
4630 PRINT "A POWERFUL BREEZE PROPELS THE BOAT"
4640 PRINT "TO THE OPPOSITE SHORE!"
4650 IF OB(11)=140 THEN OB(11)=142 : GOTO 100
4660 OB(11)=140 : GOTO 100
4700 IF V$<>"LEA" AND V$<>"EXI" THEN 4800
4710 IF R<>13 THEN PRINT "PLEASE GIVE A DIRECTION!" : GOTO
100
4720 IF N$<>"BOA" AND N$<>"" THEN PRINT "HUH?" : GOTO
100
4730 R=OB(11)-128 : GOTO 90
4800 IF V$<>"FIG" THEN 4900
4810 IF N$="" THEN PRINT "WHOM DO YOU WANT TO FIGHT?" :
```

```
GOTO 100
4820 IF N$<>"GUA" THEN PRINT "YOU CAN'T FIGHT HIM!" : GOTO
100
4830 IF R<>16 THEN PRINT "THERE'S NO GUARD HERE!" : GOTO
100
4840 IF OB(10)<>-1 THEN PRINT "YOU DON'T HAVE A WEAPON!"
: GOTO 100
4850 PRINT "THE GUARD, NOTICING YOUR SWORD,"
4860 PRINT "WISELY RETREATS INTO THE CASTLE."
4870 MA(16,0)=17 : OB(13)=0 : GOTO 100
4900 IF V$<>"WEA" THEN 5000
4910 IF N$<>"GLO" THEN PRINT "YOU CAN'T WEAR THAT!" :
GOTO 100
4920 IF OB(16)<>R AND OB(16)<>-1 THEN PRINT "YOU DON'T
HAVE THE GLOVES." : GOTO 100
4930 PRINT "YOU ARE NOW WEARING THE GLOVES." : GF=1 :
GOTO 100
5000 REM
24900 PRINT "I DON'T KNOW HOW TO DO THAT" : GOTO 100
24990 REM
24991 REM The following routine reads the map data into
24992 REM the map array, MA(ROOM, DIRECTION).
24993 REM
25000 IF NR=0 THEN RETURN
25010 DI$(0)="NORTH" : DI$(1)="SOUTH" : DI$(2)="EAST"
25020 DI$(3)="WEST" : DI$(4)="UP" : DI$(5)="DOWN"
25030 FOR I=1 TO NR
25040 FOR J=0 TO ND-1
25050 READ MA(I,J)
25060 NEXT J
25070 NEXT I
25080 RETURN
25099 REM
25100 DATA 4,3,2,0,0,0 : REM LIVING ROOM
```

```
25110 DATA 0,0,0,1,0,0 : REM KITCHEN
25120 DATA 1,0,0,0,0,0 : REM LIBRARY
25130 DATA 0,1,0,5,0,0 : REM FRONT YARD
25140 DATA 0,0,4,0,0,0 : REM GARAGE
25150 DATA 9,7,0,0,0,0 : REM OPEN FIELD
25160 DATA 6,0,0,0,0,0 : REM EDGE OF FOREST
25170 DATA 0,0,0,0,0,7 : REM BRANCH OF TREE
25180 DATA 0,6,10,0,0,0 : REM LONG, WINDING ROAD (1)
25190 DATA 11,0,0,9,0,0 : REM LONG, WINDING ROAD (2)
25200 DATA 0,10,0,12,0,0 : REM LONG, WINDING ROAD (3)
25210 DATA 0,0,11,0,0,0 : REM SOUTH BANK OF RIVER
25220 DATA 0,0,0,0,0,0 : REM BOAT
25230 DATA 15,0,0,0,0,0 : REM NORTH BANK OF RIVER
25240 DATA 16,14,0,0,0,0 : REM WELL-TRAVELED ROAD
25250 DATA 128,15,0,0,0,0 : REM SOUTH OF CASTLE
25260 DATA 0,0,0,0,18,0 : REM NARROW HALL
25270 DATA 0,0,0,0,0,17 : REM LARGE HALL
25280 DATA 0,0,0,0,0,8 : REM TOP OF TREE
25900 REM
25910 REM The following subroutine reads the object data
25920 REM into the three object arrays, OB(X), OB$(X), and
25930 REM O2$(X).
25940 REM
26000 IF NO=0 THEN RETURN
26010 FOR I=0 TO NO-1
26020 READ OB$(I),02$(I),OB(I)
26030 NEXT I
26040 RETURN
26099 REM
26100 DATA AN OLD DIARY, DIA, 1 : REM OBJECT #0
26110 DATA A SMALL BOX, BOX, 1 : REM OBJECT #1
26120 DATA CABINET, CAB, 130 : REM OBJECT #2
26130 DATA A SALT SHAKER, SAL, 0 : REM OBJECT #3
26140 DATA A DICTIONARY, DIC, 3 : REM OBJECT #4
26150 DATA WOODEN BARREL, BAR, 133 : REM OBJECT #5
```

```
26160 DATA A SMALL BOTTLE, BOT, 0 : REM OBJECT #6
26170 DATA A LADDER, LAD, 4 : REM OBJECT #7
26180 DATA A SHOVEL, SHO, 5 : REM OBJECT #8
26190 DATA A TREE, TRE, 135 : REM OBJECT #9
26200 DATA A GOLDEN SWORD, SWO, 0 : REM OBJECT #10
26210 DATA A WOODEN BOAT, BOA, 140 : REM OBJECT #11
26220 DATA A MAGIC FAN, FAN, 8 : REM OBJECT #12
26230 DATA A NASTY-LOOKING GUARD, GUA, 144 : REM OBJECT
#13
26240 DATA A GLASS CASE, CAS, 146 : REM OBJECT #14
26250 DATA A GLOWING RUBY, RUB, 0 : REM OBJECT #15
26260 DATA A PAIR OF RUBBER GLOVES, GLO, 19 : REM OBJECT
#17
26990 REM
27000 R$(1)="IN YOUR LIVING ROOM."
27010 R$(2)="IN THE KITCHEN."
27020 R$(3)="IN THE LIBRARY."
27030 R$(4)="IN THE FRONT YARD."
27040 R$(5)="IN THE GARAGE."
27050 R$(6)="IN AN OPEN FIELD."
27060 R$(7)="AT THE EDGE OF A FOREST."
27070 R$(8)="ON A BRANCH OF A TREE."
27080 R$(9)="ON A LONG, WINDING ROAD."
27090 R$(10)="ON A LONG, WINDING ROAD."
27100 R$(11)="ON A LONG, WINDING ROAD."
27110 R$(12)="ON THE SOUTH BANK OF A RIVER."
27120 R$(13)="INSIDE THE WOODEN BOAT."
27130 R$(14)="ON THE NORTH BANK OF A RIVER."
27140 R$(15)="ON A WELL-TRAVELED ROAD."
27150 R$(16)="IN FRONT OF A LARGE CASTLE."
27160 R$(17)="IN A NARROW HALL."
27170 R$(18)="IN A LARGE HALL."
27180 R$(19)="ON THE TOP OF A TREE."
29900 RETURN
29990 REM
```

```
30000 PRINT "ALL YOUR LIFE YOU HAD HEARD THE STORIES"
30010 PRINT "ABOUT YOUR CRAZY UNCLE SIMON. HE WAS AN"
30020 PRINT "INVENTOR, WHO KEPT DISAPPEARING FOR"
30030 PRINT "LONG PERIODS OF TIME, NEVER TELLING"
30040 PRINT "ANYONE WHERE HE HAD BEEN."
30050 PRINT
30060 PRINT "YOU NEVER BELIEVED THE STORIES, BUT"
30070 PRINT "WHEN YOUR UNCLE DIED AND LEFT YOU HIS"
30080 PRINT "DIARY, YOU LEARNED THAT THEY WERE TRUE."
30090 PRINT "YOUR UNCLE HAD DISCOVERED A MAGIC"
30100 PRINT "LAND, AND A SECRET FORMULA THAT COULD"
30110 PRINT "TAKE HIM THERE. IN THAT LAND WAS A"
30120 PRINT "MAGIC RUBY, AND HIS DIARY CONTAINED"
30130 PRINT "THE INSTRUCTIONS FOR GOING THERE TO"
30140 PRINT "FIND IT."
30150 INPUT A
31999 RETURN
```

APPENDIX B

ADVENTURE GAME SKELETON

```
1 REM ** ADVENTURE GAME SKELETON **
2 REM
10 REM The first line of the program should contain a CLEAR
11 REM statement, if your computer requires one. (See text
12 REM for details.) Also, you include a statement
13 REM that will clear the video display, such as 'CLS'
14 REM (IBM and Radio Shack computers), 'PRINT CHR$(147)'
15 REM (Commodore computers), 'HOME' (Apple II computers),
16 REM and so forth.
17 REM
20 NR=0 : NO=0 : ND=6 : NI=5
30 DIM R$(NR),OB(NO),OB$(NO),O2$(NO),MA(NR+1,ND)
40 PRINT "Please stand by . . . ." : PRINT : PRINT
50 GOSUB 25000 : GOSUB 26000 : GOSUB 27000 : REM initialize
arrays
60 R=1 : IN=0
70 GOSUB 30000 : REM Execute introductory sequence, if any.
80 REM Put a statement here to clear the video display.
90 GOSUB 700 : GOSUB 500 : GOSUB 600
91 REM
```

```
92 REM *** PARSER
94 REM The following routine prompts the player to type a
95 REM one- or two-word command to the computer, then breaks
96 REM the command into two strings of no more than three
97 REM characters apiece. The first string (the verb) is
98 REM stored in variable V$, the second (the noun) in N$.
99 REM
100 PRINT : CM$="" : INPUT "WHAT NOW";CM$ : IF CM$="" THEN
100
110 C=0 : V$="" : N$=""
120 C=C+1 : IF C>LEN(CM$) THEN 150
130 W$=MID$(CM$,C,1) : IF W$=" " THEN 150
140 V$=V$+W$ : GOTO 120
150 C=C+1 : IF C>LEN(CM$) THEN 180
160 W$=MID$(CM$,C,1) : IF W$=" " THEN 180
170 N$=N$+W$ : GOTO 150
180 IF V$="" THEN 100
190 IF LEN(V$)>3 THEN V$=LEFT$(V$,3)
200 IF LEN(N$)>3 THEN N$=LEFT$(N$,3)
259 REM
260 REM If you have any special routines that you want
270 REM the computer to perform between moves, such as a
280 REM timekeeping routine, put them here.
290 REM
480 GOTO 2000 : REM Execute verb routines.
490 REM
491 REM *** DIRECTIONS
492 REM The following subroutine prints the words 'YOU CAN GO',
493 REM followed by a list of the directions that the player
494 REM may go from the current room, based on the information
495 REM in the map array, MA(ROOM, DIRECTION).
496 REM
500 PRINT "YOU CAN GO: ";
510 FOR I=0 TO 5
520 IF MA(R,I)>0 THEN PRINT DI$(I) ;" ";
```

```
530 NEXT I
540 PRINT
550 RETURN
590 REM
591 REM *** OBJECTS
592 REM The following routine prints the words 'YOU CAN SEE',
593 REM followed by a description of all objects visible in
594 REM the current room, based on the information in the
595 REM object array, OB(ROOM).
596 REM
600 PRINT "YOU CAN SEE: "
610 FL=0 : FOR I=0 TO NO-1
620 IF (OB(I)AND127)=R THEN PRINT " ";OB$(I) : FL=1
630 NEXT I
640 IF FL=0 THEN PRINT " NOTHING OF INTEREST."
650 RETURN
690 REM
691 REM ** ROOM DESCRIPTION
692 REM The following routine prints the words 'YOU ARE',
693 REM followed by a description of the current room.
694 REM (A room, in adventure game language, refers to any
695 REM location in the game map, both indoors and outdoors.)
696 REM
700 PRINT "YOU ARE ";R$(R)
710 RETURN
1000 IF NO=0 THEN RETURN
1010 FOR I=0 TO NO-1
1020 IF O2$(I)=N$ THEN FL=1 : RO=OB(I) : GOTO 1050
1030 NEXT I
1040 FL=0 : RETURN
1050 RO=OB(I) : IF RO>127 THEN RO=RO-128
1060 RETURN
1900 REM Verb routines begin on line 2000. We have already
1901 REM included routines for the verbs 'GO', 'GET', 'DROP',
1902 REM and 'INVENTORY'. You may add as many additional
```

```
1903 REM routines as you wish.
1904 REM
1998 REM Routine for the verb 'GO'.
1999 REM
2000 IF V$<>"GO" THEN 2500
2010 IF N$="NOR" THEN DI=0 : GOTO 2400
2020 IF N$="SOU" THEN DI=1 : GOTO 2400
2030 IF N$="EAS" THEN DI=2 : GOTO 2400
2040 IF N$="WES" THEN DI=3 : GOTO 2400
2050 IF N$="UP" THEN DI=4 : GOTO 2400
2060 IF N$="DOW" THEN DI=5 : GOTO 2400
2069 REM
2070 REM If there is a specific place where the player can
2090 REM 'GO', such as a building or a door or a hole, etc.,
2100 REM you should use this section of the program to
2110 REM check to see if the player wants to go there. For
2120 REM instance, if the player can type 'GO HOUSE', you
2130 REM can add a line here that reads 'IF N$="HOU" AND R=X
2140 REM THEN R=X1', where X is the number of the room from
2150 REM which the player can go to the house, and X1
2160 REM is the house itself, or the first room inside the
2170 REM house. If the player uses a noun that is not
2180 REM recognized by your program, line 2490 will tell
2190 REM the player that 'YOU CAN'T GO THERE!'
2200 REM
2390 PRINT "YOU CAN'T GO THERE!" : GOTO 100
2391 REM
2392 REM The following lines check to see if the player can
2393 REM move in the requested direction. If so, the room
2394 REM number (R) is changed to the number of the new room
2395 REM and the program goes back to line 90, which prints a
2396 REM description of the new room. Otherwise, the program
2397 REM goes to line 2480, which announces that "YOU CAN'T
2398 REM GO THERE!"
2399 REM
```

```
2400 IF MA(R,DI)>0 AND MA(R,DI)<128 THEN R=MA(R,DI) : GOTO
90
2409 REM
2410 REM Special routines, represented on the map by direction
2411 REM numbers larger than 127, should go here.
2412 REM
2480 GOTO 2390
2490 REM
2491 REM Routine for the verb 'GET'.
2492 REM
2500 IF V$<>"GET" AND V$<>"TAK" THEN 2600
2510 GOSUB 1000
2520 IF FL=0 THEN PRINT "YOU CAN'T GET THAT!" : GOTO 100
2530 IF RO=-1 THEN PRINT "YOU ALREADY HAVE IT!" : GOTO
100
2540 IF OB(I)>127 THEN PRINT "YOU CAN'T GET THAT!" : GOTO
100
2550 IF RO<>R THEN PRINT "THAT'S NOT HERE!" : GOTO 100
2570 IF IN>NI THEN PRINT "YOU CAN'T CARRY ANY MORE." :
GOTO 100
2580 IN=IN+1 : OB(I)=-1 : PRINT "TAKEN." : GOTO 100
2600 IF V$<>"DRO" AND V$<>"THR" THEN 2700
2610 GOSUB 1000
2620 IF FL=0 THEN PRINT "YOU DON'T HAVE THAT!" : GOTO 100
2640 IF RO<>-1 THEN PRINT "YOU DON'T HAVE THAT!" : GOTO
100
2650 IN=IN-1 : OB(I)=-1 : PRINT "DROPPED." : GOTO 100
2700 IF V$<>"INV" AND V$<>"I" THEN 2800
2710 FL=0 : PRINT "YOU ARE CARRYING:"
2720 FOR I=0 TO NO-1
2730 IF OB(I)=-1 THEN PRINT " ";OB$(I) : FL=1
2740 NEXT I
2750 IF FL=0 THEN PRINT " NOTHING"
2760 GOTO 100
2800 IF V$<>"LOO" AND V$<>"L" THEN 2900
```

```
2810 IF N$<>"" THEN 2910
2820 GOTO 90
2900 IF V$<>"EXA" THEN 3400
2910 GOSUB 1000
2920 FOR I=0 TO NO-1
2960 IF RO<>R AND RO<>-1 THEN PRINT "IT'S NOT HERE!" :
GOTO 100
2999 REM
3000 REM Put your 'EXAMINE' routines here.
3001 REM
3400 IF V$<>"QUI" THEN 3500
3410 PRINT "ARE YOU SURE YOU WANT TO QUIT (Y/N)"; : INPUT
QU$
3420 IF QU$="N" THEN GOTO 100
3430 PRINT "WOULD YOU LIKE TO PLAY AGAIN (Y/N)"; : INPUT
QU$
3440 IF QU$="Y" THEN RUN
3450 IF QU$="N" THEN END
3460 GOTO 3430
3490 PRINT "YOU SEE NOTHING UNUSUAL" : GOTO 100
3500 REM Put the rest of your verb routines here.
3510 REM
24900 PRINT"I DON'T KNOW HOW TO DO THAT" : RETURN
24990 REM
24991 REM The following routine reads the map data into
24992 REM the map array, MA(ROOM, DIRECTION).
24993 REM
25000 IF NR=0 THEN RETURN
25010 DI$(0)="NORTH" : DI$(1)="SOUTH" : DI$(2)="EAST"
25020 DI$(3)="WEST" : DI$(4)="UP" : DI$(5)="DOWN"
25030 FOR I=1 TO NR
25040 FOR J=0 TO ND-1
25050 READ MA(I,J)
25060 NEXT J
25070 NEXT I
```

```
25080 RETURN
25099 REM
25100 REM Here is where you put the DATA statements con-
25110 REM taining the map directions. Each DATA statement
25120 REM represents one of the rooms in the game map,
25130 REM starting with room 1. Each DATA statement should
25140 REM contain 6 numbers, one for each of the six
25150 REM map directions: north, south, east, west, up,
25160 REM and down, in that order. (If you are using more
25170 REM than six directions, the DATA statements should
25180 REM have more than six numbers.) If the number for any
25190 REM direction is 0, then the player cannot go in that
25200 REM direction from that room. If the number is
25210 REM greater than 0, then it is the number of the room
25220 REM that the player will be in if he or she goes in
25230 REM that direction.
25240 REM
25900 REM
25910 REM The following subroutine reads the object data
25920 REM into the three object arrays, OB(X), OB$(X), and
25930 REM O2$(X).
25940 REM
26000 IF NO=0 THEN RETURN
26010 FOR I=0 TO NO-1
26020 READ OB$(I),O2$(I),OB(I)
26030 NEXT I
26040 RETURN
26099 REM
26100 REM The DATA statements containing the object
26110 REM information should go here. There should be three
26120 REM data items for each object, the first containing
26130 REM the name of the object, as it will be identified
26140 REM to the player (i.e., 'SMALL RED MATCHBOOK'), the
26150 REM second a three-letter 'tag' that the computer
26160 REM can use to recognize the name when the user types
```

```
26170 REM it (i.e., 'MAT'), and the third the number of the
26180 REM room in which the object is located at the
26190 REM beginning of the game. If the object does not
26200 REM yet exist, or is not yet visible, at the beginning
26210 REM of the game, it should be given a room number of 0.
26220 REM If it is in the player's inventory, it should be
26230 REM given a room number of -1. If the object is
26240 REM immovable--that is, if the player cannot 'GET'
26250 REM the object--you should add 128 to the room
26260 REM number.
27000 REM Here is where you create the array R$ that contains
27010 REM the names of the rooms. You create the array as a
27020 REM series of assignment statements (that is, commands
27030 REM that set a string variable equal to a string, using
27040 REM the equals ('=') sign). Here is what one of these
27050 REM assignment statements might look like:
27060 REM
27070 REM R$(1)="IN A GIANT CAVERN."
27080 REM
27090 REM Be sure NOT to put the word REM in front of the
27100 REM statement! There should be one assignment statement
27110 REM like this for every room in the adventure. The
27120 REM number in parentheses after R$ should be the number
27130 REM of the room on your original map. The 'name' of the
27140 REM room, which follows the equals sign in quotes,
27150 REM should begin with a preposition (like 'in' or 'on'
27160 REM or 'underneath'), followed by a word or words de-
27170 REM scribing the place. This name will be used whenever
27180 REM the player moves to a new room, or uses the 'LOOK'
27190 REM command. It will be preceded by the words 'YOU ARE ',
27200 REM as in 'YOU ARE IN A GIANT CAVERN'.
27210 REM
29900 RETURN
30000 REM Here is where you put your introductory sequence, if
30010 REM you have one. The introductory sequence is a part of
```

```
30020 REM the program that will be executed before the actual
30030 REM adventure begins. It gives the player information he
30040 REM or she needs in order to play the adventure, such as
30050 REM instructions and background about the story. It is an
30060 REM optional part of the program--that is, you do not
30070 REM have to include it if you do not want to.
30080 REM
31999 RETURN
```

INDEX

For a thorough understanding of computers and computer science, be sure you have *all* the titles in Franklin Watts' Computer-Awareness First Book series.

CAREERS IN THE COMPUTER INDUSTRY
COMPUTER CRIME
COMPUTER GRAPHICS
COMPUTER LANGUAGES
COMPUTER MAINTENANCE
COMPUTER PERIPHERALS
COMPUTER PIONEERS
COMPUTERS IN OUR WORLD, TODAY AND TOMORROW
CREATIVE COMPUTER-VIDEO
DATA PROCESSING
ELECTRONIC BULLETIN BOARDS
HOW TO CREATE ADVENTURE GAMES
HOW TO CREATE COMPUTER GAMES
INVENT YOUR OWN COMPUTER GAMES
MICROCOMPUTERS
PROGRAMMING IN BASIC
ROBOTS AND ROBOTICS
THE SCIENCE OF ARTIFICIAL INTELLIGENCE
WORD PROCESSING

A FRANKLIN WATTS LIBRARY EDITION
531-10119-3