WRITE YOUR OWN PROGRAM

GRAPHICS

HANGMAN

FOR COMMODORE 64 AND APPLE IIe COMPUTERS

First published in
Great Britain in 1984 by
Franklin Watts
12a Golden Square
London W1

First published in the
United States in 1984 by
Gloucester Press

Printed in Belgium

ISBN 0 531 03483 6

Library of Congress Catalog
Card Number: 84 81108

WRITE YOUR OWN PROGRAM

GRAPHICS

Mike Duck

GLOUCESTER PRESS
NEW YORK · TORONTO · 1984

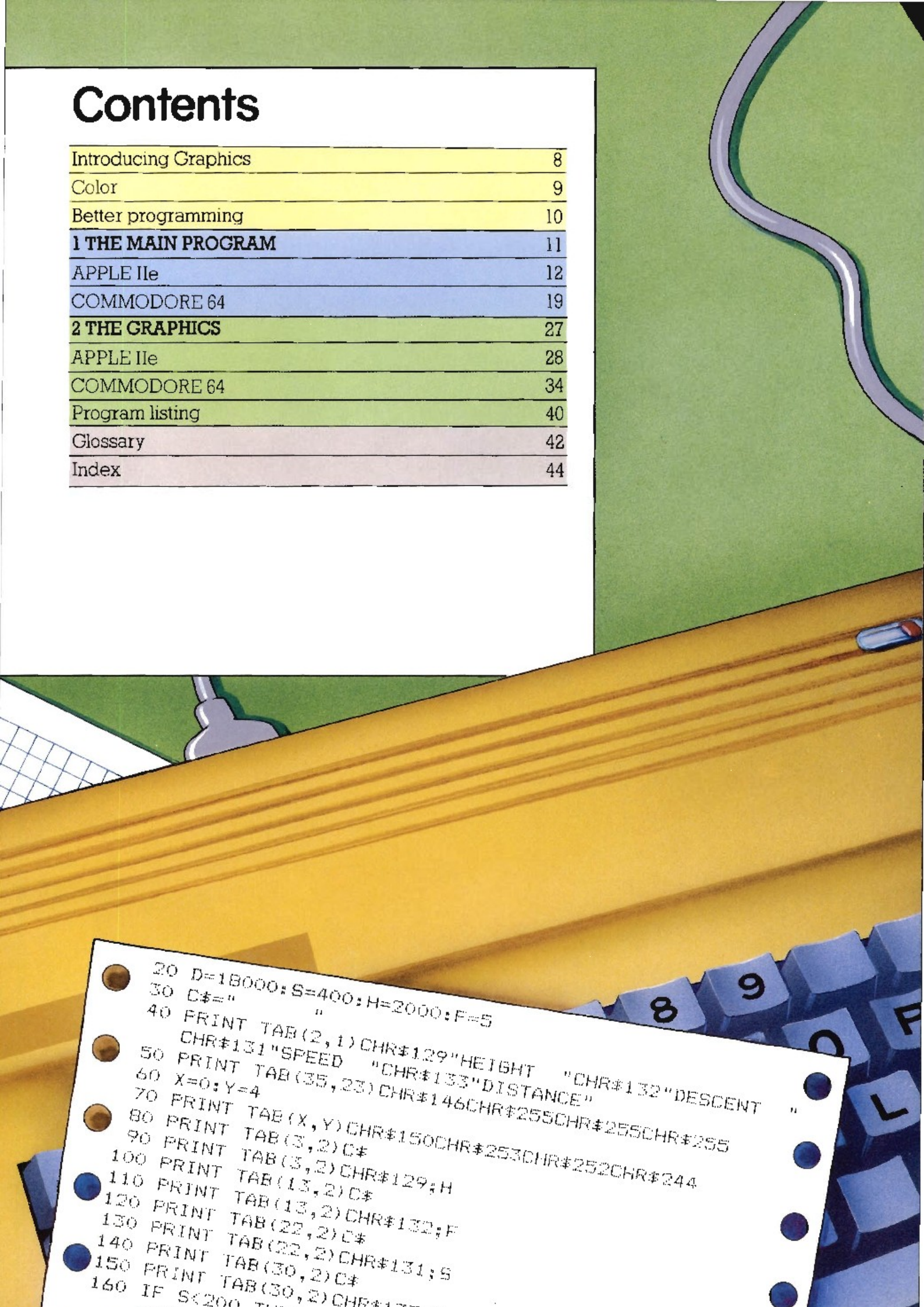
Foreword

One of the most exciting things about computers is the graphics effects they can display on your TV screen. Most ready-bought games programs allow you to control what's happening on the screen, whether it's moving a spaceship to avoid alien attack or steering a racing car around a dangerous Grand Prix track. The programs that produce these effects are very complicated, but you can use simple graphics commands to create an interesting and satisfying computer game of your own.

The program given in this book builds up the Hangman game. In this game, the computer chooses a mystery word and asks the player to guess what it is, letter by letter. You have only ten guesses, and for each wrong guess the computer builds up a section of the Hangman graphics.

The program is broken down into a number of logical stages, so that you can clearly understand what each section does. Giving programs a logical step-by-step structure not only makes them easier to write, it also helps other people to see how they work when they come to use them at a later date. The first section of the book deals with the main program which controls how the game works. A second section explains the graphics that will make up the screen display.

Contents



Introducing graphics

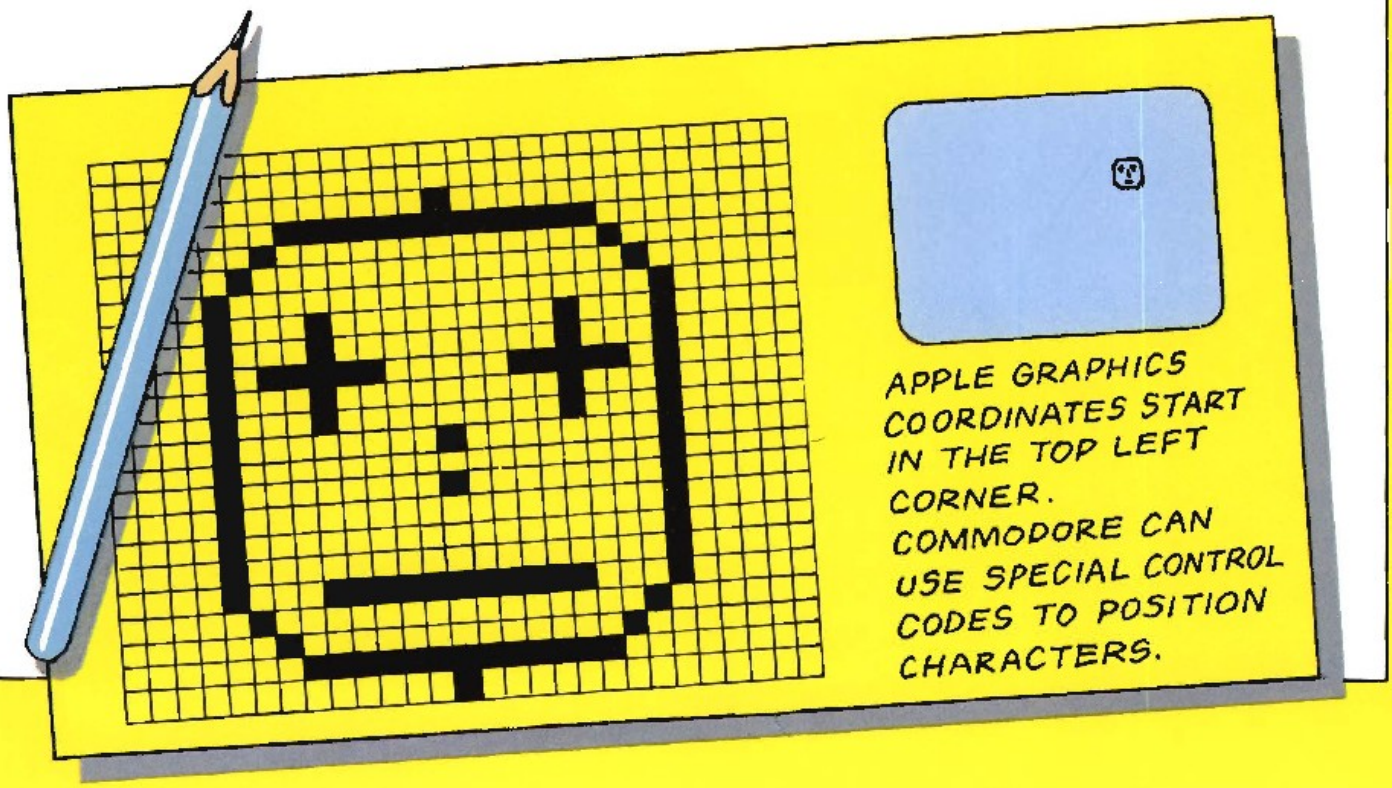
The graphics commands used in the BASIC computer language vary considerably from one model of computer to another. Graphics are positioned on the TV screen by plotting them in the same way that you can plot positions on graph paper. The graphic commands in the Apple are followed by two figures which give the horizontal coordinate (the x coordinate) first, followed by the vertical coordinate (the y coordinate). Commodore 64 can also position graphics on the screen as a series of **PRINT** statements. Type in the graphics commands below and then **RUN** them to see the result.

APPLE IIe

```
7510  HCOLOR= 3: FOR I = 225 TO 245
7520  HPLOT I,49 TO I,30
7530  NEXT I
7540  HCOLOR= 0
7550  HPLOT 230,45 TO 240,45
7560  HPLOT 228,34 TO 230,34: HPLOT 240,34 TO 242,34
7570  HPLOT 234,38 TO 236,38
```

COMMODORE 64

```
7510 PRINT"■■■■■■■■"
7520 PRINTSPC(27)"┌──┐"
7530 FORR=0TO2:PRINTSPC(27)"│  │":NEXT
7540 PRINTSPC(27)"└┬┬┘"
7550 PRINTSPC(28)"+ +":PRINTSPC(29)":":
     PRINTSPC(28)"\__/"
```



Color

To obtain colors, a number of different commands are used. In **HGR** mode of the Apple, colored graphics are produced using the **HCOLOR** command followed by a figure from 0 to 7. The exact color depends on the TV and its adjustment. The background color is black. On the Commodore 64 both graphics and text are colored by the CTRL or C= key held down with the appropriate number key from 1-8. Each color control will display a unique graphic code which will not be visible when the program is **RUN**. The background color is blue. Try out the commands given below, and experiment with the figures using the values given in the chart.

N.B. Colors on the Apple IIe may vary from those given below depending on T.V. used.

Commodore 64 keys used with the Commodore logo key appear with a stroke above them in the text. Thus the Commodore logo shifted E is printed as Ē.

Commodore control codes

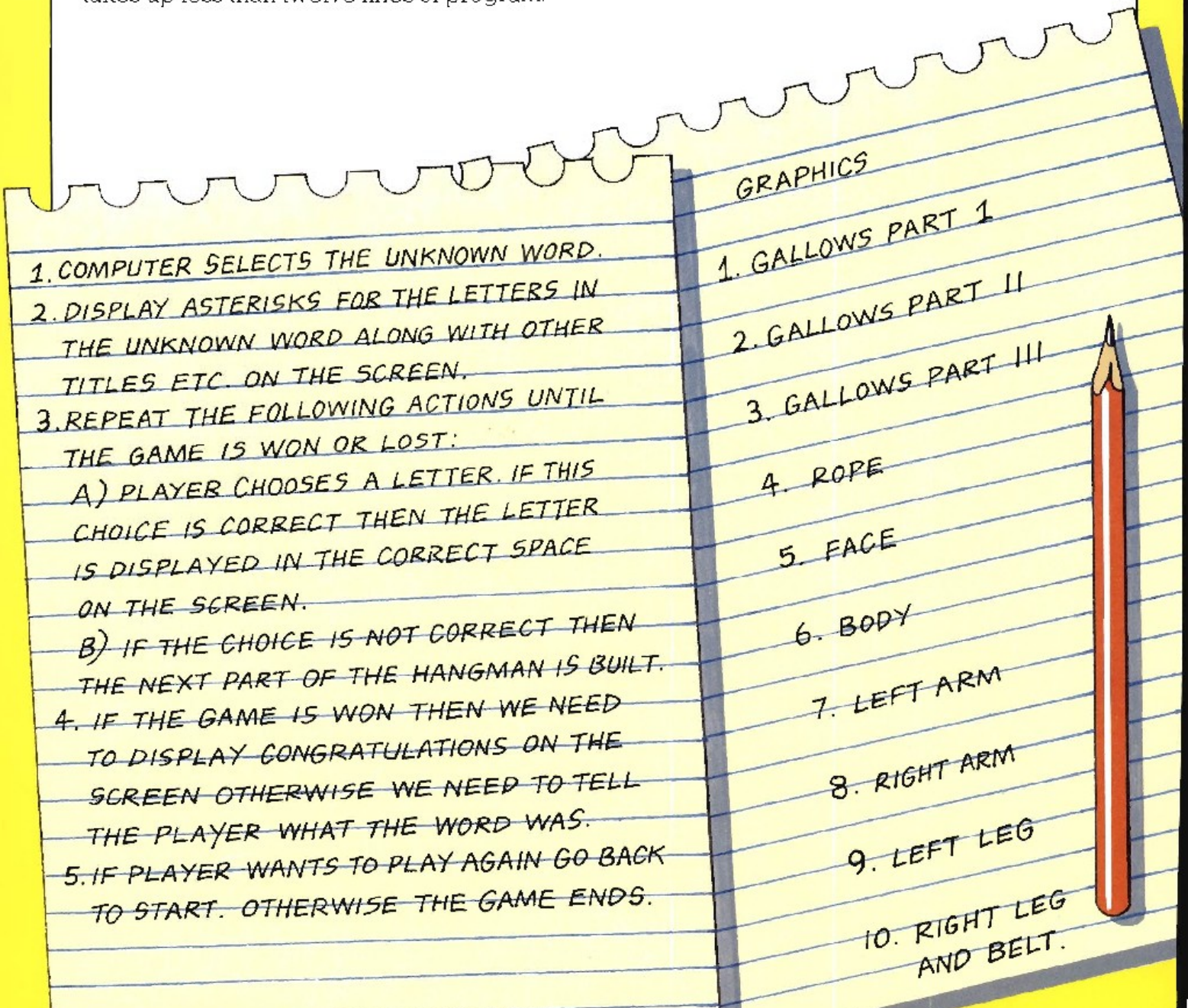
= CLEAR SCREEN	= CURSOR RIGHT
= HOME CURSOR	= CURSOR LEFT
= CURSOR DOWN	= REVERSE ON
= CURSOR UP	= REVERSE OFF

COLORS			
APPLE IIe		COMMODORE	
	HCOLOR 0 HCOLOR 4	CTRL 1	
		CTRL 4	
	HCOLOR 5	CTRL 3	
		CTRL 5	
	HCOLOR 1	CTRL 6	
	HCOLOR 6 HCOLOR 2	CTRL 7	
		CTRL 8	
	HCOLOR 3 HCOLOR 7	CTRL 2	

OTHER COMMODORE COLORS	
ORANGE	1
BROWN	2
LIGHT RED	3
DARK GRAY	4
MEDIUM GRAY	5
LIGHT GREEN	6
LIGHT BLUE	7
LIGHT GRAY	8

Better programming

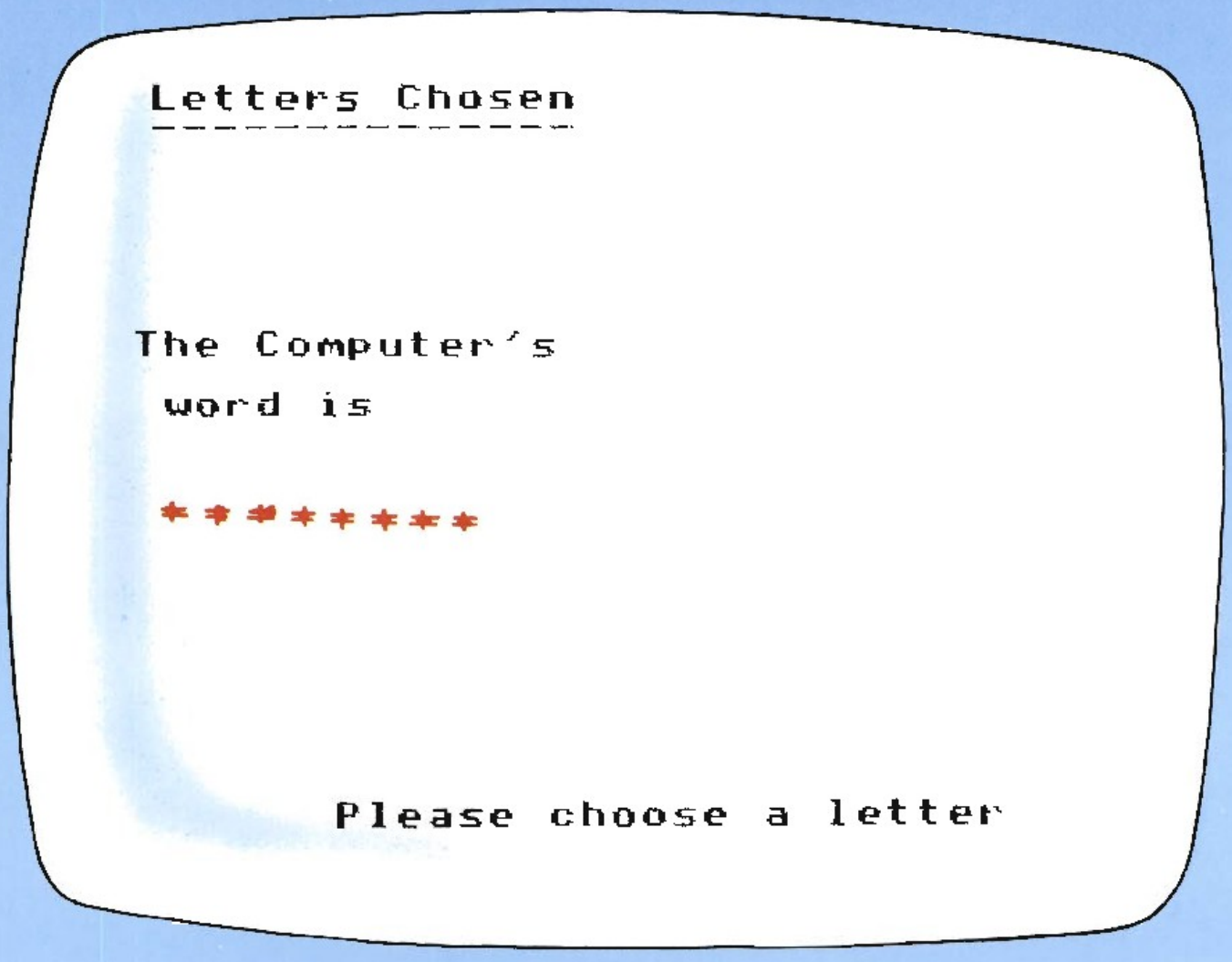
The first stage of writing any program is to break down the task which the computer is to perform into its logical steps. Each of these steps can be made into a separate stage within the program, called a *subroutine* in the Applesoft BASIC used on the Apple computer and in Commodore 64 BASIC. The main program for the game will call up and execute each of these subroutines as they are required. One of the main advantages of writing programs in this way is that it shows exactly how the program works, and it also means that you can use the same subroutine more than once in the program. In fact, in the program in this book, the graphics that are available from the graphics section are used in this way to give an attractive title display and separate displays when the game is won or lost. Using subroutines also means that the main program is much simpler. In fact, the main program which places the subroutines in their logical order takes up less than twelve lines of program.



1

THE MAIN PROGRAM

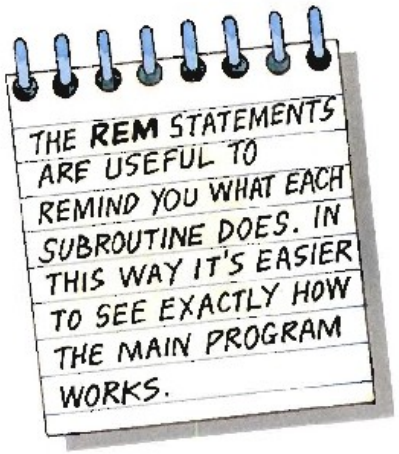
In this section, the separate subroutines which describe the Hangman game are explained. First, the main program is given, with each stage of the program placed in its logical order. The sections of program that follow set up the title screen and then ask the computer to choose a word at random from the list given it. The player is then asked to guess letters, and these guesses are compared with those in the mystery word.



The breakdown for the Hangman game shown on the previous page almost gives the main program. By changing each step into a subroutine, and by placing those subroutines into their logical order, you can arrive at the main program for the game, as shown below. Once you obtain a main program like this, all that remains is to write each subroutine in turn.

```
10   HGR : HOME
20   GOSUB 1000: REM  SELECTWORD
30   GOSUB 2000: REM  TITLE
40   IF WO = 1 OR LO = 1 THEN 80
50   GOSUB 3000: REM  CHOOSELETTER
60   IF CO = 0 THEN  GOSUB 7000: REM  HANGMAN
70   GOTO 40
80   IF WO = 1 THEN  GOSUB 4000: REM  WIN
90   IF LO = 1 THEN  GOSUB 5000: REM  LOSE
100  GOSUB 6000: REM  PLAY AGAIN
110  STOP
```

The first stage of the program is to introduce a subroutine which allows the computer to choose the mystery word. It will do this from **DATA** stored at the end of the program. Once the computer has run through this subroutine it will return to the main program to the next one – in this case the titles subroutine. This subroutine gives the initial title display. Lines 40 to 70 form a loop which is repeated until the condition in line 40 is met – that is, until the game is **WO**n or **LO**st. Line 60 tells the computer to go the *hangman* subroutine when a wrong guess is made during the *chooseletter* subroutine. In line 80 the program is sent to the *win* subroutine, if the game is won. Line 90 of the program goes to the losing display. Line 100 calls up the subroutine which asks if the player wants another game, and at line 110 the main program ends.

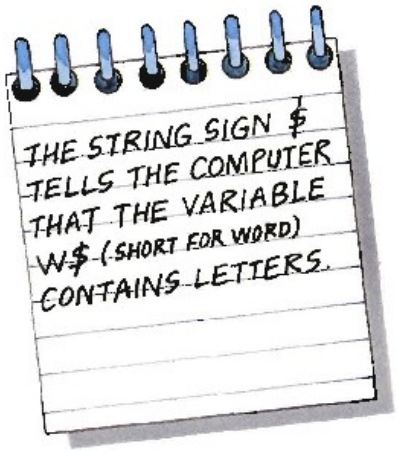


The subroutines are written separately. It is a good idea to check each section to see if it works. Before doing this, it is necessary to include the **DATA** statements at the end of the program. Because the computer automatically puts line numbers into sequence, they can be added at this point. You can have as many words as you like, but at this stage it's better to limit them to eight and add extras when you've finished the program.

```
9000  DATA  APPLE,ANGLE,ALPHABET,ASTEROID
9010  DATA  BOTTLE,BOOK,BUFFALO,BEACH
```

```
1000  REM  SELECT WORD
1010  LET W = 8
1020  RESTORE
1030  LET CH =  INT ( RND (1) * W) + 1
1040  FOR L = 1 TO CH
1050  READ W$
1060  NEXT L
1070  LET LE =  LEN (W$) : LET D$ = W$
1080  RETURN
```

The first subroutine selects the word to be guessed. All subroutines are first defined as in line 1000 – **REM SELECT WORD**. In line 1010, the value of **W** is set to the number of words stored in the **DATA**. **RESTORE** in line 1020 tells the computer to begin at the first word in the **DATA** each time the subroutine is run. Without it, the computer would begin reading where it stopped last time. Line 1030 selects a random number between one and the number of words stored in the **DATA**. The computer will run through the **DATA** to this point, using a **FOR. . . . . NEXT** loop with **L** as its loop variable. If the random number generated is four, then the computer will read through to the fourth word. This word is then held in the string variable **W$** and is the mystery word. Line 1070 defines the length of the word as the length of the word string (**LEN** counts the number of characters held in a string); and also stores the word selected in the variable dummy **D$**. The **LE**ngth is used to display the number of letters to be guessed, and **D$** is used to display the mystery word if the player fails to guess it. **RETURN** directs the computer back to the main program.




```
]GOSUB 1000
BOTTLE
]
```

TO TEST SUBROUTINE **SELECTWORD**

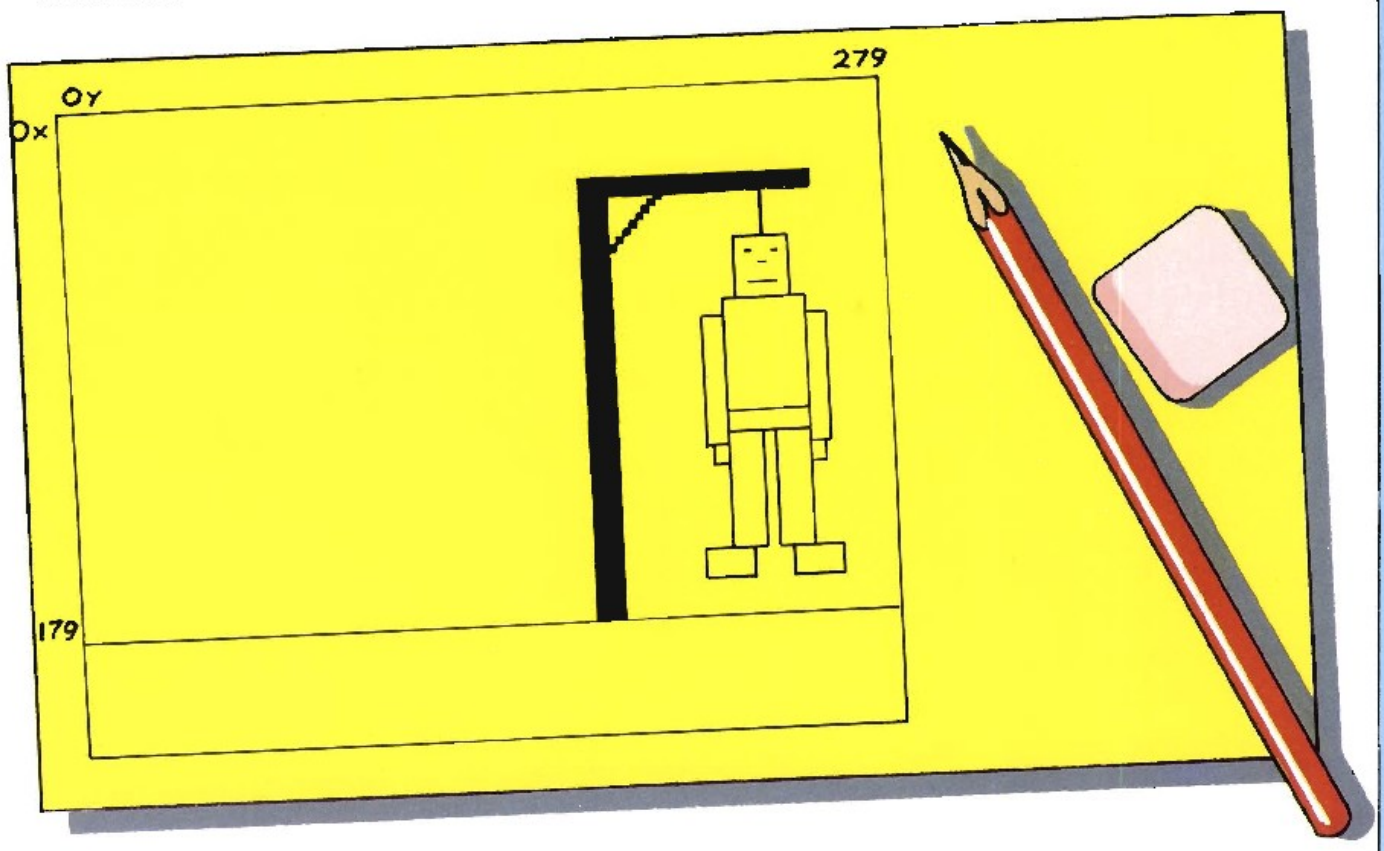
a) ADD LINE:
1075 PRINT W$

b) TYPE **HOME**
PRESS RETURN ↵

c) TYPE **GOSUB 1000**
PRESS RETURN ↵

DELETE LINE 1075 BEFORE PROCEEDING TO THE NEXT SECTION.

The next subroutine calls up the title screen. This uses some of the graphics created later in the *hangman* subroutine. These graphics appear on the right-hand side of the screen, so this should be kept in mind when placing text instructions on the screen.



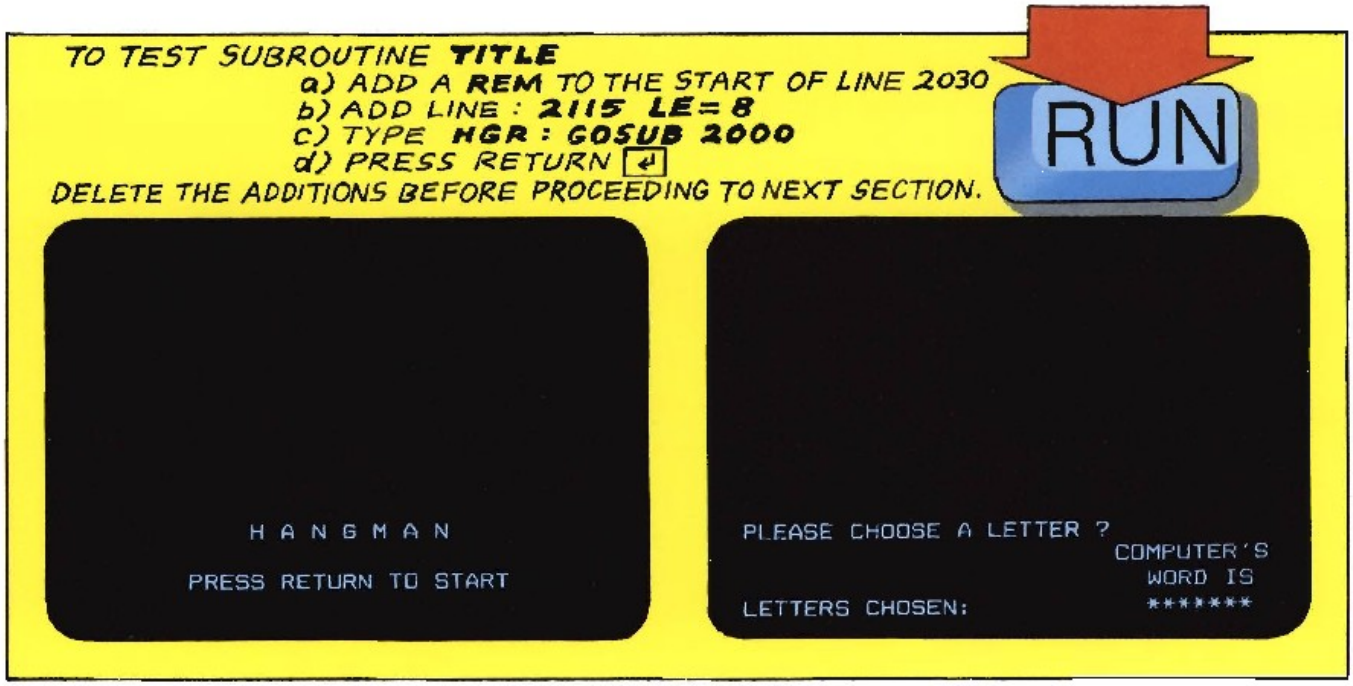

```
2000  REM  TITLE
2010  HGR : HOME
2020  FOR L = 4 TO 10
2030  ON L - 4 GOSUB 7500,7600,7700,7800,7900,8000
2040  NEXT L
2050  VTAB 21: HTAB 14: PRINT "H A N G M A N"
2060  VTAB 23: HTAB 10: INPUT "PRESS RETURN TO START"
      ;A$
```

The titles sequence has two parts: firstly to give the initial screen, secondly to give the screen for the beginning of the game. The section above achieves the first task. Line 2010 selects **HGR** and **HOME** clears the screen. Lines 2020 to 2040 sends the program to the appropriate section of the *hangman* subroutine. This will become clearer when you come to that section of the program. In brief, the hangman image is built up in ten stages. For the title screen only, the graphics created by stages four to ten are used. Line 2050 positions the word "HANGMAN" on row 21 beginning at the fourteenth column across. The player is then instructed to press the **RETURN** key to start.

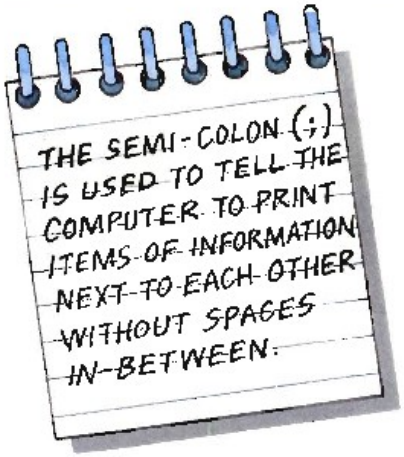
```
2070  HOME : HGR : VTAB 24: HTAB 1: PRINT "LETTERS
      CHOSEN:";
2090  VTAB 22: HTAB 28: PRINT "COMPUTER'S"
2100  VTAB 23: HTAB 30: PRINT "WORD IS"
2110  VTAB 24: HTAB 30
2120  FOR L = 1 TO LE: PRINT "*";: NEXT L
2130  VTAB 21: HTAB 1: PRINT "PLEASE CHOOSE A LETTER"
2150  LET WO = 0: LET LO = 0: LET LI = 0: LET SC = 0
2160  RETURN
```

Lines 2070 to 2160 complete the *title* subroutine. The considerations to take into account are to display the number of letters in the mystery word and to show the letters that the player has guessed. The text "LETTERS CHOSEN" and 'COMPUTER'S WORD IS" are displayed at the bottom of the screen in the text area (lines 21-24). Applesoft graphics leaves four lines for text on each graphics screen. The instruction "PLEASE CHOOSE A LETTER" is displayed on line 21 starting in column one.

At line 2120, an asterisk is printed for every letter in the selected word. This is done using **FOR. . . . . .NEXT** loop with **L** as its loop variable, with values ranging from 1 to the value for length obtained in **LE** earlier. Finally, line 2150 sets the initial values for the other variables used in the game at zero. **SC** will be used to decide whether or not the game is won. **LI** checks that the player hasn't used up ten guesses and builds up separate sections of the hangman graphics for each incorrect guess. And the values of *WOn* and *LOst* are set at zero (that is, the game has been neither won nor lost).



The next subroutine really does all the work in the program! It allows the player to choose a letter and checks to see if that letter is part of the mystery word. According to whether the guess is right or wrong it makes the appropriate changes to the values of **SC**, and checks to see if the player has won. As you might expect, it's also the most difficult section of program.

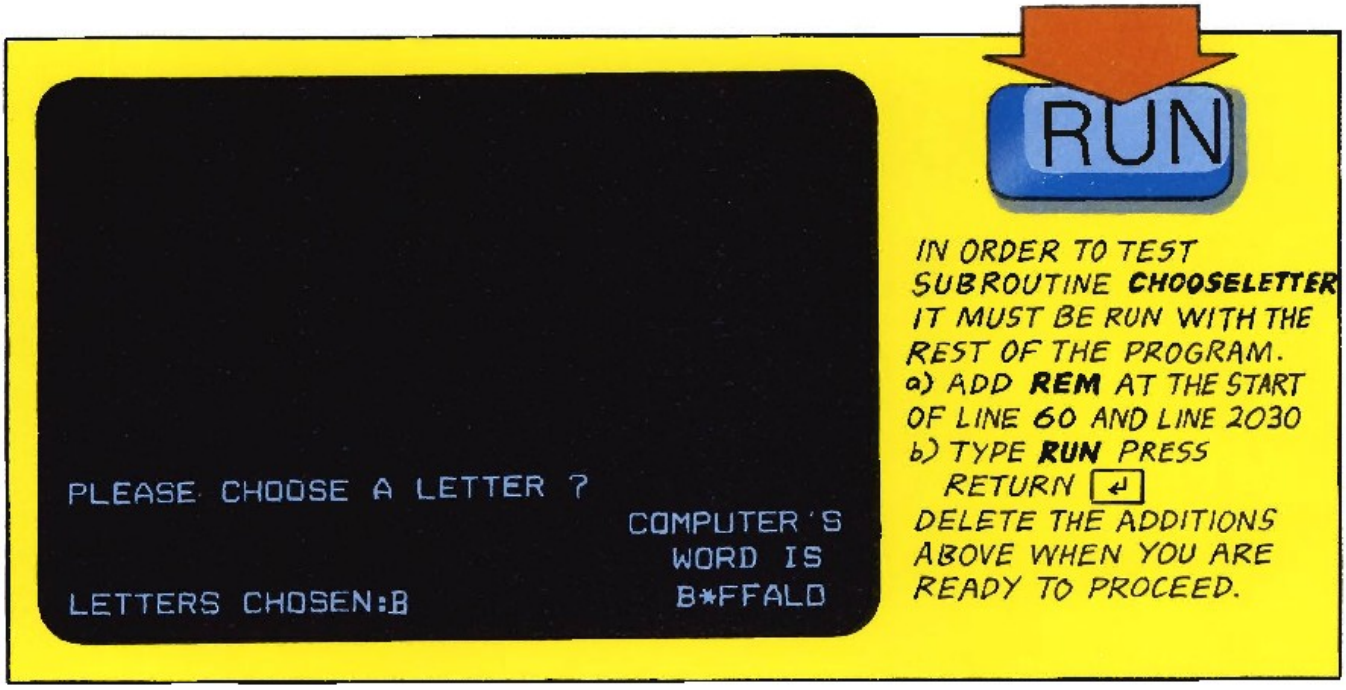


Once again, the subroutine is defined. Line 3020 asks for a letter to be **INPUT**, to be labelled as **C$** by the computer. A question mark will appear on the screen to prompt the player, followed by a flashing cursor. Line 3030 prints the letter chosen on the 24th row down the screen in the seventeenth column following the words "LETTERS CHOSEN:" at a position determined by the value of the variable **LI**. In fact all this means is that when the value of **LI** is 1 then the letter will be printed in the seventeenth column, and when life becomes two (on the second incorrect guess) then the letter chosen will be in the eighteenth column and so on for the ten guesses allowed in the game. At line 3040 the variable **CO** is set at zero so that the program goes to the *hangman* subroutine if a valid letter is not chosen. (Look back at line 60 of the main program to see why.)

```
3000  REM  CHOOSE LETTER
3010  VTAB 21: HTAB 24
3020  INPUT C$: IF  LEN (C$) <  > 1 THEN 3010
3030  VTAB 24: HTAB LI + 16: PRINT C$;
3040  LET CO = 0
3050  FOR P = 1 TO LE
3060  IF C$ <  > MID$ (W$,P,1) THEN 3100
3062  LET W1$ = "": LET W2$ = ""
3064  IF P <  > 1 THEN W1$ =  LEFT$ (W$,P - 1)
3066  IF LE <  > P THEN W2$ =  RIGHT$ (W$,LE - P)
3070  LET CO = 1: LET SC = SC + 1
3080  LET W$ = W1$ + "?" + W2$
3090  VTAB 24: HTAB P + 29: PRINT C$;
3100  NEXT P
3110  IF LE = SC THEN  LET WO = 1
3120  RETURN
```

Lines 3050 to 3100 make a **FOR. . . . . NEXT** loop which compares the letter **INPUT** with each letter in the mystery word in turn. For example, suppose that the mystery word is BOOK and the letter guessed K. **MID$** is used to copy part of a string. For the first value of **P** it would be the first letter, B. Since B is not equal to K, the computer runs through for the next value of Position, until **P** equals 4.

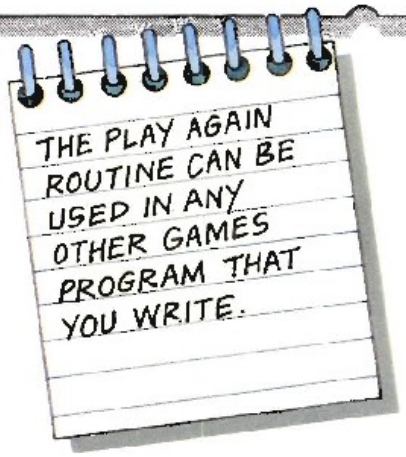
At this point, the value of **MID$** will be K. Since this *is* equal to the **INPUT**, the computer goes to line 3062, where the mystery word is changed. Applesoft will not allow **LEFT$** or **RIGHT$** to be zero as it will if **PO=1** or **PO=LE**. To avoid this line 3062 sets up temporary string variables **W1$** and **W2$**. The amount of the string copied again depends on the values of **P** and **LE** in brackets. In this example, line 3070 would change BOOK to BOO?. This makes sure that the player will lose a life if he or she guesses the same letter a second time. This is also why the original **W$** has to be stored as **D$** in line 1070 in the *selectword* subroutine. Finally, the value of **CO** is changed to 1 and **SC** is increased. In line 3090 the valid letter is printed on the screen overprinting an asterisk. Line 3110 checks if the value of **SC** is the same as the length of the word, in which case **WO** is given the value of 1 and the game is won.



The next subroutine, *win* is much simpler. It clears the screen and line 4030 calls up part of the hangman graphics. The congratulatory message is printed at the bottom

```
4000  REM  WIN
4010  HOME : HGR
4020  FOR L = 4 TO 10
4030  ON L - 4 GOSUB 7500,7600,7700,7800,7900,8000
4040  NEXT L
4050  VTAB 21: HTAB 1: PRINT "WELL DONE, YOUR
      EXECUTION"
4060  PRINT "HAS BEEN POSTPONED"
4070  RETURN
```

```
5000  REM   LOSE
5010  HOME : HGR
5020  FOR L = 1 TO 10
5030  ON L GOSUB 7100,7200,7300,7400,7500,7600,7700,
      7800,7900,8000
5040  NEXT L
5050  VTAB 21: HTAB 1: PRINT "KNAVISH FOOL!"
5060  PRINT "YOU LOST!"
5070  PRINT "THE WORD WAS ";D$
5080  RETURN
```



The subroutine for losing the game, comes into play after value of **LO** is set at 1 – this actually happens in the *hangman* subroutine later on. Again, the graphics which will be produced during the *hangman* subroutine are called up to give a good display. Notice that line 5070 also prints the original word selected at random by the computer, which was stored as **D$** in the *selectword* subroutine.

The final subroutine in this section of the book enables the player to play again. Line 6040 allows the "Y" key to be pressed. If the computer does not receive this **INPUT** the program runs through to line 6050, the screen will clear and the "BYE BYE" message will be displayed.

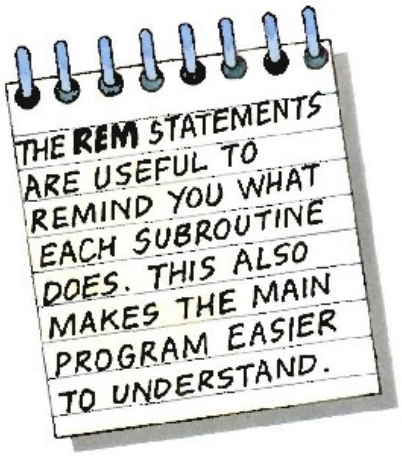
The program is now complete apart from the graphics section which is contained in the *hangman* subroutine. This is treated separately in the second section of the book. For those users without a disk drive at this point it is a good idea to save this section of program on tape. Connect the tape recorder up as instructed by the manual. Then type **SAVE** and press the play and record buttons. Then press **RETURN**. Make the recordings at different points on the tape, just in case the first recording doesn't work, as sometimes happens with programs stored on tape. Keep a note of the positions on the tape using the counter on your tape recorder.

```
6000  REM  PLAY AGAIN
6010  VTAB 24: HTAB 1: PRINT "DO YOU WISH TO PLAY
      AGAIN Y/N?";
6030  INPUT A$
6040  IF A$ = "Y" THEN 10
6050  HOME : VTAB 21: HTAB 16: PRINT "BYE BYE"
6060  RETURN
```

The main program for the Hangman game is very similar to the breakdown given in the introduction. Each logical step in the game is made into a subroutine. When the subroutines are placed in their logical order, and tests are included to see if the game has been won or lost, the main program is obtained.

```
10 PRINT"J"
20 GOSUB 1000: REM SELECTWORD
30 GOSUB 2000: REM TITLE
40 IF WO=1 OR LO=1 THEN 80
50 GOSUB 3000: REM CHOOSELETTER
60 IF CO=0 THEN GOSUB 7000: REM HANGMAN
70 GOTO 40
80 IF WO=1 THEN GOSUB 4000: REM WIN
90 IF LO=1 THEN GOSUB 5000: REM LOSE
100 GOSUB 6000: REM PLAY AGAIN
110 STOP
```

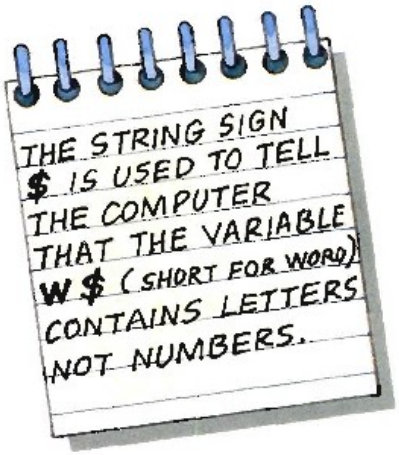
Let's look at the main program in more detail. It begins by going to a subroutine which allows the computer to choose a word at random from those stored in the **DATA** at the end of the program. Next a title screen is displayed at the beginning of the game. Line 40 tests to see if the game has been won or lost – this only comes into operation after a guess has been made. Line 70 returns the program to this point for another guess to be made. The subroutine called up in line 50 - *chooseletter* - allows the player to make a guess. At line 60, the program goes to the hangman routine which contains the graphics for the game. The value of **CO** is set at zero if an incorrect guess is made in the chooseletter routine. Finally, there are routines which come into play if the game is won or lost, and another which asks the player if he or she wants another game. In eleven lines you have the complete program for the game. All that remains to do is to write the routines – and that's when the hard work begins! Before you begin on these, add in the data statements below, so that you can test the program at various stages as you type it in. The computer will choose a word at random from this list during the course of the game. Of course, you don't have to use the words given here, but at this stage it's better to keep the number of words down to eight – you can add extra when the program is complete.




```
9000 DATA APPLE,ANGLE,ALPHABET,ASTEROID
9010 DATA BOTTLE,BOOK,BUFFALO,BEACH
```

```
1000 REM SELECTWORD
1005 UD$="SUBROUTINE"
1010 W=8
1020 RESTORE
1030 CH=INT(RND(0)*W)+1
1040 FOR L=1 TO CH
1050 READW$
1060 NEXT L
1070 LE=LEN(W$): D$=W$
1080 RETURN
```

The first subroutine allows the computer to choose a word from the list stored in the **DATA**. Line 1010 sets **W** to the number of words in the data list. **RESTORE** instructs the computer to **READ** from the beginning of the data each time it selects a word. Line 1030 generates a random number between one and **W**. The Commodore 64 generates random numbers between 0 and 1, so after this has been multiplied by the number of words in the **DATA**, it must be made a whole number using the **INT** command. The 1 is added to prevent the number ever being zero. The **FOR.....NEXT** loop of lines 1040 to 1060 gets the computer to **READ** through the **DATA** to the point indicated by the random number. For example, if the random number generated were four, the computer would **READ** through to the fourth word. This word is held in **W$** and is the mystery word. Line 1070 finds the length of the mystery word and also stores the mystery word under **D$**. **D$** is used if the game is lost when the computer displays the mystery word. Finally, **RETURN** sends the computer back to the main program.



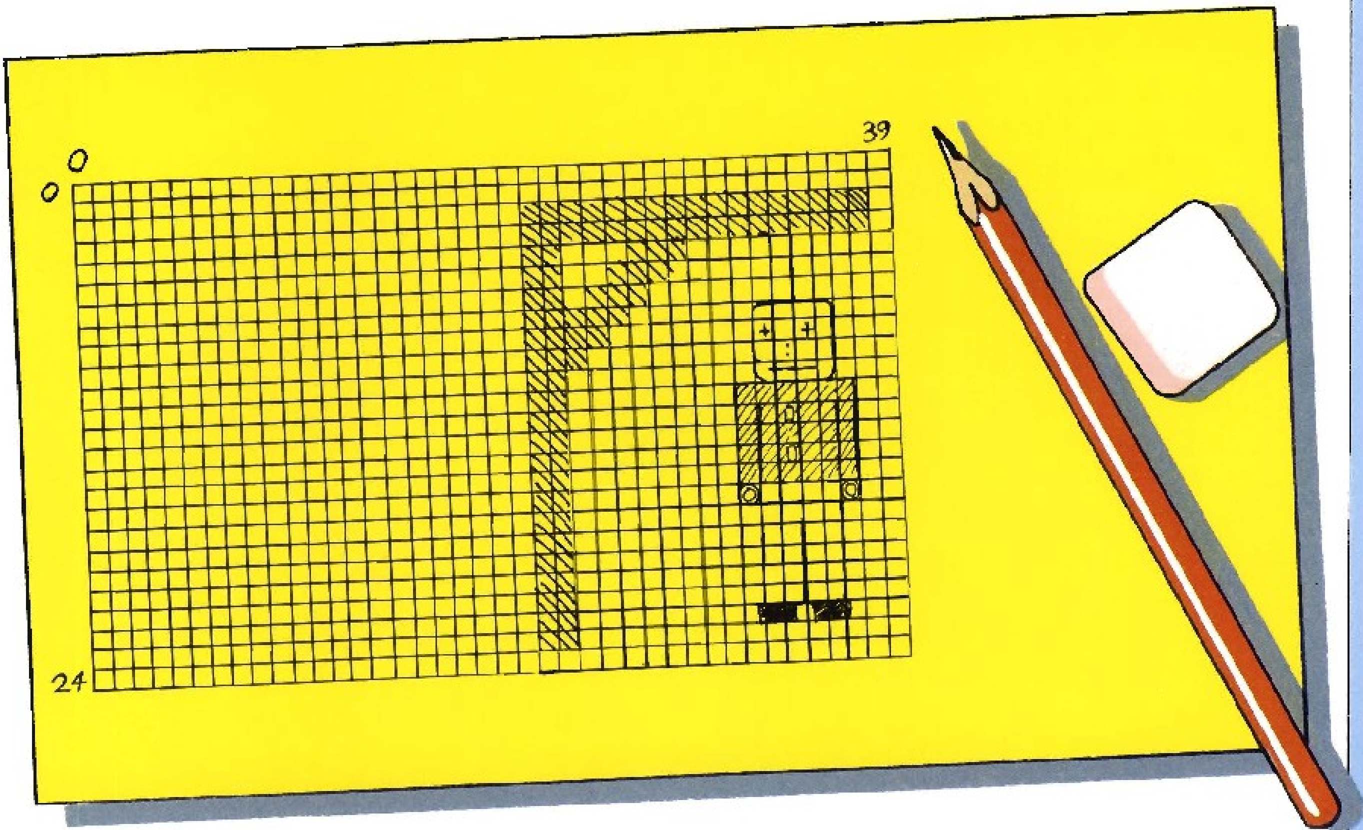

```
GOSUB 1000
ASTEROID

READY
```

TO TEST SUBROUTINE selectword:

1. TYPE **1075 PRINT W$**
   PRESS [RETURN]
2. PRESS [SHIFT] [CLR HOME]
3. TYPE **GOSUB 1000**
   PRESS [RETURN]

DELETE LINE 1075 WHEN YOU ARE READY TO PROCEED TO THE NEXT SECTION.

The next subroutine calls up the title screen. This will use some of the graphics created later in the hangman subroutine. These graphics appear on the right-hand side of the screen, so this should be kept in mind when deciding where to place the text instructions.



The title sequence has two parts. The first section, given below, gives the initial title screen, the second gives the screen for the beginning of the game. Line 2010 clears the screen. Lines 2020 to 2040 send the program to the appropriate section of graphics available in the *hangman* subroutine, stages four to ten. This will be explained later in that section of the program. In brief, the hangman image is built up in ten stages, one for each wrong guess or "life" lost. The **PRINT** command positions the word "HANGMAN" ten rows down the screen, beginning at the twelfth column across. The player is then instructed to press the **RETURN** key to start the game.

```
2000 REM TITLE
2010 PRINT"♥"
2020 FOR L=4 TO 10
2030 ON L-4 GOSUB7500,7600,7700,7800,7900,8000
2040 NEXT L
2050 PRINT LEFT$(UD$,10)SPC(12)"■HANGMAN"
2060 INPUT"QQQQQQQQQQQQQQ          PRESS ENTER";A$
```

```
2070 PRINT"■■■■■LETTERS CHOSEN"
2080 PRINT"■■■----------"
2090 PRINT"■■■■■■■■■■■■■■■THE COMPUTERS'S"
2100 PRINT"■■■■■■■WORD IS"
2110 PRINT"■■■■■■";
2120 FORL=1TOLE:PRINT"*";:NEXTL:PRINT
2130 PRINT"■■■■PLEASE CHOOSE"
2140 PRINT"■■■■■A LETTER"
2150 WO=0: LO=0: LI=0: SC=0
2160 RETURN
```

Lines 2070 to 2160 complete the title sequence. First the screen is cleared of the initial title screen and **PRINT** commands are used to display the next set of instructions. The letters chosen during the course of the game will be displayed under the heading at the top of the screen. The computer's selected word will be displayed half-way down. At line 2120, an asterisk is printed for every letter in the mystery word. This is done using another **FOR. . . . . .NEXT** loop, with figures ranging from 1 to the value for length obtained in the *selectword* subroutine. Finally, the initial values for some of the other variables used in the game, **WO, LO, LI,** and **SC** are set at zero. **WO** and **LO** are used in the main program to call up the appropriate subroutine for winning or losing a game. **SC** is used during the game to see if the mystery word has been correctly guessed, and **LI** is used to limit the game to ten guesses and to build up the hangman graphics in stages. Once again, you can check to see if you've got this part of the program right by following the instructions in the box below.

TO TEST SUBROUTINE titles:
1. TYPE LE=8 PRESS [RETURN]
2. ADD ***REM*** *AT THE START OF LINE 2030*
3. TYPE ***GOSUB 2000*** *PRESS* [RETURN]
DELETE THE ***REM*** *IN LINE 2030 BEFORE PROCEEDING TO THE NEXT SECTION.*

HANGMAN

PRESS RETURN?

LETTERS CHOSEN

THE COMPUTER'S

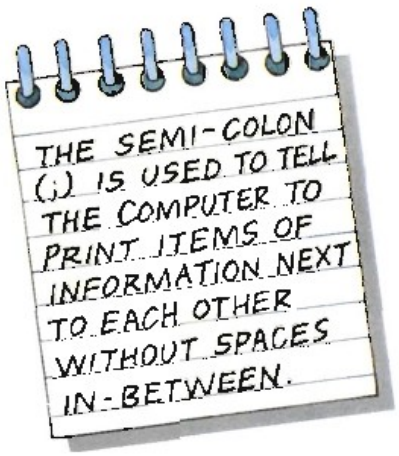
WORD IS

* * * * * * * *

PLEASE CHOOSE

A LETTER

The next subroutine does all the work of the program. It allows the player to choose a letter and checks to see if the letter is part of the mystery word. Appropriate changes are made to the value of **SC** and a check is made to see if the game is won. As you might expect, it's also the most difficult section of program.



Line 3010 moves the cursor to line twenty-three and changes the color to yellow. Three blank spaces cover over the old input and the cursor is moved back three spaces to the start of the line. **INPUTC$** causes a question mark and flashing cursor to appear on the screen to prompt the player. The second part of line 3020 makes sure that **C$** only contains one character. Line 3030 prints the guess on the fifth row down the screen. The variable **LI** is used so that the first guess is printed in the first space across, the second in the second space, and so on.

```
3000 REM CHOOSELETTER
3010 PRINTLEFT$(UD$,23)"■    ■■■";
3020 INPUTC$:C$=LEFT$(C$,1)
3030 PRINT"S■■■■■■■■■■■"SPC(LI)C$
3040 CO=0
3050 FOR P=1 TO LE
3060 IF C$<>MID$(W$,P,1) THEN 3100
3070 CO=1: SC=SC+1
3080 W$=LEFT$(W$,P-1)+"?"+RIGHT$(W$,LE-P)
3090 PRINTLEFT$(UD$,18)"■■■■■■■"SPC(P-1)C$
3100 NEXT
3110 IF SC=LE THEN WO=1
3120 RETURN
```

Line 3040 sets the value of **CO** at zero, so that the program goes to the *hangman* subroutine if a valid letter is not chosen. (Look back to line 60 of the main program). Lines 3050 to 3100 are a **FOR. . . . . .NEXT** loop which compares the letter guessed with each letter of the mystery word in turn. Let's take an example. Suppose the mystery word is BOOK, and the letter input "A". **P** would have values from one to four, the length of the word selected. **MID$** is used to copy part of the string. The instructions in brackets say which string and which part of the string to copy. For each of these, line 3060 would find that **C$** was not equal (<>) to **(W$, P, I)** so the program would run through to the end of the loop and return to the main program. If the guess was K, the program would run through the loop until **(W$, P, 4)**. Since this *is* equal to the input guess, the program would go on to line 3070 where the values of **CO** and **SC** are changed.

Line 3080 places a question mark inside the mystery word in place of the letter correctly guessed, so that if the player makes the same guess again he or she will lose a life. Line 3090 prints the correct guess in the correct place of the mystery word, and in line 3110 the computer compares the value of **SC** to the length of the word. If they are equal then the value of **WO** is made 1 and the computer returns to the main program. Line 80 of the main program will then operate the winning subroutine.

```
LETTERS CHOSEN
--------------

THE COMPUTER'S
WORD IS
B*FFALO

PLEASE CHOOSE
A LETTER
```

TO TEST SUBROUTINE chooseletter, IT MUST BE RUN WITH THE REST OF THE PROGRAM.

1. ADD ***REM*** *AT THE START OF LINE 60 AND LINE 2030.*

2. TYPE RUN PRESS RETURN

DELETE THE ***REM****'s IN LINES 60 AND 2030 BEFORE PROCEEDING TO THE NEXT SECTION.*

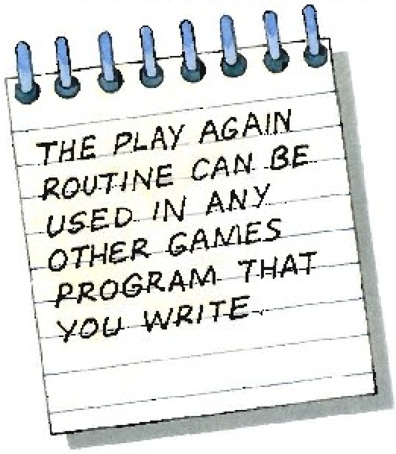
The *hangman* routine and the graphics it contains are given in the final section of the book. The graphics are used in the winning and losing subroutines that follow.

The *win* routine is very similar to that used to get the initial title display. The screen is cleared and the appropriate graphics – the man, in fact – are called up from the *hangman* routine. The congratulory message is printed on the seventh line using **LEFT$(UD$,7)** and the color white. You will remember that **UD$** was set up in line 1005 to move the cursor down by the number of rows given after the comma; in this instance 7.

```
4000 REM WIN
4010 PRINT"□"
4020 FOR L=5 TO 10
4030 ON L-5 GOSUB7500,7600,7700,7800,7900,8000
4040 NEXT
4050 PRINTLEFT$(UD$,7)"  ■WELL DONE, YOUR EXECUTION"
4060 PRINT"  HAS BEEN POSTPONED"
4070 RETURN
```

```
5000 REM LOSE
5010 PRINT"♥"
5020 FOR L=1 TO 10
5030 ON L GOSUB7100,7200,7300,7400,7500,7600,7700,
     7800,7900,8000
5040 NEXT
5050 PRINT"▓▓▓▓▓▓▓▓  KNAVISH FOOL !"
5060 PRINT"▓     YOU LOST"
5070 PRINT"▓▓▓▓THE WORD WAS▓":PRINTSPC(5)D$
5080 RETURN
```

The losing routine follows almost exactly the same structure as that for winning. In this case the whole of the hangman graphics appear on the screen. The original word selected by the computer and stored as **D$** in the *selectword* subroutine is displayed on the screen by line 5070. Notice that when you ask the computer to display a string variable, there is no need to place it in quotation marks.



The last subroutine is the one which asks the player if he or she wants to play another game. This is a routine which you could include in any other program that you might write. The **INPUT** asked for in line 6030 is checked in line 6040 to see if it is a "Y". If so, then the program returns to the beginning of the main program – line 10 – and the game begins again. Otherwise the screen clears and the computer says "Bye-Bye" in light red.

```
6000 REM PLAY AGAIN
6010 PRINT"▓▓  DO YOU WISH TO"
6020 PRINT"   PLAY AGAIN";
6030 INPUT A$
6040 IF LEFT$(A$,1)="Y" THEN 10
6050 PRINT"▓▓▓▓▓▓▓▓▓▓▓▓▓▓    BYE BYE"
6060 RETURN
```

Apart from the graphics contained in the *hangman* subroutine, the program is now complete. At this point it is a good idea to save the program on tape. Connect the tape recorder as instructed in the manual. Then type **SAVE "HANG1"** and press the **RETURN** Key. The Commodore 64 will respond by displaying the following message: **"PRESS PLAY AND RECORD ON TAPE"**. To check that the program has been safely recorded, rewind the tape and enter **VERIFY "HANG1"** and then press **RETURN**. If the recording is good, the computer will report "VERIFYING O.K.". If there are any errors, **SAVE** the program again.

Improve your program

Once you've typed in the program, you'll probably know the list of words in the **DATA** off by heart. Get a friend to type in a new set of data, or better still, take turns in typing in words while the other isn't watching. Keep a note of the number of words typed in and change the figure for the number of words in line 1010 accordingly

Adding sound

Apple

The Apple is capable of producing many varied sounds, but this requires more extensive explanation, and is beyond the scope of this book. However, a simple buzz or bell noise can be easily produced. In the addition below the computer is **PEEK**ing the memory address (−16336) a total of 300 times. This memory location is particular to the Apple loudspeaker, and the noise is added to the *lose* subroutine when the game is lost. Further additions are given in the full listing at the back of this book.

```
5072  FOR S = 1 TO 300
5074  LET T =  PEEK ( - 16336)
5076  NEXT S
```

Commodore 64

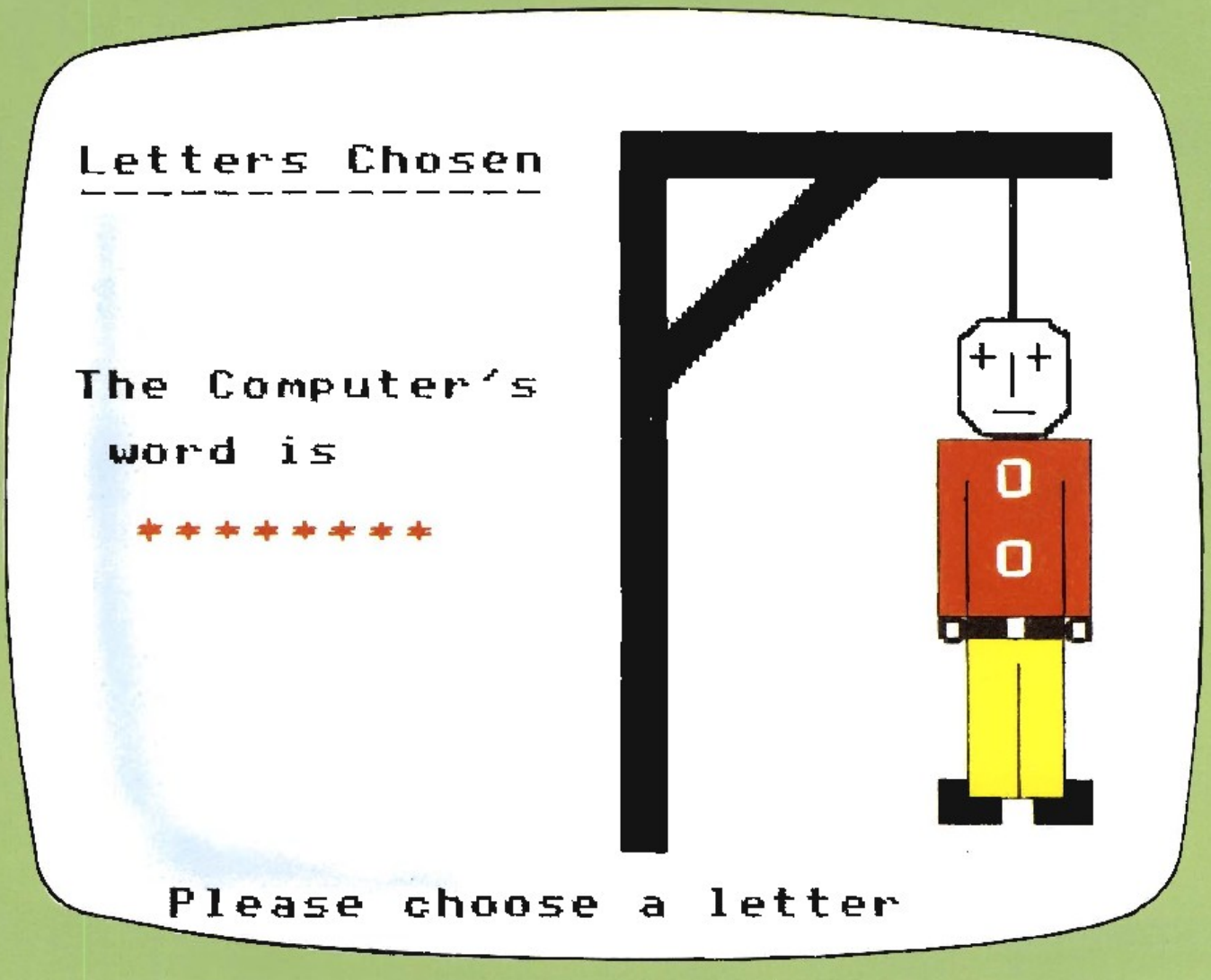
The Commodore 64 contains a powerful music synthesiser but to get the best from it is beyond the scope of this book. In the addition shown below the variable in the **FOR. . . . . NEXT** loop is being **POKE**d into the **HI** frequency register to make the pitch change (**POKE** 54273,I). The sound falls when the game is lost and rises when it is won. Both additions are given in the full listing at the back.
Consult your manual for more details.

```
4063 FORI=0TO24:POKE54272,0:NEXT
4067 POKE54277,15:POKE54278,240:POKE54296,15
4068 POKE54276,33:FORI=0TO255:POKE54273,I:NEXT
4069 POKE54276,32
```

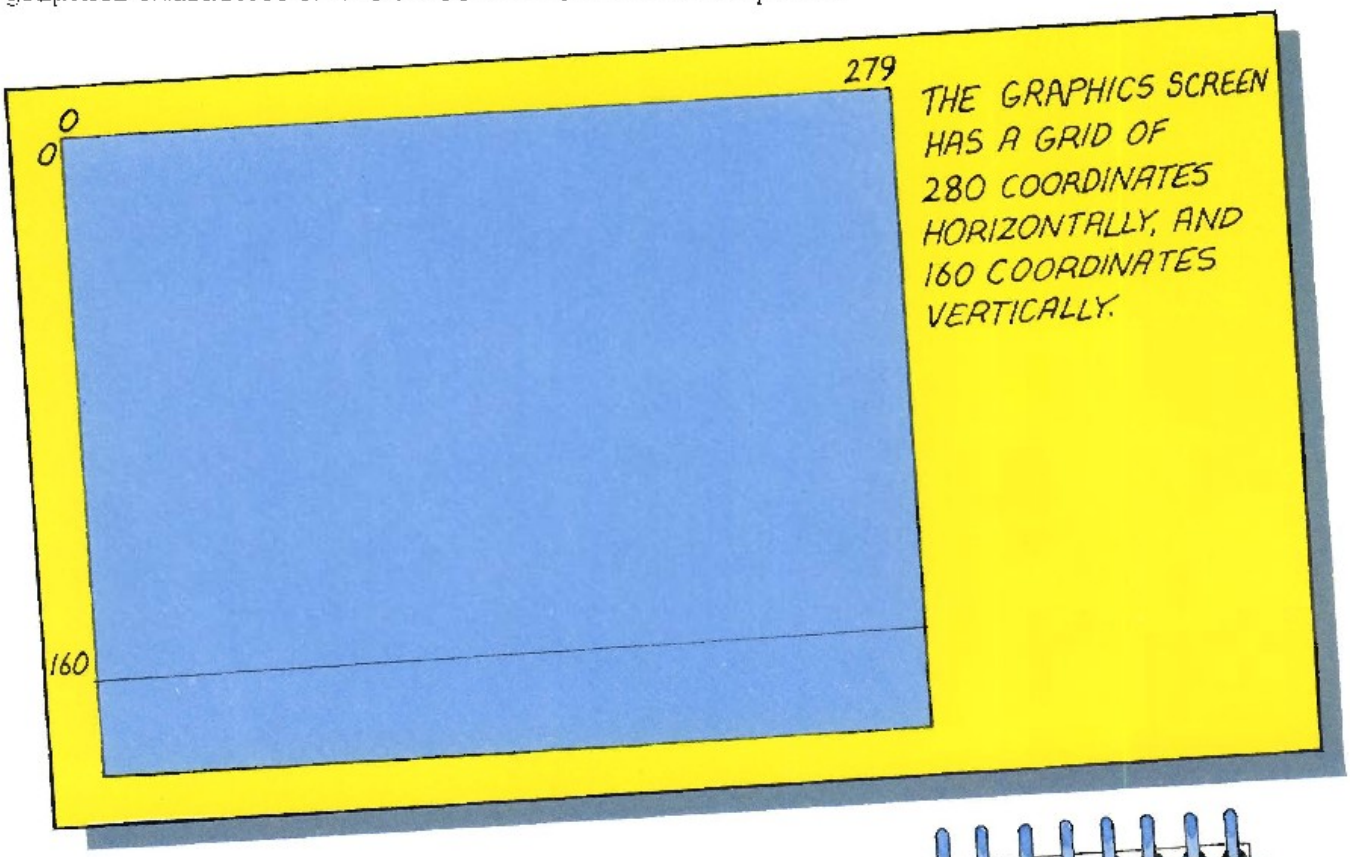
2

THE GRAPHICS

All that remains to have the complete Hangman game is to add in the Hangman graphics. These use all the graphics commands mentioned in the introduction to build up an image like the one shown below. In addition, there are hints on how you can make changes to the program to get a different result. The full listing for the game, including suggested additions is given at the end of this section.



All the graphics in the program are contained in the *hangman* subroutine. The graphics screen has a grid of 280 coordinates horizontally, and 160 coordinates vertically. These coordinates are used to plot the position of the graphics characters on the screen and are known as pixels.

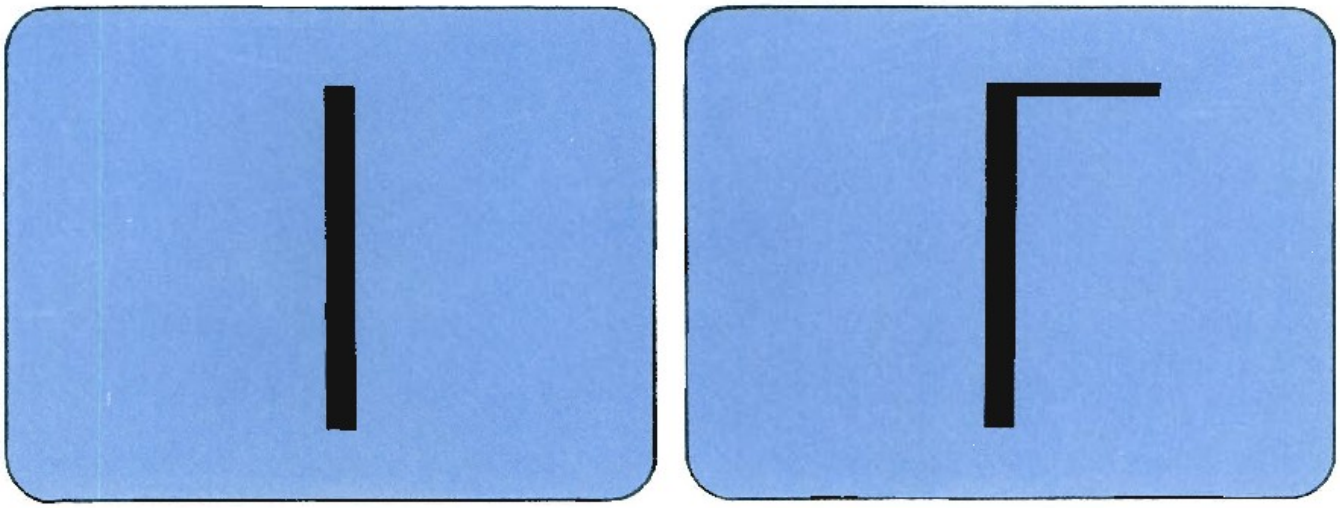


The hangman image is built up in stages, one for each time the player makes an incorrect guess. Line 7010 does this by increasing the value of **LI** by one each time the subroutine is called up. Line 7020 sends the program to the relevant section of graphics for each life lost. On the first life lost, the upright post of the gallows is drawn. **HCOLOR** sets the graphics color - in this case green on a black screen. A **FOR. . . . .NEXT** loop used **HPLOT** to place an element on the screen at the given X, Y coordinates and draws a line from this point to the second set of coordinates.

THE INSTRUCTION **ON L GOTO** CAUSES THE COMPUTER TO JUMP TO A NEW SECTION OF GRAPHICS FOR EACH NEW VALUE OF **L**

```
7000  REM  HANG MAN
7010  LET LI = LI + 1: IF LI = 10 THEN LO = 1
7020  ON LI GOTO 7100,7200,7300,7400,7500,7600,7700,
      7800,7900,8000
7100  REM  GALLOWS 1
7110  HCOLOR= 1
7120  FOR R = 180 TO 170
7130  HPLOT R,160 TO R,14
7140  NEXT R
7150  RETURN
```

The gallows image should be on the right-hand side of the screen. The coordinates used place the gallows in the position shown in the diagrams below. The **HPLOT** command is used to draw the lines The loop variable **R** gives the first X coordinate across – in this case 180 – to determine where it shall appear horizontally. A line is drawn from this position to **R,14**. This system of line drawing is repeated in increasing values of **R** in a **FOR. . . . .NEXT** loop.



The diagram on the left, above, shows the first part of the gallows. The second part of the *hangman* subroutine uses the same set of commands to draw the second part of the gallows. This is shown in the diagram on the right. The bar is a five line loop placed on top of the first stem. It's a good idea to note down the various positions on graph paper, so that you can keep track of where you are on the screen.

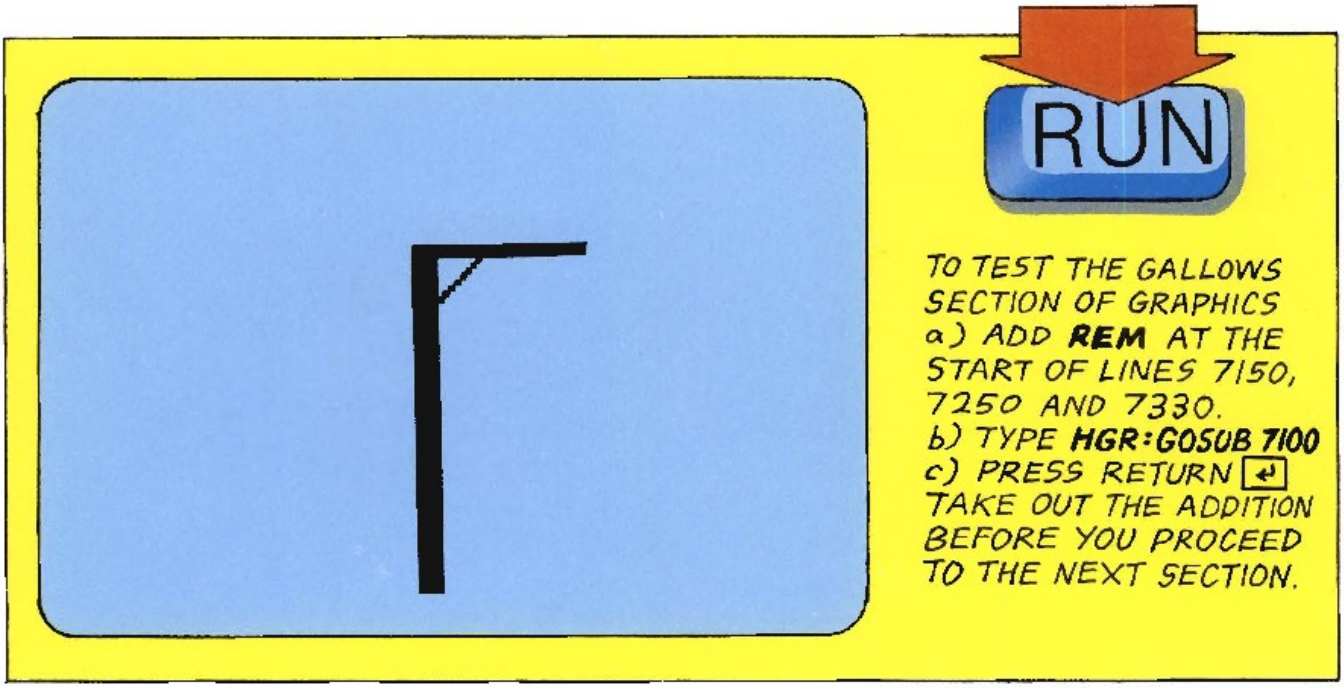
```
7200   REM   GALLOWS 2
7210   HCOLOR= 1
7220   FOR R = 10 TO 14
7230   HPLOT 180,R TO 250,R
7240   NEXT R
7250   RETURN
```

When the third life is lost the computer draws the third section of the gallows. This is the cross-bar between the first two beams. It is made up of a single diagonal using the **HPLOT** instruction.

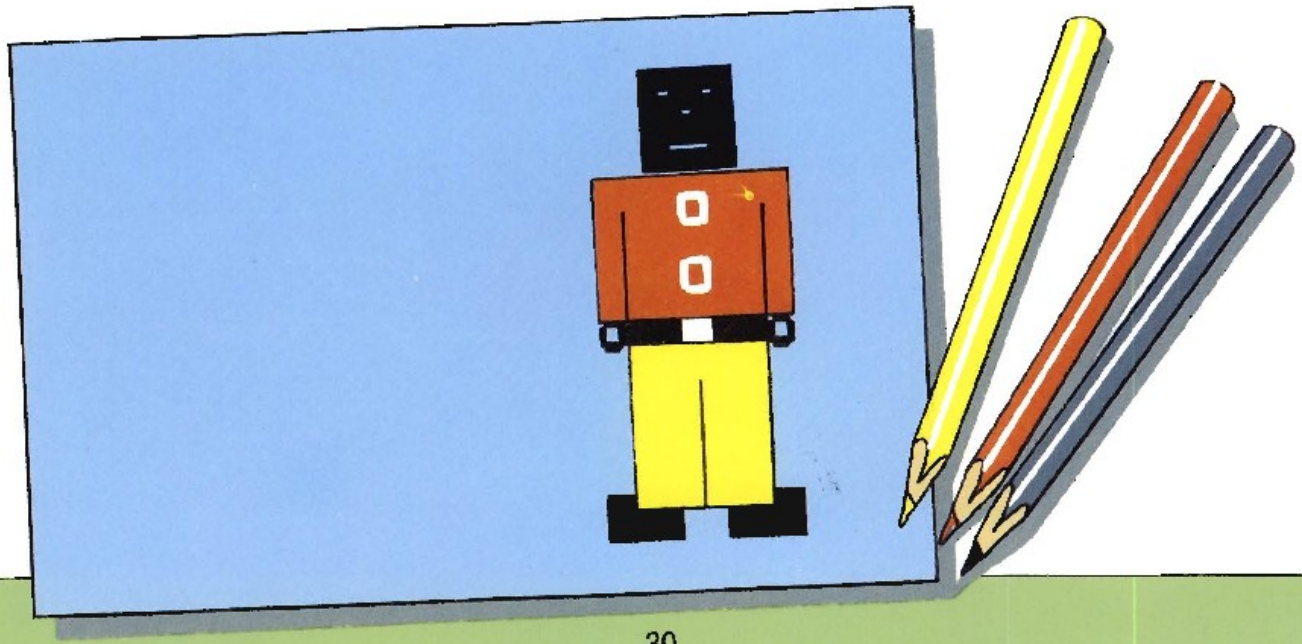
```
7300   REM   GALLOWS 3
7310   HCOLOR= 1
7320   HPLOT 190,30 TO 206,14
7330   RETURN
```

```
7400  REM  ROPE
7410  HCOLOR= 2
7420  HPLOT 234,29 TO 234,15
7430  RETURN
```

Next, a rope is added for the fourth life lost. This is simply a line drawn from the middle of the gallows extending downwards for 14 vertical coordinates. To test the graphics written so far, follow the instructions in the box below.



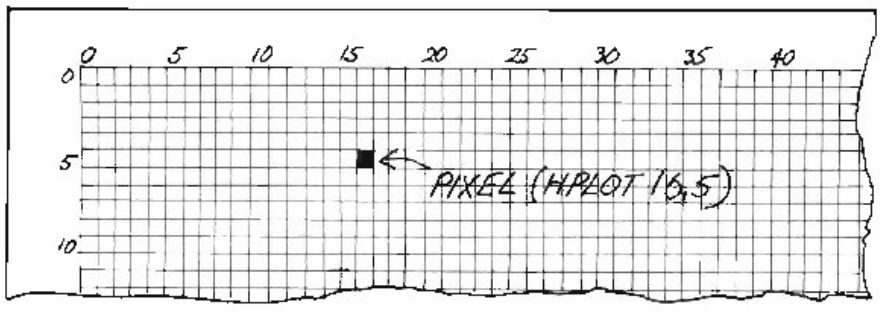
The next stage of the *hangman* subroutine will draw the man himself. In **HGR**, there are only eight colors available: we will use six. The man will be drawn up as shown in the box below, but you could plan out your own design. To complete the ten lives for the game, the hangman is built up in six stages.



The first stage of drawing the man is the face. This is a block of color made up of straight vertical lines drawn close together using a loop in lines 7510 to 7530. The color is changed to draw the eyes still using the **HPLOT** command. Line 7550 draws a straight line for the mouth and line 7570 draws the nose. The eyes are drawn in line 7560 as two straight lines.

```
7500  REM  FACE
7510  HCOLOR= 3: FOR I = 225 TO 245
7520  HPLOT I,49 TO I,30
7530  NEXT I
7540  HCOLOR= 0
7550  HPLOT 230,45 TO 240,45
7560  HPLOT 228,34 TO 230,34: HPLOT 240,34 TO 242,34
7570  HPLOT 234,38 TO 236,38
7580  RETURN
```

The body of the man is drawn in red. **HCOLOR= 5** gives red on certain TVs. Values of I from 220 to 250 set the X coordinates for the **HPLOT** command to act upon, filling in with the color already specified. To add a bit more detail, the man is given coat buttons. The color is set at black, the same used for the details on the face. Line 7650 places the buttons horizontally on coordinates 235 and vertically on coordinates 65 and 85. You may wish to add more buttons, or give the man pockets. It is easier to draw the effect on graph paper first to see which pixels to choose before programming. The pixels are those small squares similar to your graph paper. An accurate plan can be made to scale by counting 0-279 horizontally and 0-159 vertically.




```
7600  REM  BODY
7610  HCOLOR= 5
7620  FOR I = 220 TO 250
7630  HPLOT I,90 TO I,50
7640  NEXT I
7650  HCOLOR= 0: HPLOT 235,65: HPLOT 235,85
7660  RETURN
```

The next section of graphics draws the man's arms. These are again in red and are boxes with hands on the end. The position is chosen, then a loop sequence is set up to fill in the rectangle. The hands are represented in the same color as the face. Line 7760 draws in the hands using the **HPLOT** command.

```
7700  REM  ARM 1
7710  HCOLOR= 5
7720  FOR I = 213 TO 218
7730  HPLOT I,55 TO I,105
7740  NEXT I
7750  HCOLOR= 3
7760  HPLOT 214,106 TO 214,110 TO 218,110 TO 218,106
7770  RETURN
```

The second arm is built up in exactly the same way as the first: only the horizontal coordinates change to position the arm on the other side of the body. **HCOLOR** in line 7810 sets the graphics color back to red.

```
7800  REM  ARM2
7810  HCOLOR= 5
7820  FOR I = 252 TO 257
7830  HPLOT I,55 TO I,105
7840  NEXT I
7850  HCOLOR= 3
7860  HPLOT 252,106 TO 252,110 TO 256,110 TO 256,106
7870  RETURN
```

The first leg and foot are built up by the next section of the graphics, which are displayed for the ninth life lost. First the color of the trousers is chosen – in this case **HCOLOR** = 6. Then the loop is set up using **FOR. . . . .NEXT** and **HPLOT** draws in the color. The shoe is built up in the same manner.

```
7900  REM  LEG1
7910  HCOLOR= 6
7920  FOR I = 220 TO 232
7930  HPLOT I,100 TO I,140
7940  NEXT I
7950  HCOLOR= 7
7960  FOR I = 141 TO 148
7970  HPLOT 212,I TO 228,I
7980  NEXT I
7990  RETURN
```

```
8000  REM  LEG2
8010  HCOLOR= 6
8020  FOR I = 238 TO 250
8030  HPLOT I,100 TO I,140
8040  NEXT I
8050  HCOLOR= 1
8060  FOR I = 92 TO 98
8070  HPLOT 220,I TO 250,I
8080  NEXT I
8090  HCOLOR= 7
8100  FOR I = 141 TO 148
8110  HPLOT 242,I TO 258,I
8120  NEXT I
8130  REM  DELAY
8140  FOR D = 1 TO 1000: NEXT D
8150  RETURN
```

HPLOT X,Y PLACES A DOT AT X,Y COORDINATES.
HPLOT X_1,Y_1 TO X_2,Y_2 PLOTS A LINE FROM X_1,Y_1 TO X_2,Y_2

The final life lost gives the man's other leg and also adds a belt. The second leg follows exactly the same steps as the first, with only the coordinates differing. The belt is made by drawing a series of seven straight lines using the **FOR.....NEXT** loop. You will see that the man receives his belt which is the same color as the gallows, before his foot is drawn. All that remains in this section is to add a small delay so that the computer doesn't run straight back to the main program, and this is achieved in lines 8130 and 8140.

The hangman program is now complete and ready to run. You should have the hangman graphics on the title screen and when you win or lose the game. At the end of the book the listing is given in full and there are some suggested additions you can make to your program.

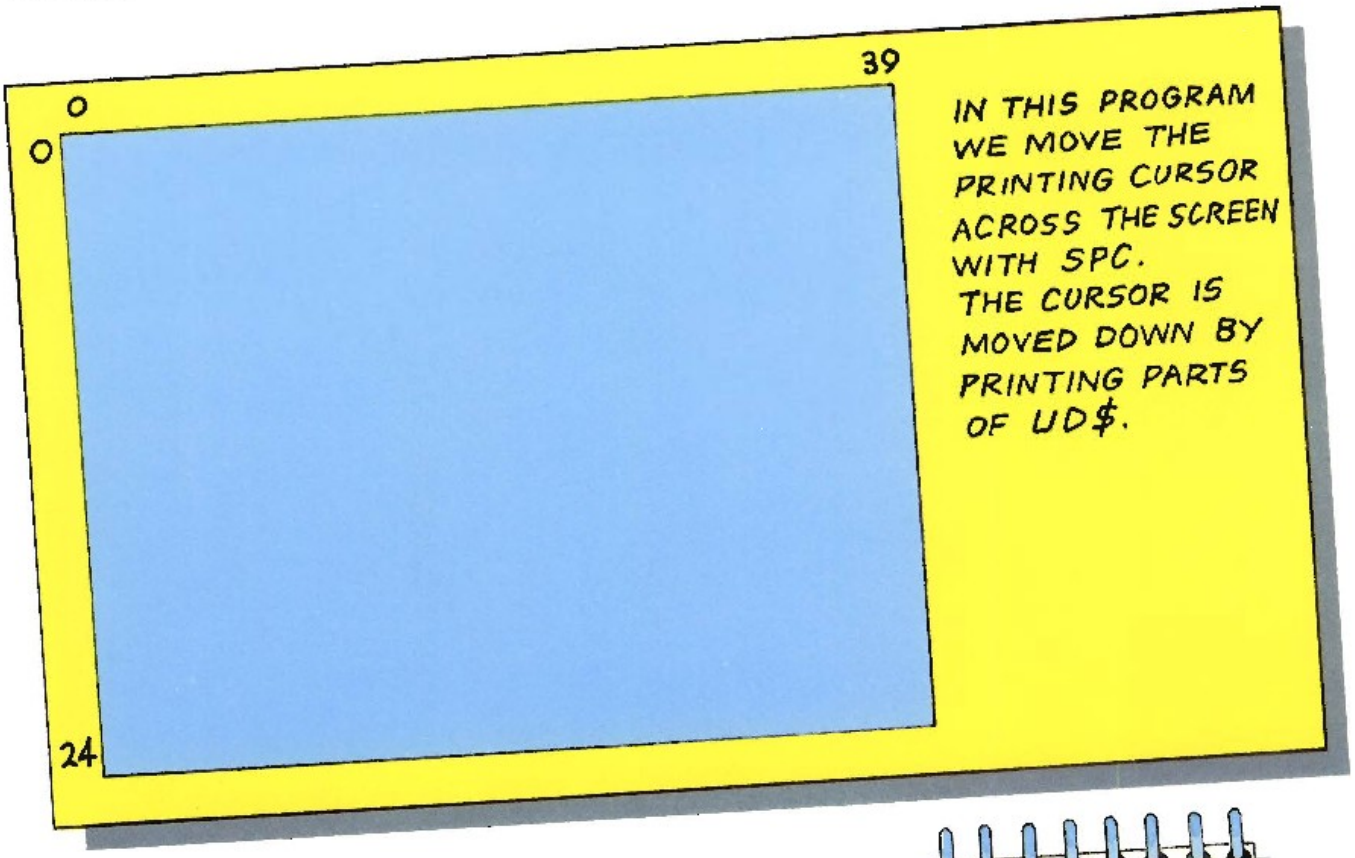
WHEN THE PROGRAM IS RUN THE SCREENS SHOULD SHOW AS BELOW WHEN YOU:

START

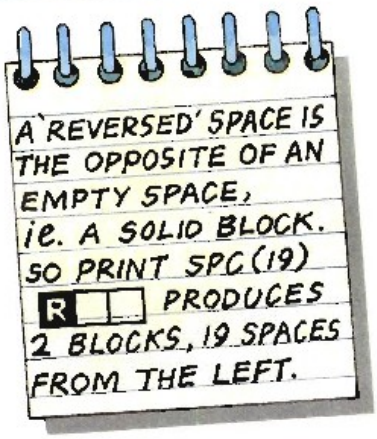
WIN

LOSE

The Commodore 64 text screen is divided into 40 rows across the screen and 25 vertical columns. To move the printing position around the screen use **UD$** (or parts of it) to move the cursor down. To move the cursor across the screen use **SPC**.

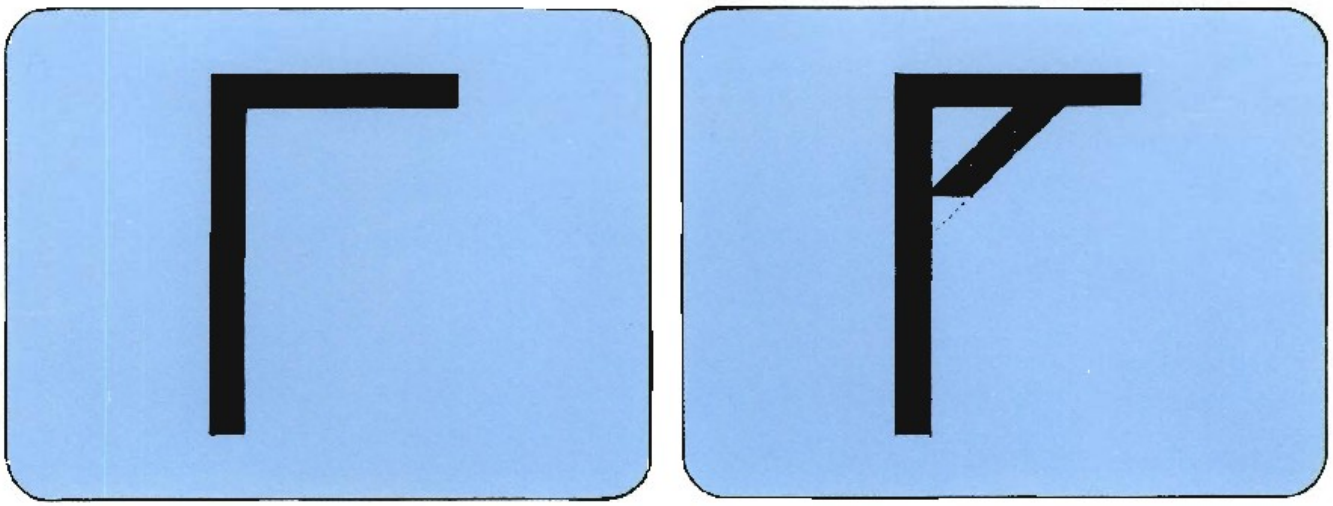


The *hangman* subroutine is divided into ten stages, each stage being displayed when a wrong guess is made and a life is lost. In line 7010 the value of **LI** is increased by one each time the routine is visited, and a check is made to see if it has reached the value of ten, in which case the game is lost. Line 7020 directs the program to the appropriate section of the graphics, using the new value of **LI**. The first section of graphics builds the gallows post, using the simple block graphics obtained by **REVERSE ON** and two spaces. The reverse of a space is a block and can be used with any color.




```
7000 REM HANGMAN
7010 LI=LI+1 :IF LI=10 THEN LO=1
7020 ON LI GOTO 7100,7200,7300,7400,7500,7600,7700,
     7800,7900,8000
7100 REM GALLOWS 1
7110 PRINT"■■"
7120 FOR R=0 TO 21
7130 PRINTSPC(19)"■  "
7140 NEXT
7150 RETURN
```

The three symbols on line 7110 first home the cursor, move it down one and set the color to brown. The cursor home is represented by the reversed "S". The cursor down is represented by the reversed "Q" and the brown is obtained with 2. The **FOR. . . .NEXT** loop sets out the row values to run through from 0 to 21 and prints two solid blocks starting in column 19. To check this first section of the graphics type **GOSUB7100**.



The second step of the graphics is to add the horizontal bar of the gallows. It uses exactly the same method as for building the post. Lines 7220 to 7240 produced a block eleven spaces wide covering two rows.

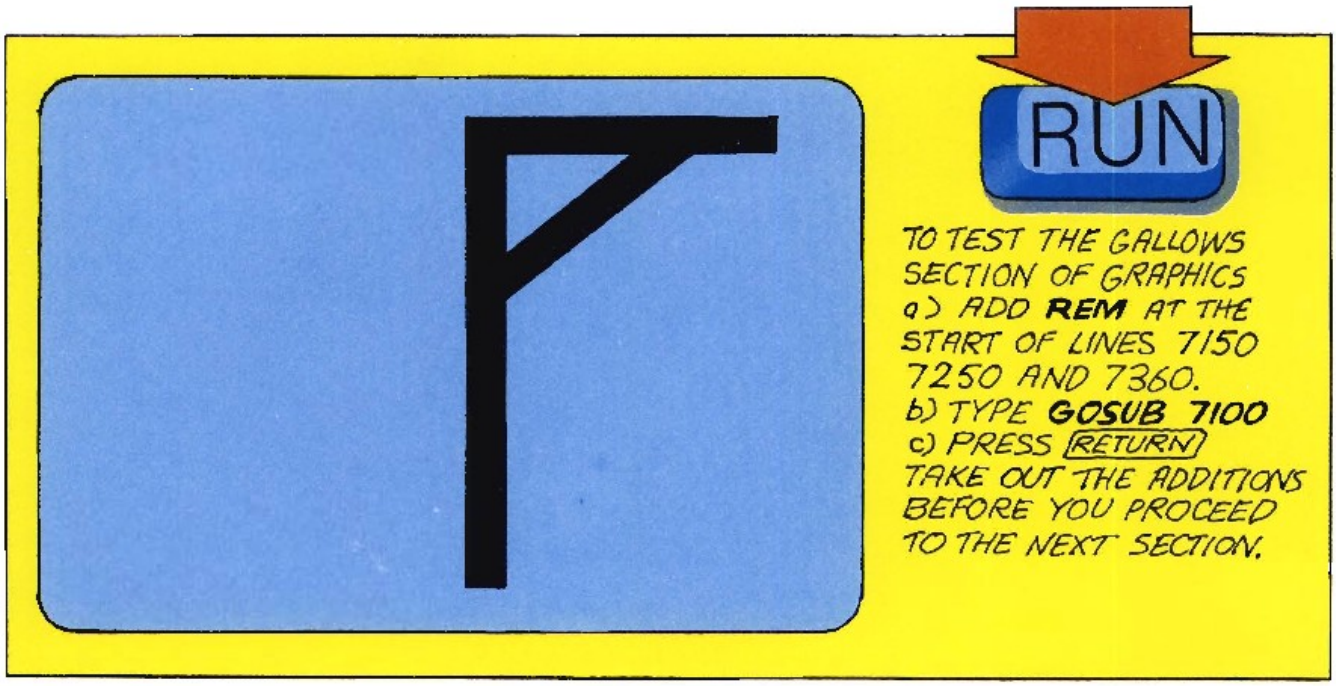
```
7200 REM GALLOWS 2
7210 PRINT"
7220 FOR R=0 TO 1
7230 PRINTSPC(21)"
7240 NEXT
7250 RETURN
```

The third stage is to add the cross-bar which joins the first two parts together. The **FOR. . . . .NEXT** loop creates a variable in steps of −1. A parallelogram is drawn from the top of the cross bar until the upright gallows is reached. The triangle used to make each half of the parallelogram is a shifted "£" sign. Line 7350 adds the last triangle.

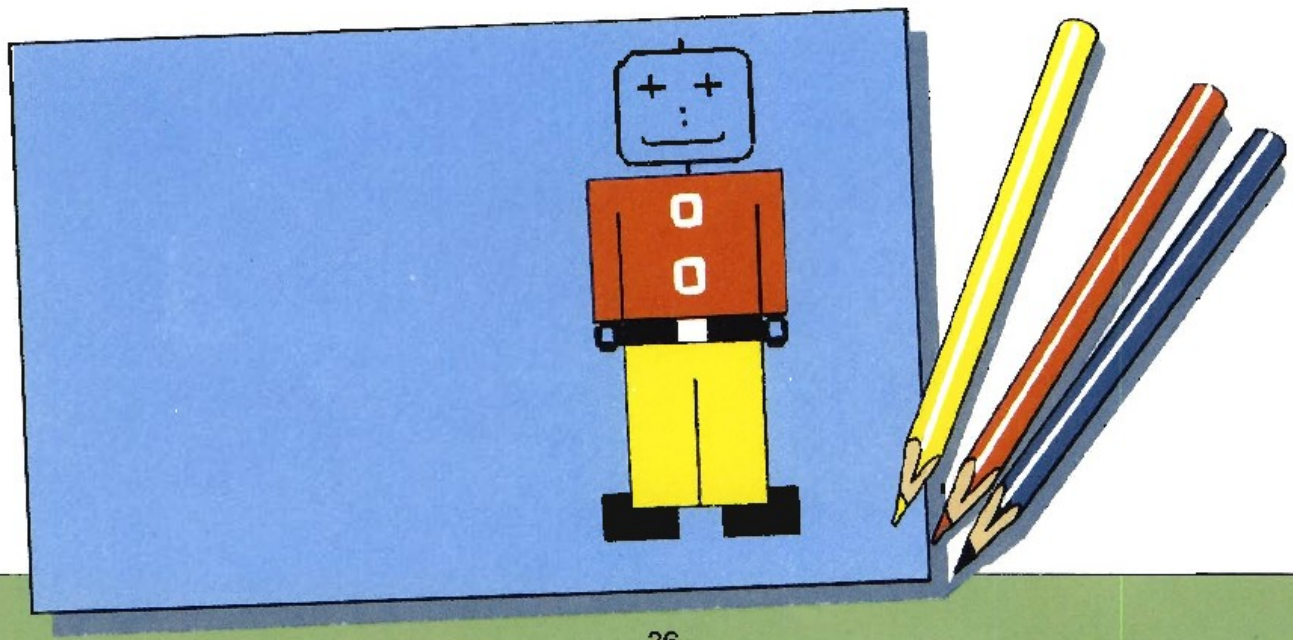
```
7300 REM GALLOWS3
7310 PRINT"
7320 FOR R=2 TO 0 STEP-1
7330 PRINTSPC(21+R)"
7340 NEXT
7350 PRINTSPC(21)"
7360 RETURN
```

```
7400 REM ROPE
7410 PRINT"SHIFT-MINUS"
7420 FORR=0TO1:PRINTSPC(29)"I":NEXT
7430 RETURN
```

The fourth stage is to add the rope. It is made from one "shifted minus" graphic on top of another using a **FOR . . . .NEXT** loop. The loop is repeated once and the line appears in the twenty-ninth column. If you **RUN** the program you should have the completed gallows on the screen.



The next section of the graphics will draw the man. Before you begin, it's a good idea to plan out your design as in the box below. At this stage you can decide which colors you want to use – there are sixteen available on the Commodore 64, including black and white. The graphics for the man is built up in six stages.

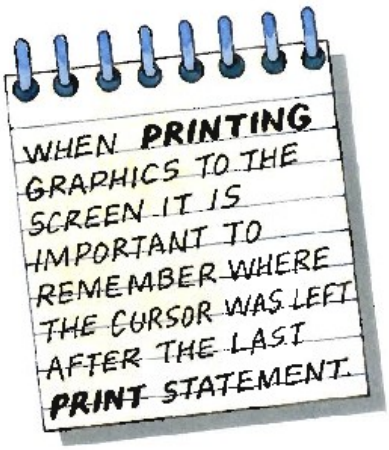


The first section of the man is the face. You will remember that we have chosen to move the cursor round the screen rather than print the exact positioning and colors on the full graphics chart using **POKE**. For that we would need to program both the screen and color memory maps.

The top of the face is made using the graphics on keys U, Ē and I. The cheeks are made by the **FOR. . . .NEXT** loop with the "shifted minus" key (remember how you made the rope). The graphics for the bottom of the face are J, R̄ and K.

```
7500 REM FACE
7510 PRINT"■■■■■■■"
7520 PRINTSPC(27)"┌─┴─┐"
7530 FORR=0TO2:PRINTSPC(27)"│   │":NEXT
7540 PRINTSPC(27)"└─┬─┘":PRINT"   "
7550 PRINTSPC(28)"+ +":PRINTSPC(29)":":
     PRINTSPC(28)"──"
7560 RETURN
```

The details of the face are created by one line in the program – line 7550 - which gives two plus's for the eyes, a colon for the nose and a line for the mouth. To place these in position the cursor has been moved up four rows at the end of the 7540. Unless a print statement ends with a semi-colon or a comma, the cursor moves down one line. Therefore, in line 7550, the features of the face appear under one another even though the commands are on the same line.



The next subroutine builds the body. The color (light red) and position are set in line 7610. The **FOR. . . .NEXT** loop prints a rectangle three blocks wide and five rows deep in the same manner as the gallows. Line 7630 moves the cursor up four rows and prints a reversed "O". Another reversed "O" is printed two rows below that. This produces two round buttons on the hangman's coat. If you wanted to, you could use a similar command to give the hangman pockets by using two subtraction symbols. It's a good idea to run the program at this point, to check that you've positioned the body correctly.

```
7600 REM BODY
7610 PRINT"■■■■■■■■■■■■■"
7620 FORR=0TO4:PRINTSPC(28)"■■■":NEXT
7630 PRINT"[UP][UP][UP][UP]"SPC(29)"■":PRINTSPC(29)"■"
7640 RETURN
```

For the next life lost an arm is added to the hangman. This is made in three parts, shoulder, arm and hand. The shoulder is made from two blocks (reversed spaces) on top of each other in line 7720. To separate the arm from the body a block with a bit of space missing on the right hand side is needed. N̄ produces a line on the right, so by printing it in reverse the required space is formed. Line 7730 gives a three line loop which makes the arm. The hand is represented by an "O".

```
7700 REM ARM1
7710 PRINT"SQQQQQQQQQQX"
7720 FORR=0TO1:PRINTSPC(27)"R ":NEXT
7730 FORR=0TO2:PRINTSPC(27)"R |":NEXT
7740 PRINTSPC(27)"o"
7750 RETURN
```

The next life lost produces the second arm. This is created in exactly the same way as the first. Only the number in **SPC( )** changes, to place the arm on the right-hand side of the body. The H̄ is used to separate this arm.

```
7800 REM ARM2
7810 PRINT"SQQQQQQQQQQX"
7820 FORR=0TO1:PRINTSPC(31)"R ":NEXT
7830 FORR=0TO2:PRINTSPC(31)"R ":NEXT
7840 PRINTSPC(31)"o"
7850 RETURN
```

The next part of the hangman to be added is the legs. The first leg is created by a six line loop using **REVERSE ON** space and **REVERSE OFF** K̄ and is displayed when the ninth life is lost. These will occupy the one and a half spaces that are available for each leg – the body is made up of three spaces and each leg has to be half this width. The command given will produce light green legs, but, of course, you can produce any color that you wish by using another of the color codes. The legs are drawn starting at row fifteen. Line 7910 could be stated as **PRINT LEFT$ (UD$, 16)** to place the leg on the sixteenth row. This is to avoid counting a large number of cursor down symbols which could lead to errors. The color code would have to be placed in inverted commas.

```
7900 REM LEG1
7910 PRINT"SQQQQQQQQQQQQQQQ■I"
7920 FORR=0TO5:PRINTSPC(28)"R ■":NEXT
7930 RETURN
```

```
8000 REM LEG2
8010 PRINT"SQQQQQQQQQQQQQQQQ■"
8020 FORR=0TO5:PRINTSPC(28)"R | ":NEXT
8030 PRINTSPC(28)"RIIIIIIIIR * "
8040 PRINTSPC(28)"■QQQQQQQ‾ ‾"
8050 REM DELAY
8060 FOR D=1 TO 1000: NEXT D
8070 RETURN
```

The second leg is drawn by overprinting the first leg. The legs are printed as a large block with a line up the middle. This line is produced by using the "shifted minus" key. Remember line 8020 is all in reverse, so the line appears as a gap.

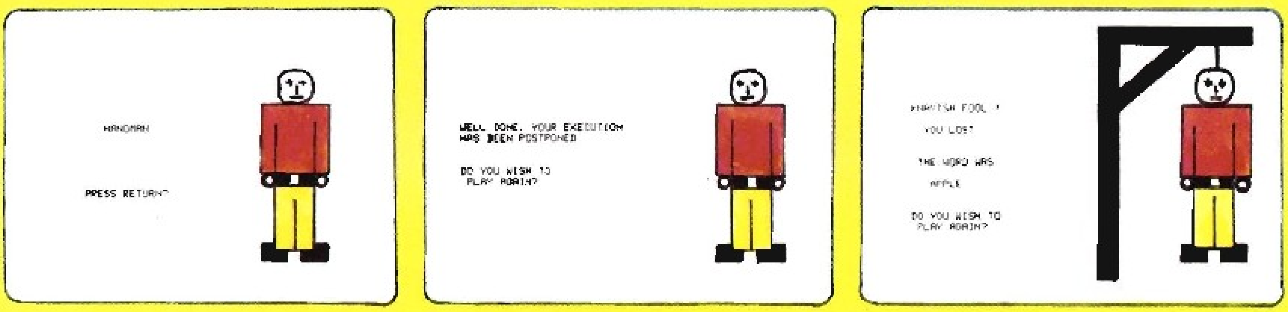
For this last stage of the graphics, we will also add a belt and shoes to give the hangman detail. The belt is colored medium gray. It is produced by using the **REVERSE ON** space with the ordinary asterisk, to act as the buckle. The shoes are printed in brown at the end of each leg by using the Ū key. Finally, there is a short delay at the end of this section before the computer returns to the main program.

THE DELAY LOOP IN LINE 8130 TELLS THE COMPUTER TO COUNT FROM ONE TO ONE THOUSAND BEFORE RETURNING TO THE MAIN PROGRAM.

With this last section, the program is now complete and ready to **RUN**. Because the hangman graphics are called up in the subroutine for the title, the hangman should appear on your screen when you type **RUN**. The graphics are also called up when the game is won or lost. At the end of the book the program listing is given in full, and there are some suggested additions to make the game more interesting. There are noises and mouth movements. For example, when you win the hangman smiles and when you lose his mouth turns downward!

WHEN THE PROGRAM IS RUN THE SCREENS SHOULD SHOW AS BELOW WHEN YOU:

START **WIN** **LOSE**



Improving your program (2)

Now that you have the complete program typed in you should save it on tape. The method for storing programs is given in the user's manual. Why not try to build completely different graphics?

Apple

The program listing below gives the complete Hangman game. Those program lines marked with an asterisk (*) are additional to the program given in the main part of the book.

Commodore 64

Given below is the complete program for the Hangman game for the Commodore computer. There are additional lines marked with an asterisk (*) not given in the main part of the book. These give added detail to the game.

```
  10  HGR : HOME
  20  GOSUB 1000: REM  SELECTWORD
  30  GOSUB 2000: REM  TITLE
  40  IF WO = 1 OR LO = 1 THEN 80
  50  GOSUB 3000: REM  CHOOSELETTER
  60  IF CO = 0 THEN  GOSUB 7000: REM  HANGMAN
  70  GOTO 40
  80  IF WO = 1 THEN  GOSUB 4000: REM  WIN
  90  IF LO = 1 THEN  GOSUB 5000: REM  LOSE
 100  GOSUB 6000: REM  PLAY AGAIN
 110  STOP
1000  REM  SELECT WORD
1010  LET W = 8
1020  RESTORE
1030  LET CH =  INT ( RND (1) * W) + 1
1040  FOR L = 1 TO CH
1050  READ W$
1060  NEXT L
1070  LET LE =  LEN (W$): LET D$ = W$
1080  RETURN
2000  REM  TITLE
2010  HGR : HOME
2020  FOR L = 4 TO 10
2030  ON L - 4 GOSUB 7500,7600,7700,7800,7900,8000
2040  NEXT L
2050  VTAB 21: HTAB 14: PRINT "H A N G M A N"
2060  VTAB 23: HTAB 10: INPUT "PRESS RETURN TO START"
      ;A$
2070  HOME : HGR : VTAB 24: HTAB 1: PRINT "LETTERS
      CHOSEN:";
2090  VTAB 22: HTAB 28: PRINT "COMPUTER'S"
2100  VTAB 23: HTAB 30: PRINT "WORD IS"
2110  VTAB 24: HTAB 30
2120  FOR L = 1 TO LE: PRINT "*";: NEXT L
2130  VTAB 21: HTAB 1: PRINT "PLEASE CHOOSE A LETTER"
2150  LET WO = 0: LET LO = 0: LET LI = 0: LET SC = 0
2160  RETURN
3000  REM  CHOOSE LETTER
3010  VTAB 21: HTAB 24
3020  INPUT C$: IF  LEN (C$)   > 1 THEN 3010
3030  VTAB 24: HTAB LI + 14: PRINT C$;
3040  LET CO = 0
3050  FOR P = 1 TO LE
3060  IF C$      MID$ (W$,P,1) THEN 3100
3062  LET W1$    "": LET W2$ = ""
3064  IF P <     1 THEN W1$ =  LEFT$ (W$,P - 1)
3066  IF LE <    P THEN W2$ =  RIGHT$ (W$,LE   P)
3070  LET CO = 1: LET SC = SC + 1
3080  LET W$ = W1$ + "?" + W2$
3090  VTAB 24: HTAB P + 29: PRINT C$;
3100  NEXT P
3110  IF LE = SC THEN  LET WO = 1
3120  RETURN
4000  REM  WIN
4010  HOME : HGR
4020  FOR L = 4 TO 10
4030  ON L - 4 GOSUB 7500,7600,7700,7800,7900,8000
4040  NEXT L
*4045  HCOLOR= 0: HPLOT 227,40 TO 230,45:
*4046  HPLOT 240,45 TO 243,40
4050  VTAB 21: HTAB 1: PRINT "WELL DONE, YOUR
      EXECUTION"
4060  PRINT "HAS BEEN POSTPONED"
*4065  FOR S = 1 TO 25
*4066  PRINT  CHR$ (7);
*4067  NEXT S
4070  RETURN
```

```
  10 PRINT"[CLR]"
  20 GOSUB 1000: REM SELECTWORD
  30 GOSUB 2000: REM TITLE
  40 IF WO=1 OR LO=1 THEN 80
  50 GOSUB 3000: REM CHOOSELETTER
  60 IF CO=0 THEN GOSUB 7000: REM HANGMAN
  70 GOTO 40
  80 IF WO=1 THEN GOSUB 4000: REM WIN
  90 IF LO=1 THEN GOSUB 5000: REM LOSE
 100 GOSUB 6000: REM PLAY AGAIN
 110 STOP
1000 REM SELECTWORD
1005 UD$="[HOME][DOWN×24]"
1010 W=8
1020 RESTORE
1030 CH=INT(RND(0)*W)+1
1040 FOR L=1 TO CH
1050 READW$
1060 NEXT L
1070 LE=LEN(W$): D$=W$
1080 RETURN
2000 REM TITLE
2010 PRINT"[CLR]"
2020 FOR L=4 TO 10
2030 ON L-4 GOSUB7500,7600,7700,7800,7900,8000
2040 NEXT L
2050 PRINT LEFT$(UD$,10)SPC(12)"HANGMAN"
2060 INPUT"[DOWN×11]          PRESS ENTER",A$
2070 PRINT"[CLR][DOWN]LETTERS CHOSEN"
2080 PRINT"[UP]-----------------"
2090 PRINT"[RIGHT×…]THE COMPUTERS'S"
2100 PRINT"[RIGHT×…]WORD IS"
2110 PRINT"[RIGHT×…]";
2120 FORL=1TOLE:PRINT"*";:NEXTL:PRINT
2130 PRINT"[…]PLEASE CHOOSE"
2140 PRINT"[…]A LETTER"
2150 WO=0: LO=0: LI=0: SC=0
2160 RETURN
3000 REM CHOOSELETTER
3010 PRINTLEFT$(UD$,23)"[…]    […]";
3020 INPUTC$:C$=LEFT$(C$,1)
3030 PRINT"[…]"SPC(LI)C$
3040 CO=0
3050 FOR P=1 TO LE
3060 IF C$<>MID$(W$,P,1) THEN 3100
3070 CO=1: SC=SC+1
3080 W$=LEFT$(W$,P-1)+"?"+RIGHT$(W$,LE-P)
3090 PRINTLEFT$(UD$,18)"[…]"SPC(P-1)C$
3100 NEXT P
3110 IF SC=LE THEN WO=1
3120 RETURN
4000 REM WIN
4010 PRINT"[CLR]"
4020 FOR L=4 TO 10
4030 ON L-4 GOSUB7500,7600,7700,7800,7900,8000
4040 NEXT
*4045 PRINT"[…]"SPC(20)"—"
4050 PRINTLEFT$(UD$,7)"   WELL DONE, YOUR EXECUTION"
```

```
5000 REM  LOSE
5010 HOME : HGR
5020 FOR L = 1 TO 10
5030 ON L GOSUB 7100,7200,7300,7400,7500,7600,7700,
     7800,7900,8000
5040 NEXT L
5050 VTAB 21: HTAB 1: PRINT "KNAVISH FOOL!"
5060 PRINT "YOU LOST!"
5070 PRINT "THE WORD WAS ";D$
*5072 FOR S = 1 TO 300
*5074 LET T = PEEK (- 16336)
*5076 NEXT S
*5078 HCOLOR= 0: HPLOT 228,48 TO 230,45:
*5079 HPLOT 240,45 TO 243,48
5080 FOR I = 141 TO 148: HPLOT 212,I TO 220,I:
     NEXT I
*5082 HCOLOR= 7: FOR I = 149 TO 156: HPLOT 220,I
      TO 228,I: NEXT I
*5084 HCOLOR= 0: FOR J = 141 TO 148: HPLOT 250,J
      TO 258,J: NEXT J
*5086 HCOLOR= 7: FOR J = 149 TO 156: HPLOT 242,J
      TO 250,J: NEXT J
5090 RETURN
6000 REM  PLAY AGAIN
6010 VTAB 24: HTAB 1: PRINT "DO YOU WISH TO PLAY
     AGAIN Y/N?";
6030 INPUT A$
6040 IF A$ = "Y" THEN 10
6050 HOME : VTAB 21: HTAB 16: PRINT "BYE BYE"
6060 RETURN
7000 REM  HANG MAN
7010 LET LI = LI + 1: IF LI = 10 THEN LO = 1
7020 ON LI GOTO 7100,7200,7300,7400,7500,7600,7700,
     7800,7900,8000
7100 REM  GALLOWS 1
7110 HCOLOR= 1
7120 FOR R = 180 TO 190
7130 HPLOT R,160 TO R,14
7140 NEXT R
7150 RETURN
7200 REM  GALLOWS 2
7210 HCOLOR= 1
7220 FOR R = 10 TO 14
7230 HPLOT 180,R TO 250,R
7240 NEXT R
7250 RETURN
7260 RETURN
7300 REM  GALLOWS 3
7310 HCOLOR= 1
7320 HPLOT 190,30 TO 206,14
7330 RETURN
7400 REM  ROPE
7410 HCOLOR= 2
7420 HPLOT 234,29 TO 234,15
7430 RETURN
7500 REM  FACE
7510 HCOLOR= 3: FOR I = 225 TO 245
7520 HPLOT I,49 TO I,30
7530 NEXT I
7540 HCOLOR= 0
7550 HPLOT 230,45 TO 240,45
7560 HPLOT 228,34 TO 230,34: HPLOT 240,34 TO 242,34
7570 HPLOT 234,38 TO 236,38
7580 RETURN
7600 REM  BODY
7610 HCOLOR= 5
7620 FOR I = 220 TO 250
7630 HPLOT I,70 TO I,50
7640 NEXT I
7650 HCOLOR= 0: HPLOT 235,65: HPLOT 235,85
7660 RETURN
7700 REM  ARM 1
7710 HCOLOR= 5
7720 FOR I = 213 TO 218
7730 HPLOT I,55 TO I,105
7740 NEXT I
7750 HCOLOR= 3
7760 HPLOT 214,106 TO 214,110 TO 218,110 TO 218,106
7770 RETURN
7800 REM  ARM2
7810 HCOLOR= 5
7820 FOR I = 252 TO 257
7830 HPLOT I,55 TO I,105
7840 NEXT I
7850 HCOLOR= 3
7860 HPLOT 252,106 TO 252,110 TO 256,110 TO 256,106
7870 RETURN
7900 REM  LEG1
7910 HCOLOR= 6
7920 FOR I = 220 TO 232
7930 HPLOT I,100 TO I,140
7940 NEXT I
7950 HCOLOR= 7
7960 FOR I = 141 TO 148
7970 HPLOT 212,I TO 228,I
7980 NEXT I
7990 RETURN
8000 REM  LEG2
8010 HCOLOR= 6
8020 FOR I = 238 TO 250
8030 HPLOT I,100 TO I,140
8040 NEXT I
8050 HCOLOR= 1
8060 FOR I = 92 TO 98
8070 HPLOT 220,I TO 250,I
8080 NEXT I
8090 HCOLOR= 7
8100 FOR I = 141 TO 148
8110 HPLOT 242,I TO 258,I
8120 NEXT I
8130 REM  DELAY
8140 FOR D = 1 TO 1000: NEXT D
8150 RETURN
9000 DATA  APPLE,ANGLE,ALPHABET,ASTEROID
9010 DATA  BOTTLE,BOOK,BUFFALO,BEACH
```

```
4060 PRINT"  HAS BEEN POSTPONED"
*4063 FORI=0TO24:POKE54272,0:NEXT
*4067 POKE54277,15:POKE54278,240:POKE54296,15
*4068 POKE54276,33:FORI=0TO255:POKE54273,I:NEXT
*4069 POKE54276,32
4070 RETURN
5000 REM LOSE
5010 PRINT"J"
5020 FOR L=1 TO 10
5030 ON L GOSUB7100,7200,7300,7400,7500,7600,7700,
     7800,7900,8000
5040 NEXT
5050 PRINT"SQQQQQQ  KNAVISH FOOL !"
5060 PRINT"Q    YOU LOST"
5070 PRINT"QQQQTHE WORD WASQ":PRINTSPC(5)D$
*5072 FORI=0TO24:POKE54272,0:NEXT
*5074 POKE54277,15:POKE54278,240:POKE54296,15
*5076 POKE54276,33:FORI=255TO0STEP-1:POKE54273,I:NEXT
*5077 POKE54276,32
*5078 PRINT"SQQQQQQQQQQ"SPC(28)",—,"
*5079 PRINT"QQQQQQQQQQQQ"SPC(28)"■■ ■ "
5080 RETURN
6000 REM PLAY AGAIN
6010 PRINT"SQQQQQQQQQQQQQQQQQQQQQ  DO YOU WISH TO"
6020 PRINT"    PLAY AGAIN";
6030 INPUT A$
6040 IF LEFT$(A$,1)="Y" THEN 10
6050 PRINT"JQQQQQQQQQQX     BYE BYE"
6060 RETURN
7000 REM HANGMAN
7010 LI=LI+1 :IF LI=10 THEN LO=1
7020 ON LI GOTO 7100,7200,7300,7400,7500,7600,7700
     7800,7900,8000
7100 REM GALLOWS 1
7110 PRINT"SQ■"
7120 FOR R=0 TO 21
7130 PRINTSPC(19)"Q  "
7140 NEXT
7150 RETURN
7200 REM GALLOWS 2
7210 PRINT"SQ■"
7220 FOR R=0 TO 1
7230 PRINTSPC(21)"Q          "
7240 NEXT
7250 RETURN
7300 REM GALLOWS3
7310 PRINT"SQQQ■"
7320 FOR R=2 TO 0 STEP-1
7330 PRINTSPC(21+R)"■■■"
7340 NEXT
7350 PRINTSPC(21)"■"
7360 RETURN
7400 REM ROPE
7410 PRINT"SQQQ■"
7420 FORR=0TO1:PRINTSPC(29)"I":NEXT
7430 RETURN
7500 REM FACE
7510 PRINT"SQQQQQQ■"
7520 PRINTSPC(27)",—+—,"
7530 FORR=0TO2:PRINTSPC(27)"I   I":NEXT
7540 PRINTSPC(27)"—+—"ITII"
7550 PRINTSPC(28)"+ +":PRINTSPC(29)":":
     PRINTSPC(28)"——"
7560 RETURN
7600 REM BODY
7610 PRINT"SQQQQQQQQQQX"
7620 FORR=0TO4:PRINTSPC(28)"Q   ":NEXT
7630 PRINT"IIII"SPC(29)"QQ":PRINTSPC(29)"QQQ"
7640 RETURN
7700 REM ARM1
7710 PRINT"SQQQQQQQQQQX"
7720 FORR=0TO1:PRINTSPC(27)"Q ":NEXT
7730 FORR=0TO2:PRINTSPC(27)"QI":NEXT
7740 PRINTSPC(27)"O"
7750 RETURN
7800 REM ARM2
7810 PRINT"SQQQQQQQQQQX"
7820 FORR=0TO1:PRINTSPC(31)"Q ":NEXT
7830 FORR=0TO2:PRINTSPC(31)"Q ":NEXT
7840 PRINTSPC(31)"O"
7850 RETURN
7900 REM LEG1
7910 PRINT"SQQQQQQQQQQQQQQQQ■"
7920 FORR=0TO5:PRINTSPC(28)"Q ■":NEXT
7930 RETURN
8000 REM LEG2
8010 PRINT"SQQQQQQQQQQQQQQQQ■"
8020 FORR=0TO5:PRINTSPC(28)"Q  ":NEXT
8030 PRINTSPC(28)"IIIIIIIQ * "
8040 PRINTSPC(28)"■QQQQQQ  —"
8120 REM DELAY
8130 FOR D=1 TO 1000:NEXT D
8140 RETURN
9000 DATA APPLE,ANGLE,ALPHABET,ASTEROID
9010 DATA BOTTLE,BOOK,BUFFALO,BEACH
```

Glossary

DATA A list of information that is required by a program. **DATA** can consist of words or numbers, or both together. A program is sent to the **DATA** with the instruction **READ**.

FOR.....NEXT This is a sequence of commands that are used to make the computer repeat an operation a certain number of times. For example, the loop **FOR X=1 TO 5:PRINT 2*X:NEXT X** would cause the computer to print the two times table up to five.

GOTO This statement tells the computer to go to the specified line, missing out any lines in-between. It is often used with **IF.....THEN** (see below) and is only operated if certain conditions are true. Be careful when using **GOTOs**, as it's easy to have the program jumping backward and forward so much that it is impossible to read.

HGR This sets the high resolution graphics mode on the Apple.

HPLOT This places a set of x, y coordinates on the Apple screen. If **HPLOT** is followed by **TO**, it draws a line from the last point plotted to the coordinates indicated. This works both horizontally and vertically.

IF.....THEN This is used as a way of telling the computer to do something only when certain conditions are true. This instruction often looks something like this: **IF score=LE THEN WO=1**.

INPUT This instruction allows the computer to be given information while a program is running. When the computer comes to an **INPUT** instruction it prints a question mark (or, for some computers, a different symbol) to prompt the user, and waits for the input to be given and carriage return to be pressed.

INT **INT** is short for integer, and instructs the computer to make a whole number of a figure with decimal places in it. It is often used in conjunction with the **RND** command which instructs the computer to generate a random number (see below).

LEFT$ This instruction is used to copy part of a string, starting at the left hand end. It is followed in brackets by the string name and the number of characters to be copied.

LEN This is a **BASIC** instruction which counts the number of characters in a string.

LET This is one way of giving the computer information. In some programs there may be statements such as: **X=10**
This simply means that the number ten is stored under the label X. It is often clearer to write:
LET X=10
The **LET** statement also gives rise to something that at first sight seems illogical if not impossible. In many programs you will see things like:
LET X=X+1
Of course, in mathematical terms **X** can't equal **X+1**. All this type of statement means is "increase the value of whatever is stored in **X** by one."

LIST This makes the computer display whatever program it has in its memory. You can **LIST** single lines, or parts of a program by following the **LIST** with appropriate line numbers.

MID$ This is used to copy the middle part of a string. It is followed in brackets by the string name, the start position, and the number of characters to be copied.

PEEK This instruction looks at a particular memory location. It is often associated with **POKE**.

PIXEL This represents a point on the grid in graphics mode. The number of pixels per screen is determined by the quality of the graphics, e.g. high or low resolution mode.

POKE This stores numeric information in the computer's memory. It is often used for sound and places a binary number in a particular location.

PRINT This tells the computer to display something on the screen. Letters and symbols that are to be displayed should be enclosed in quotation marks, but numbers need not be.

RIGHT$** Similar to **LEFT$, but copies the right-hand end of a string.

RND This instruction makes the computer generate a random number. The precise instruction varies between different models of computer. **RND** is followed by a figure in brackets, usually 0 or 1. To obtain random whole numbers it must be multiplied and made a whole number by the **INT** statement.

STEP The **STEP** statement is always used following a **FOR....** statement. It indicates the amount that the variable should be changed for each operation. For example: **FOR X=0 TO 20 STEP 5: PRINT X: NEXT X** Would mean that **X** would rise in steps of five, so that the computer would print 0,5, 10, 15, 20

Variables When you give the computer information you have to give it a label under which it is stored. This label is called a variable since the information it contains may change during the course of the program. When you want the computer to do something with the information, you must refer to it by its label – its variable name. For example, the statement **LET A=6** places 6 under the variable name **A**.

There are two types of variable. A *numeric variable* is one in which the information stored will always be numbers. If the data to be stored consists of letters or words then a *string variable* must be used. The variable name must then be followed by the string sign – **$**. So, for example, if you wanted a name stored, the statement would read: **LET N$="JAMES"**. String variable information must always be in quotes.

Index

Design
Cooper · West

Program editors
Steve Rodgers
Marcus Milton

Illustrators
Lionel Jeans
Rob Shone

WRITE YOUR OWN PROGRAM

THIS NEW SERIES INTRODUCES THE ART OF PROGRAMMING YOUR COMPUTER. EACH BOOK SHOWS HOW TO STRUCTURE A PROGRAM INTO ROUTINES, AND AT THE SAME TIME EXPLAINS AND ANALYZES WHAT EXACTLY YOU ARE ASKING THE COMPUTER TO DO AND WHY.

AND THERE'S FUN TOO! EACH BOOK CONTAINS ONE OR MORE EXCITING AND ORIGINAL COMPUTER GAMES. THESE CAN BE ADAPTED AND EXTENDED TO DO BIGGER AND BETTER THINGS.

THE BOOKS ARE SPECIFIC TO THE COMMODORE 64 AND APPLE IIe.

TITLES IN THE SERIES
BEGINNING BASIC—SPACE JOURNEY
GRAPHICS—HANGMAN

A Gloucester Press Library Edition

ISBN 0-531-03483-6

I14