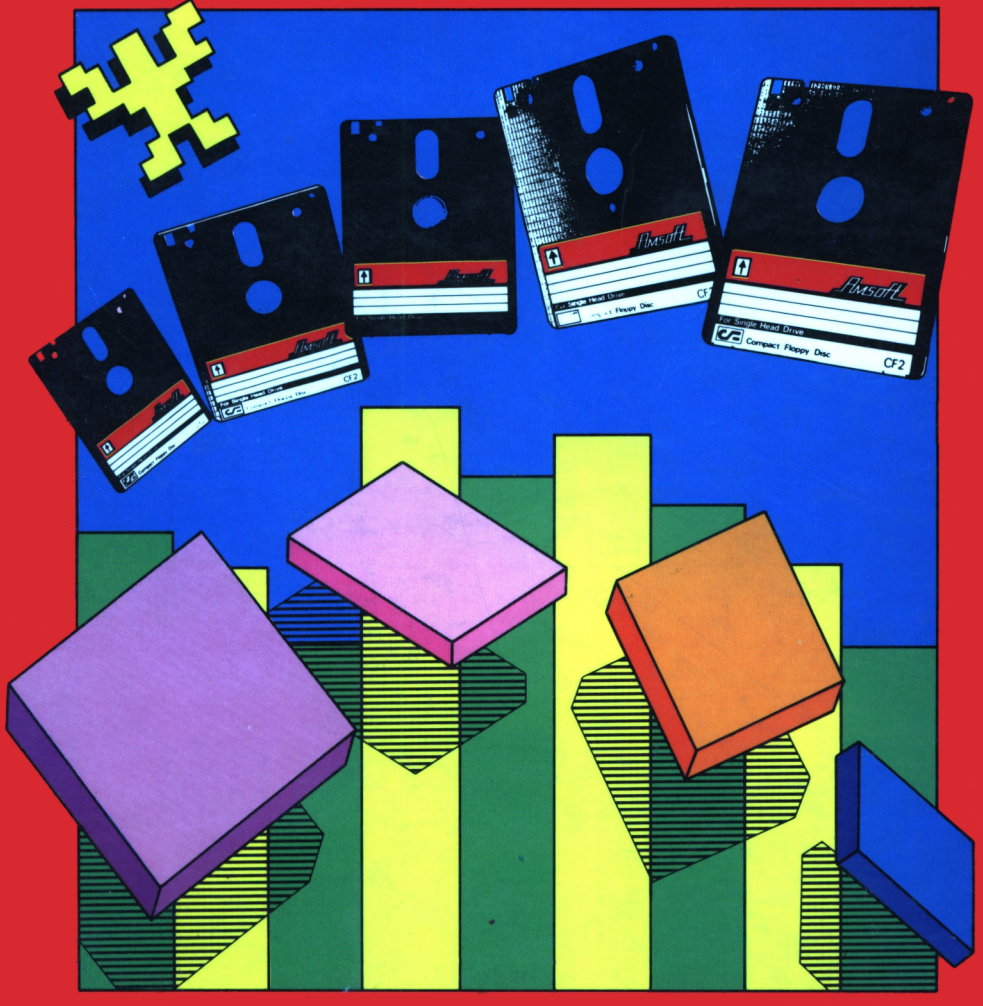
INTRODUCING THE AMSTRAD CPC 664

Jennifer & Cameron Procter



MICRO PRESS

Introducing the Amstrad CPC 664

Introducing the Amstrad CPC 664

Jennifer & Cameron Procter

MICRO PRESS

First published in 1985 in the United Kingdom by
Micro Press
Castle House, 27 London Road
Tunbridge Wells, Kent

British Library Cataloguing in Publication Data

Procter, Cameron
Introducing the Amstrad CPC 664
1. Amstrad CPC 664 (Computer) 2. Amstrad CPC 464 (Computer)
I. Title
001.64'04 QA76.8.A4/

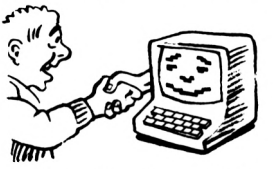
ISBN 0–7447–00370X

Typeset by MC Typeset, Chatham, Kent
Printed by Mackays of Chatham Ltd.

Contents

Chapter One

Getting to know your Amstrad CPC 664



Computing is a skill and like all other skills can only be learned by practice. Don't just read this book but try it out on your Amstrad CPC 664. Although the text tells you how the computer responds, you will learn much faster by using your Amstrad as you read the book. Inevitably you will make mistakes but don't be disheartened and always try to understand what you did wrong.

As your Amstrad CPC 664 has no brain it cannot decide what to do next, it can only obey your instructions. Since it is unable to understand English you must learn its language which is called BASIC. This language is just a set of instructions that the computer knows how to carry out. In this book you will learn to make your Amstrad CPC 664 do such things as calculations, draw coloured patterns on the screen and play simple tunes.

Remember that the computer always obeys your instructions and so if things are not working as planned then you must have made a mistake.

Getting started

When you unpack your Amstrad CPC 664, you will find that it consists of a computer, a disk and a monitor which resembles a television. Part 1 of your User Instructions Manual gives instructions on fitting a plug to the mains lead of the monitor, on connecting the computer to the monitor and on switching on the system.

When you switch on, the computer sends you a message by displaying on the screen:

```
Amstrad 64k Microcomputer  (v2)

©1984 Amstrad Consumer Electronics plc
         and Locomotive Software Ltd.

BASIC 1.1
Ready
```

In the first three lines the computer is introducing itself and telling you that it converses in the language called BASIC. When the computer displays:

```
Ready
```

it is telling you that it is waiting for an instruction. If you do not send it one it will remain in this state until you switch off. Since the computer cannot understand the spoken word you must type your instructions at the keyboard.

Introducing the keyboard

Beneath the `Ready` on the screen is a square symbol. This is called the cursor and marks the position where the computer will display the next character on the screen. When you enter a character at the keyboard the computer automatically displays it on the screen as a check on your typing.

Press the **K** key. The cursor will move to the right and a `k` will take its place.

Press and hold down the **K** key. You will find that the computer displays `k`'s across the screen. When one line of the screen is complete the display automatically continues onto the next. Release the key after the computer has displayed two lines of `k`'s on the screen. All the letter keys have this repeat action when held down.

Count the number of `k`'s in each line. You will find that the screen displays 40 characters to the line.

The blue key labelled **DEL** is called the delete key. Press this key once and the `k` to the left of the cursor will be deleted. The cursor will move one character to the left. Press and hold down the **DEL** key until all the `k`'s are deleted. When there are no `k`'s

left to delete the computer bleeps as a warning that you are trying to delete when there is nothing to be deleted.

To type an upper case (i.e. capital) K press and hold down either of the blue shift keys that are labelled **SHIFT**. Press the **K** key once and then release the **SHIFT** key. We shall write this as **SHIFT K**.

Use the **DEL** key to delete the K.

Tutorial

1.1) Type

```
This is the Amstrad computer
```

and then delete it using the **DEL** key.

The number keys

Your Amstrad CPC 664 keyboard has two sets of numbers, one in the usual typewriter position and the other forming a number pad to the right of the main keyboard. Either set may be used. The keys on the number pad are labelled ***f1*** ***f2*** etc. The significance of the ***f*** 's will be discussed later.

Press one of the keys labelled **4** and a 4 will be displayed on the screen. The number keys in the typewriter position have a repeat action similar to the letter keys. However, the number pad keys do not have this action.

Be careful to distinguish the number zero from the letter O. On the keyboard the number zero is the key labelled **Ø** whereas the letter O is the key labelled **O** between the **I** and **P** keys.

Use the **DEL** key to delete from the screen any numbers you have typed.

Tutorial

1.2) Type:

```
The Amstrad CTM644 colour monitor
```

and then delete it using the **DEL** key.

Symbols

The symbol keys are shown in Table 1.1. The numbers must be typed on the main keyboard.

Symbol	Key
Comma	the key to the right of the **M** key
Full stop	the key to the right of the comma
Colon	the key to the right of the **L** key
Semi-colon	the key to the right of the colon
Exclamation mark	**SHIFT 1**
Quotation mark	**SHIFT 2**
Dollar sign	**SHIFT 4**
Apostrophe	**SHIFT 7**
Underscore	**SHIFT Ø**
Pound sign	**SHIFT** ↑ where ↑ is the key to the left of the key labelled **CLR**
Question mark	**SHIFT /** where the **/** is the key to the right of the full stop

Table 1.1

Type these symbols and they will be displayed on the screen as expected. Use the **DEL** key to delete them.

Tutorial

1.3) Type:

```
"Stop! Who goes there? It is after nine o'clock."
```

and then delete it using the **DEL** key.

Symbol	Function	Key
+	addition	**SHIFT ;**
–	subtract	the key to the right of **Ø**
*	multiplication	**SHIFT :**
/	division	the key to the right of the full stop
=	equals	**SHIFT –**
<	less than	**SHIFT ,**
>	greater than	**SHIFT .**
(	open bracket	**SHIFT 8**
)	close bracket	**SHIFT 9**

Table 1.2

The arithmetic symbols

The arithmetic symbols are shown in Table 1.2. Be careful not to confuse the brackets (and) with the less than < and greater than > symbols. As before the numbers must be typed on the main keyboard.

Type these symbols and they will be displayed on the screen as expected. Use the **DEL** key to delete them.

Tutorial

1.4) Type:

```
(6+5)-2 < (24/6)*4
```

and then delete it using the **DEL** key.

1.5) Type:

```
CHR$(224)
```

and then delete it using the **DEL** key.

The **CAPS LOCK** key

Press once the blue key labelled **CAPS LOCK** which is to the left of the **A** key. The computer will not display anything on the screen. Using the main keyboard, press the three keys labelled **A** **;** and **2**. The computer displays on the screen:

```
A ; 2
```

When you pressed the **CAPS LOCK** key you instructed the keyboard to send upper case letters to the computer and hence a capital A was displayed above. However the **CAPS LOCK** key does not affect the numbers or symbols and hence a semi-colon and a 2 were displayed rather than their upper case symbols.

It is unlikely that you will require this use of the keyboard, but you may press the key accidentally. To return to the normal keyboard press the **CAPS LOCK** key again. Now when you press the **H** key the screen will display h.

Use the **DEL** key to remove any characters that you have typed from the screen.

The other keys will be introduced as required.

Giving the computer instructions

The computer will now be instructed to clear the screen. You cannot type 'clear the screen' as the computer does not understand English. The statement **CLS** tells the computer to CLear the Screen. Type:

```
CLS
```

The computer has now memorised the instruction but has not carried it out. It will not be obeyed until you press one of the two blue keys labelled **ENTER**. Press one of them. The computer now carries out the instruction by clearing the screen. It then displays:

```
Ready
▮
```

This shows that it has completed the last instruction and is ready to accept another one. When the computer carries out an instruction it is said to *execute* it. Once it has executed the instruction it forgets it. Press an **ENTER** key. As the computer does not have an instruction in its memory it does nothing except display:

```
Ready

▮
```

If you press a wrong key while entering an instruction it can be corrected provided you have not pressed the **ENTER** key. Perhaps you typed **CLAS** instead of **CLS**. Type:

```
CLAS
```

Press the **DEL** key twice and the screen will display:

```
CL▮
```

Now type a capital **S** and the screen will display:

```
CLS▮
```

As the instruction has now been corrected, press an **ENTER** key. The computer executes your instruction by clearing the screen.

Syntax errors

A syntax error occurs when the computer does not understand your instruction. Type:

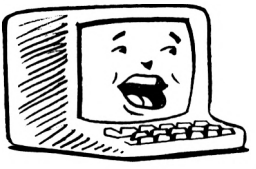
```
CLSS
```

Assume that you did not notice that you had typed `CLSS` for `CLS`. Press an **ENTER** key. As `CLSS` is not one of the statements that the computer recognises it is not able to execute it. The computer tells you that it cannot recognise the statement by displaying:

followed by the usual `Ready` message.

You will notice that the computer is very particular. If you have made any mistakes in your instructions then it will not recognise them and will display a syntax error. If you create a syntax error and you cannot find the fault then refer to Appendix 1 where possible causes of the errors are suggested.

The computer speaks to you



Displaying sentences on the screen

In this chapter you will use the PRINT statement to instruct the computer to display a sentence on the screen. Type:

```
PRINT "This is the Amstrad computer."
```

The computer displays on the screen exactly what you have typed so that you can check that you have not made any mistakes. It has also memorised what you typed but has made no attempt to obey the instruction.

Press an **ENTER** key. This tells the computer that you have finished typing the instruction and that it should now execute it. The computer displays on the screen:

```
This is the Amstrad computer.
```

You will see that the computer has displayed everything inside the quotation marks but has not shown the quotation marks themselves.

To summarise: a PRINT statement followed by a sentence within quotation marks instructs the computer to display the sentence on the screen.

The above instruction:

```
PRINT "This is the Amstrad computer."
```

has a space between the T of PRINT and the quotation marks. This space is optional but it is recommended and will be included throughout this book as it makes the text easier to read.

Example 2a

Use a PRINT statement to display on the screen:

```
My name is John.
```

Type:

```
PRINT "My name is John."
```

Press an **ENTER** key. The computer now displays on the screen:

```
My name is John.
```

Example 2b

Use a PRINT statement to display on the screen:

```
There isn't a train to London tonight.
```

Type:

```
PRINT "There isn't a train to London tonight."
```

Press an **ENTER** key. The computer now displays on the screen:

```
There isn't a train to London tonight.
```

Tutorial

Remember to press an **ENTER** key after typing the PRINT statement.

2.1) Use a PRINT statement to display on the screen:

```
I am 25 years old.
```

2.2) Use a PRINT statement to display on the screen:

```
I live in Great Britain.
```

2.3) Use a PRINT statement to display on the screen:

```
Would you like a cup of tea?
```

In the PRINT statement, quotation marks are used to enclose the sentence to be displayed on the screen. Because they have this special meaning to the computer, the sentence

being displayed must not contain quotation marks. The computer is unable to display:

"I don't know," he said.

Solutions to the tutorials

2.1) `PRINT "I am 25 years old."`

2.2) `PRINT "I live in Great Britain."`

2.3) `PRINT "Would you like a cup of tea?"`

Concept of a program

In Chapters 1 and 2 the `CLS` and `PRINT` statements were introduced. You shall now use these two statements to instruct the computer to clear the screen and then display:

```
My name is John.
```

Type:

```
CLS
```

Press an **ENTER** key. The computer now clears the screen and displays at the top left hand corner:

```
Ready
```

Type:

```
PRINT "My name is John."
```

Press an **ENTER** key. The computer now displays on the screen:

```
My name is John.
Ready
```

Notice that you had to type each instruction as it was required. It would be more convenient if you could ask the computer to remember both the instructions:

```
CLS
PRINT "My name is John."
```

and then execute them on request. The above two instructions treated together in this way are called a *program*. A program is therefore a sequence of instructions that the computer can remember and execute on request.

The computer must know the order in which the instructions are to be executed. This is done by giving each instruction a number called a line number which is placed before the instruction as follows:

```
1 CLS
2 PRINT "My name is John."
```

The computer executes the instructions in ascending order of line number starting with the lowest. The space between the line number and the instruction is optional but it is recommended as it makes the program easier to read.

In the above program the lines were given the numbers 1 and 2. However when writing programs it is conventional to number the lines 10, 20, 30 . . . etc. Using this convention the program would become:

```
10 CLS
20 PRINT "My name is John."
```

This convention will be used in all future programs.

The above program must be placed in the computer's memory before it can be executed. This is called entering the program.

Entering a program

Before entering a program, you must instruct the computer to forget the previous one. This is achieved by using the **NEW** statement. Type:

```
NEW
```

Press an **ENTER** key. The computer has now forgotten any previous programs and displays:

```
Ready
```

To enter the program:

```
10 CLS
20 PRINT "My name is John."
```

type:

```
10 CLS
```

Press an **ENTER** key. The computer has remembered the line and displays the cursor showing that it is waiting for the next instruction or program line. Type the next line:

```
20 PRINT "My name is John."
```

Press an **ENTER** key. The program is now stored in the computer's memory and the cursor is displayed on the screen.

Read your program carefully. If you have made any mistakes then type NEW, press an **ENTER** key and re-enter the program. The **DEL** key can still be used to remove a mistake while typing a line as explained in Chapter 1.

Running a program

Running a program means that you are instructing the computer to execute the program that is in its memory. Type:

```
RUN
```

Press an **ENTER** key. The computer now runs the program entered above. It first executes line 10 where it clears the screen and then line 20 where it displays on the screen:

```
My name is John.
```

As the computer has now completed the program it indicates that it is waiting for another instruction by displaying:

```
Ready
```

One of the advantages of a program is that it is retained in the computer's memory after it has been RUN. This means that it can be RUN again without being retyped. Type:

```
RUN
```

Press an **ENTER** key. The computer now runs the program and as before it clears the screen and displays:

```
My name is John.
```

If you had a typing error at line 10 which you did not notice the computer may display an error message on the screen. If

this occurs then press an **ENTER** key, type NEW, press an **ENTER** key again and re-enter the program. Errors will be discussed in the next section.

Example 3a

Enter and RUN the following program.

```
10 PRINT "There are six biscuits on the plate."
20 PRINT "Would you like one?"
```

Type:

```
NEW
```

Press an **ENTER** key. Type:

```
10 PRINT "There are six biscuits on the plate."
```

Press an **ENTER** key. Type:

```
20 PRINT "Would you like one?"
```

Press an **ENTER** key. Check that you have entered the program correctly. If you have made a mistake then type NEW, press an **ENTER** key and re-enter the program.

You have now entered the program and are ready to RUN it. Type:

```
RUN
```

Press an **ENTER** key. The computer now executes line 10 and displays on the screen:

```
There are six biscuits on the plate.
```

It then executes line 20 and displays on the screen:

```
Would you like one?
```

As the computer has now completed the program it indicates that it is waiting for another instruction by displaying:

```
Ready
```

Tutorial

Remember to use the NEW instruction before entering a program and to press an **ENTER** key after typing each line.

3.1) Enter and RUN the following program:

```
10 CLS
20 PRINT "How far is it to Paris?"
```

3.2) Enter and RUN the following program:

```
10 PRINT "I have found a tie."
20 PRINT "Does it belong to John?"
```

3.3) Enter and RUN the following program:

```
10 CLS
20 PRINT "Stop! You have dropped your glove."
```

Editing a program

So far if you notice a mistake after typing a line, you have to type NEW, press the **ENTER** key and re-enter the complete program. For long programs this would be unsatisfactory. Once the program is in the computer's memory you can:

- list the program
- change a line
- insert a line
- remove a line
- renumber the lines.

This is called editing. To demonstrate editing the computer's memory must contain a program. Enter the program:

```
10 PRINT "My name is John James Brown."
20 PRINT "I live in Britain."
```

Listing a program

It is often useful to be able to display the program in the computer's memory on the screen. This is called listing the program. Type:

```
LIST
```

Press an **ENTER** key. The computer now displays the program on the screen:

```
10 PRINT "My name is John James Brown."
20 PRINT "I live in Britain."
```

You may find it is easier to read the program if you clear the screen before listing it. Type:

```
CLS
```

Press an **ENTER** key. Type:

```
LIST
```

Press an **ENTER** key. The computer now displays the program on a clear screen.

Changing a line

The `EDIT` statement can be used to:

- insert an extra word into a line
- remove a word from a line
- alter a word in a line.

Inserting an extra word into a line

Let's assume that line 20 should have been:

```
20 PRINT "I live in Great Britain."
```

The line can be modified by typing:

```
EDIT 20
```

The word `EDIT` must be followed by a space. Press an **ENTER** key. The computer now displays line 20 and places the cursor over the `P` of `PRINT`.

The cursor right key marked **>** and positioned to the right of the grey **COPY** key will now be used to move the cursor to the right. Use this key to move the cursor over the `B` of `Britain`. If you accidentally move the cursor too far then the cursor left key marked **<** can be used to move the cursor back. Type the word that was missed out (i.e. `Great`) followed by a space. Press an **ENTER** key. This instructs the computer that your alterations are complete.

Now LIST the program. The computer will display:

```
10 PRINT "My name is John James Brown."
20 PRINT "I live in Great Britain."
```

Notice that line 20 has been altered.

Deleting a word from a line

Let's remove James from line 10. Type:

```
EDIT 10
```

Press an **ENTER** key. The computer now displays line 10 and places the cursor over the P of PRINT. Use the cursor right key > to move the cursor over the J of James. The **CLR** key to the left of the blue **DEL** key can now be used to remove the character under the cursor. Use this key to remove the word James. The cursor is now over the space before the word Brown. Press the **CLR** key to remove this space. Press an **ENTER** key. This instructs the computer that your alterations are complete.

Now LIST the program. The computer will display:

```
10 PRINT "My name is John Brown."
20 PRINT "I live in Great Britain."
```

Notice that line 10 has been altered.

Altering a word in a line

Assume that you wish to change John in line 10 to Richard. Type:

```
EDIT 10
```

Press an **ENTER** key. The computer now displays line 10 and places the cursor over the P of PRINT. Use the cursor right key > to move the cursor until it is over the J of John. Use the **CLR** key to remove John. Type: Richard. Press an **ENTER** key. This instructs the computer that your alterations are complete. Now LIST the program. The computer will display:

```
10 PRINT "My name is Richard Brown."
20 PRINT "I live in Great Britain."
```

Notice that line 10 has been altered.

The above editing techniques have allowed you to modify a line. However if the line has several mistakes then it may be quicker to retype it without using the EDIT statement. Let's assume that line 20 should have been:

```
20 PRINT "I am 20 years old"
```

The line can be changed by typing:

```
20 PRINT "I am 20 years old"
```

Press an **ENTER** key. Retyping a line automatically deletes the original line. Now LIST the program and notice that line 20 has been altered. Retyping a line may seem an easy way to edit. However it is worth practising the editing techniques as they will save you time once you are familiar with them.

Inserting a line

Let's assume that you wish to insert the following line:

```
PRINT "My telephone number is 09876 1234."
```

between lines 10 and 20 in the above program. The line could be given the line number 15. Type:

```
15 PRINT "My telephone number is 09876 1234."
```

Press an **ENTER** key. LIST the program and observe that line 15 has been inserted.

This also shows that when entering a program the order in which the lines are typed is not important as the computer sorts them according to their line numbers.

Removal of a line

You will now remove line 20. Type:

```
20
```

Press an **ENTER** key. LIST the program and observe that line 20 has been removed.

Renumbering the lines

You will notice that your program now has line numbers 10

and 15. If you wish to renumber these to the conventional 10 and 20 then type:

```
RENUM
```

Press an **ENTER** key. The computer has now renumbered your program. `LIST` the program and observe that it now has conventional line numbers.

Tutorial

3.4) Enter the program:

```
10 PRINT "The fire is very hot."
```

Edit line 10 so that it becomes:

```
10 PRINT "The fire is hot."
```

3.5) Enter the program:

```
10 PRINT "The river is deep."
```

Edit line 10 so that it becomes:

```
10 PRINT "The river is very deep."
```

3.6) Enter the program:

```
10 CLS
20 PRINT "My favourite colour is red."
```

Edit line 20 so that it becomes:

```
20 PRINT "My favourite colour is green."
```

3.7) Enter the program:

```
10 PRINT "What is the weather forecast for today?"
20 PRINT "It is going to be sunny."
```

Edit line 20 so that it becomes:

```
20 PRINT "It is going to be wet."
```

3.8) Enter the program:

```
10 PRINT "I want to buy three white mice."
```

Edit line 10 so that it becomes:

```
10 PRINT "I want to buy two white rats."
```

3.9) Enter the program:

```
10 CLS
20 PRINT "How much is the car?"
30 PRINT "It costs £10000!"
```

Edit line 30 so that it becomes:

```
30 PRINT "It costs £5000!"
```

3.10) Enter the program:

```
10 CLS
20 PRINT "I would like an ice-cream."
30 PRINT "No, I shall have a drink instead."
```

Delete line 10 and renumber the program.

3.11) Enter the program:

```
10 PRINT "Open the door."
20 PRINT "Walk through it."
30 PRINT "Walk away."
```

Insert a line between 20 and 30 which will display on the screen:

```
Shut the door.
```

LIST the program and then renumber it.

Error messages

You will remember from Chapter 1 that if you ask the computer to execute a statement that it does not understand then it may reply:

```
Syntax error
```

Enter the following program:

```
10 CLS
20 PRIINT "My name is John Brown."
```

Notice the deliberate spelling mistake PRIINT at line 20. RUN the program. The computer clears the screen at line 10 but when it tries to execute line 20 it does not understand the command PRIINT and displays on the screen:

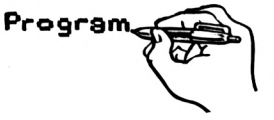
```
Syntax error in 20
20 PRIINT "My name is John Brown"
```

When the computer found the error at line 20 it automatically displayed the syntax error message and then executed an `EDIT 20` instruction. This caused it to display line 20 and put the cursor over the `P` of `PRINT`. You can correct the error by using the cursor right key to move the cursor over the first `I` of `PRINT`. Press the **CLR** key to remove the `I`. Press an **ENTER** key to tell the computer that you have finished editing the line. `LIST` the program and notice that line 20 is now correct.

Sometimes when the computer detects an error it displays a different error message. However it still indicates the line number where the error was detected. In these cases edit the line and correct the error.

If at any point in the book you do not understand the reason for an error message then refer to Appendix 1 where possible causes are suggested.

Writing your own programs



Introduction

In Chapter 3 you entered and ran programs. You will now learn to write your own programs.

When writing a program you must first decide what you want the program to do. Let's consider the following example:

Example 4a

Write a program to clear the screen and then display on the screen:

```
Where has the ball gone?
I think it is over there.
```

The above explanation of the program's function is called the program description. In this example it can be split into two parts. First you are asked to clear the screen. As the CLS statements clears the screen, a suitable line would be:

```
10 CLS
```

Next you are asked to display on the screen:

```
Where has the ball gone?
I think it is over there.
```

As the PRINT statement displays information on the screen, the next two lines could be:

```
20 PRINT "Where has the ball gone?"
30 PRINT "I think it is over there."
```

As you have now satisfied both parts of the program

description, the complete program would be:

```
10 CLS
20 PRINT "Where has the ball gone?"
30 PRINT "I think it is over there."
```

Enter and RUN this program. The computer should now display on the screen:

```
Where has the ball gone?
I think it is over there.
```

Tutorial

In the following tutorials if the computer does not create the desired screen display when you run the program, then LIST it and examine the listing for your mistake.

4.1) Write, enter and RUN a program that clears the screen and then displays:

```
We are going out now.
Have you switched off the television?
```

4.2) Write, enter and RUN a program that displays on the screen:

```
It will soon be holiday time.
Where are you going this year?
```

Example 4b

Write, enter and RUN a program that clears the screen and then displays:

```
The marathon was cancelled because of bad weather.
```

A possible program could be:

```
10 CLS
20 PRINT "The marathon was cancelled because of bad weather."
```

Enter and RUN this program. The computer will clear the screen and then display:

```
The marathon was cancelled because of ba
d weather.
```

You will see that when the computer reached the end of the first line of its screen it automatically continued to the next. This has resulted in splitting the word **bad** between the two lines which does not give a good screen presentation. A better presentation can be obtained by changing the **PRINT** statement in line 20 into two separate **PRINT** statements. The program becomes:

```
10 CLS
20 PRINT "The marathon was cancelled because of"
30 PRINT "bad weather."
```

Enter and **RUN** the program. The computer will clear the screen and then display:

```
The marathon was cancelled because of
bad weather.
```

Tutorial

4.3) Write, enter and **RUN** a program that clears the screen and then displays:

```
I want a library book but I can't make up my mind which
book to borrow.
```

4.4) Write, enter and **RUN** a program that clears the screen and then displays:

```
Please shut the gate or the dog will escape.
```

The REM (remark) statement

So far you have only written short programs which can be easily understood from the listings. However longer programs would be difficult to follow unless accompanied by some explanation. The **REM** statement allows this and other relevant comments to be contained within the program.

Example 4c

Rewrite the solution to tutorial 4.1 but include the following remark at the beginning of the program:

Solution to tutorial 4.1

As the remark is to be at the beginning of the program, we shall make it line number 5 as follows:

```
5 REM Solution to tutorial 4.1
```

The space after **REM** in line 5 must not be omitted. The program now becomes:

```
 5 REM Solution to tutorial 4.1
10 CLS
20 PRINT "We are going out now."
30 PRINT "Have you switched off the television?"
```

Enter and **RUN** the program. The computer first executes line 5. However the **REM** statement tells it that this is a remark and is to be ignored. The computer then executes line 10 etc.

A program may contain more than one **REM** statement and they may be placed at any point within the program.

Example 4d

Write, enter and **RUN** a program that displays on the screen a set of instructions to make a telephone call.

A possible program could be:

```
10 REM Making a telephone call
20 CLS
30 PRINT "Lift the receiver."
40 PRINT "Dial the number."
50 PRINT "Chat to the person."
60 PRINT "Replace the receiver."
```

Tutorial

4.5) Repeat Tutorial 4.2 but this time include a **REM** statement at the beginning.

4.6) Repeat Tutorial 4.3 but this time include a **REM** statement at the beginning.

4.7) Write, enter and **RUN** a program that displays on the screen a set of instructions to choose a library book.

4.8) Write, enter and **RUN** a program that displays on the

screen a set of instructions to wash clothes in an automatic washing machine.

Creation of outline shapes

The **PRINT** statement can be used to create some simple shapes and drawings.

Example 4e

Write a program to display the following shape on the screen:

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

A possible program could be:

```
10 REM Draws a rectangle
20 CLS
30 PRINT "**************"
40 PRINT "*            *"
50 PRINT "*            *"
60 PRINT "*            *"
70 PRINT "**************"
```

Example 4f

Write a program that draws a sailing boat on the screen.

A possible program could be (the dashes in line 50 are the underscore symbol **SHIFT Ø** and in lines 130 and 160 are the minus sign):

```
10  REM Draws a sailing boat.
20  CLS
30  PRINT "           <1"
40  PRINT "            1"
```

```
50 PRINT "          _________"
60 PRINT "         )         )"
70 PRINT "          )         )"
80 PRINT "           )         )"
90 PRINT "           )         )"
100 PRINT "          )         )"
110 PRINT "         )         )"
120 PRINT "        )         )"
130 PRINT "        -----------"
140 PRINT "             1"
150 PRINT "             1"
160 PRINT "  -----------------"
170 PRINT "  1              /"
180 PRINT "  1            /"
190 PRINT "vvvvvvvvvvvvvvvvvvvvvv"
```

Tutorial

4.9) Write, enter and RUN a program to give the following screen display for the beginning of a computer game:

```
+++++++++++++++
+             +
+  SPACEGAME  +
+             +
+     by      +
+             +
+  J. Smith   +
+             +
+++++++++++++++
```

4.10) Write, enter and RUN a program that draws a face on the screen, using PRINT statements.

4.11) Write, enter and RUN a program that draws a car on the screen using PRINT statements.

Solutions to the tutorials

When reading these solutions remember that there is often an

alternative way to write a program. If your solution to 4.10 and 4.11 gives the desired result then it is equally acceptable.

4.1)
```
10 CLS
20 PRINT "We are going out now."
30 PRINT "Have you switched off the television?"
```

4.2)
```
10 PRINT "It will soon be holiday time."
20 PRINT "Where are you going this year?"
```

4.3)
```
10 CLS
20 PRINT "I want a library book but I can't make"
30 PRINT "up my mind which book to borrow."
```

4.4)
```
10 CLS
20 PRINT "Please shut the gate or the dog will"
30 PRINT "escape."
```

4.5)
```
 5 REM Solution to tutorial 4.5
10 PRINT "It will soon be holiday time."
20 PRINT "Where are you going this year?"
```

4.6)
```
 5 REM Solution to tutorial 4.6
10 CLS
20 PRINT "I want a library book but I can't make"
30 PRINT "up my mind which book to borrow."
```

4.7)
```
10 REM Choosing a library book.
20 CLS
30 PRINT "Go to the library."
40 PRINT "Look at the books on the shelves."
50 PRINT "Choose your book."
60 PRINT "Take it to the librarian for stamping."
70 PRINT "Give the librarian your ticket."
80 PRINT "Leave with your book."
```

4.8)
```
10 REM Using an automatic washing machine.
20 CLS
30 PRINT "Put the clothes in the machine."
40 PRINT "Shut the door."
50 PRINT "Place the correct amount of washing"
60 PRINT "powder in the dispenser."
70 PRINT "Select the required washing program."
80 PRINT "Switch on the machine."
90 PRINT "Wait until the washing is complete."
```

```
100 PRINT "Open the door."
110 PRINT "Remove the clothes."
```

4.9)
```
10  REM Draws title page of game
20  CLS
30  PRINT "          ++++++++++++++++"
40  PRINT "          +              +"
50  PRINT "          +  SPACEMAZE   +"
60  PRINT "          +              +"
70  PRINT "          +     by       +"
80  PRINT "          +              +"
90  PRINT "          +  J. Smith    +"
100 PRINT "          +              +"
110 PRINT "          ++++++++++++++++"
```

Storing your programs

Enter the program:

```
10 CLS
20 PRINT "It is five o'clock."
```

The computer will remember this program until the power is switched off or until it is instructed to forget it. Switch off the computer, wait a few seconds and then switch it on again. Type:

```
LIST
```

Press an **ENTER** key. You will find that the program cannot be listed as it was lost from the computer's memory when the power was switched off. If you require it again you will have to re-enter it. This could be very time-consuming especially if the program was long. The floppy disk drive to the right of the keyboard can be used to store programs onto a disk. The programs can then be re-entered into the computer's memory when required.

The systems disk for the Amstrad CPC 664

A systems disk is supplied with the computer. The surface of the actual disk is visible through the large elliptical hole and the small round holes on both sides of the disk. As the surfaces are very delicate you should never touch them and you should keep them free of dust by returning the disk to its plastic package when not in use. If the disk is damaged your programs will not load back into the computer and you will have to retype them. Check that the write protection is on as described on page 12 Chapter 1 of the User Instructions Manual. When the write protection is on the computer cannot

write to the disk thus preventing you from accidentally overwriting or erasing the systems programs.

Making a copy of a disk

It is strongly recommended that you make a copy of the systems disk, keep the original as a backup and use the copy. Switch on the computer and insert the systems disk into the floppy disk drive with side one upwards as shown on page 11 Chapter 1 of your User Instructions Manual. Type (where ▌ is **SHIFT** @):

```
|cpm
```

Press an **ENTER** key. The red indicator light will illuminate for a few seconds while the disk is spinning. The computer then displays:

```
CP/M 2.2 - Amstrad Consumer Electronics plc
A>
```

The systems program DISCCOPY can now be used to copy side one of the systems disk onto side one of a new disk. Copy the systems disk by following the instructions on page 74 Chapter 1 of the User Instructions Manual, taking care at each stage not to remove the disk from the floppy disk drive until the red indicator light has gone out. When the computer asks:

```
Do you want to copy another disc (Y/N):
```

Press **Y**. Now copy side two of the systems disk onto side two of the backup disk by inserting the disks with side two upwards. When the second side has been copied put the write protection on the backup disk as described on pages 12 and 13 Chapter 1 of the User Instructions Manual. Now return both disks to their plastic covers. You can return to BASIC if you switch the computer off and then on, or press and hold down both the **CTRL** and **SHIFT** keys. Now press and release the **ESC** key and then release the **CTRL** and **SHIFT** keys. This is called a *keyboard reset*. Both of these methods clear the computer's memory.

You should always remove the disk from the floppy disk drive before switching the computer on or off or you may

damage the disk.

Formatting a disk

Before a disk can store your programs, you must *format* it. Place the backup systems disk in the floppy disk drive with side one upwards and type:

```
|cpm
```

Press an **ENTER** key. The computer then displays:

```
CP/M 2.2 - Amstrad Consumer Electronics plc
A>
```

Type:

```
format
```

Press an **ENTER** key. The computer now displays:

```
FORMAT V2.0
Please insert disc to be formatted into drive A
then press any key:
```

Remove the systems disk and insert the disk with the side that you wish to format upwards. Press the space bar. When the formatting is complete the computer asks if you wish to format another disk. If you wish to format the other side press the **Y** key, turn the disk over and press the space bar.

Saving a program onto a disk

Enter the program:

```
10 CLS
20 PRINT "It is five o'clock."
```

Place a formatted disk into the floppy disk drive. Before you can store the program onto a disk you must decide on a name for the program. The program name must contain less than nine characters made up of letters or numbers. Assume that you decide to call it `Time`. The program can be stored by typing:

```
SAVE "Time"
```

Press an **ENTER** key. The disk spins for a few seconds while it is storing the program. Remove the disk from the floppy disk drive and place it in its plastic cover.

Notice that when naming a program the computer cannot distinguish between upper and lower case letters. Hence the program names

```
Time  TIME  time
```

would all be regarded as the same. Also when naming a program always give it a different name from those already stored on the disk.

Cataloguing the disk

The `CAT` statement will list the names of all the programs currently stored on the disk. Place the disk into the floppy disk drive and type:

```
CAT
```

Press an **ENTER** key. The disk runs for a few seconds after which the computer displays a list of all the programs stored on the disk followed by the amount of free space on the disk. Notice that the program named `TIME` that you stored has been changed to:

```
TIME    .BAS   1K
```

the `BAS` is short for BASIC indicating that it is a BASIC program. The number following the `BAS` gives an indication of the size of the program. As this is not important to us it will not be discussed further.

Loading a program from a disk

Assume that you wish to `LOAD` the program called `Time`, then insert the disk containing the program into the floppy disk drive and type:

```
LOAD "Time"
```

Press an **ENTER** key. The disk runs for a few seconds. When it stops the program is loaded. Return the disk to its cover and type:

```
LIST
```

Press an **ENTER** key. The program should now be displayed on the screen.

Removal of a program from the disk

Place the disk containing the program `Time` into the floppy disk drive. Assume that you wish to delete the program called `Time`, then type:

```
|ERA,"Time.BAS"
```

Press an **ENTER** key. The disk spins for a few seconds. When it stops the program is deleted. Catalogue the disk to verify this. Return the disk to its plastic cover.

Chapter Six

Say it in colour

Although all your previous screen displays have had yellow lettering on a blue background, your Amstrad CPC 664 can display several colours at once depending on the screen display mode.

Screen display mode 1

In this mode the screen has 25 lines, each having up to 40 characters. The computer can display four colours which can be thought of as inks numbered from 0 to 3. The colours of the four inks are shown in Table 6.1.

Ink number	Colour
0	Blue
1	Bright Yellow
2	Bright Cyan
3	Bright Red

Table 6.1

The computer enters mode 1 when it is switched on. It then fills its pen with ink number 1 (i.e. bright yellow) and colours the background with ink number 0 (i.e. blue). Hence it displays bright yellow characters on a blue background.

Screen display mode 0

In this mode the screen has 25 lines each having up to 20 characters. The computer can display sixteen colours which

can again be thought of as inks numbered from 0 to 15. The colours of the sixteen inks are shown in Table 6.2.

Ink number	Colour
0	Blue
1	Bright Yellow
2	Bright Cyan
3	Bright Red
4	Bright White
5	Black
6	Bright Blue
7	Bright Magenta
8	Cyan
9	Yellow
10	Pastel Blue
11	Pink
12	Bright Green
13	Pastel Green
14	Flashing Blue/Bright Yellow
15	Flashing Pink/Sky blue

Table 6.2

To enter mode 0 type (noticing the space between the **MODE** and the **Ø**):

```
MODE Ø
```

Press an **ENTER** key. The computer enters mode 0, clears the screen and displays the usual **Ready** message. Type:

```
PRINT "This is mode Ø."
```

Press an **ENTER** key. You will find that the characters including the cursor are now twice the usual width. This size of print is useful for short phrases but is difficult to read in long sentences. However as this mode has 16 colours it is useful when creating coloured pictures on the screen.

You can return to mode 1 by typing (noticing the space between the **MODE** and the **1**):

```
MODE 1
```

Press an **ENTER** key. The computer enters mode 1, clears the screen and displays the usual **Ready** message.

In the **MODE** statement the space after the word **MODE** must

not be omitted.

The PEN statement

This statement allows you to change the colour of the characters. Enter mode 1. The computer, which can now use the four inks shown in Table 6.1, fills its pen with ink number 1 (i.e. bright yellow). If you wish to display bright red characters then you must instruct the computer to fill its pen with bright red ink which is ink number 3 in Table 6.1. Type (noticing the space between PEN and the 3):

```
PEN 3
```

Press an **ENTER** key. The PEN 3 statement instructs the computer to fill its pen with ink number 3 in Table 6.1 (i.e. bright red). Hence the computer displays its Ready message in bright red ink. Type:

```
PRINT "My pen is filled with bright red ink."
```

Press an **ENTER** key. The computer now displays the sentence on the screen in bright red.

In the PEN 3 statement the space after PEN must not be omitted.

Example 6a

Write a program to select mode 1 and display on the screen in bright cyan:

```
This is mode 1.
My pen is filled with bright cyan ink.
```

In mode 1, bright cyan is ink number 2 and hence the statement PEN 2 will be required to fill the computer's pen with this colour. A possible program could be:

```
10 REM Write bright cyan characters in mode 1
20 MODE 1
30 PEN 2
40 PRINT "This is mode 1."
50 PRINT "My pen is filled with bright cyan ink."
```

Before entering any program it is best to have a pen colour that offers a good contrast with the background. The easiest way to achieve this is to return to the power up colours using a keyboard reset.

Do a keyboard reset, then enter and **RUN** the program. If you wish to list the program after it has been **RUN**, it will be easier to read if it is in the power up colours. However if you switch off or do a keyboard reset then you will clear the computer's memory and lose the program. Type (noticing the spaces between both the words **CALL** and the minus signs):

```
CALL -17409:CALL -17586
```

Press an **ENTER** key. This is called a *software reset* and returns the screen to the power up colours without affecting your program. Now **LIST** your program.

As the numbers in the software reset are not easy to remember, it is helpful to store them in the computer on power up. If you type:

```
key 128,"CALL -17409:CALL -17586"+CHR$(13)
```

and then press an **ENTER** key, the software reset will be stored in the number pad key **f0**. Press the key **f0** and the computer will do a software reset. The information will be retained in this key until you do a keyboard reset or switch off. Notice that the key cannot now be used to enter the number 0.

Example 6b

Extend the program in Example 4f so that a bright yellow sailing boat is drawn in mode 0 with the flag flashing between pink and sky blue and the water being bright blue.

Table 6.2 shows that in mode 0:

- flashing pink and sky blue is ink number 15 and hence the statement **PEN 15** will fill the computer's pen with this colour
- bright blue is ink number 6 and hence the statement **PEN 6** will fill the computer's pen with this colour
- bright yellow is ink number 1 and hence the statement **PEN 1** will fill the computer's pen with this colour.

The program would become:

```
10 REM Draws a sailing boat
20 REM Select mode 0 and flashing pink/sky blue characters
30 MODE 0
40 PEN 15
50 REM Draw the flag
60 PRINT "        <1"
70 REM Select bright yellow characters and draw remainder of
the boat
80 PEN 1
90 PRINT "          1"
100 PRINT "       __________"
110 PRINT "      )         )"
120 PRINT "       )         )"
130 PRINT "        )         )"
140 PRINT "        )         )"
150 PRINT "       )         )"
160 PRINT "      )         )"
170 PRINT "     )         )"
180 PRINT "     -----------"
190 PRINT "          1"
200 PRINT "          1"
210 PRINT "----------------"
220 PRINT "1             /"
230 PRINT "1           /"
240 REM Select bright blue characters for the water
250 PEN 6
260 PRINT "vvvvvvvvvvvvvvvvvv"
```

Do a software reset and use the **NEW** statement to forget the previous program. Enter the above program and **LIST** it. As the program has 26 lines and the screen has only 25 lines the first line will have scrolled off the top of the screen. Type (noticing that the symbol between **10** and **100** is a minus sign and that there is a space between **LIST** and **10**):

```
LIST 10-100
```

Press an **ENTER** key. The **10—100** in the **LIST** statement instructs the computer to list only lines 10 to 100. In this way you can **LIST** part of a program.

Tutorial

Before entering a program, remember to do a software reset and use the NEW statement to forget the previous one.

6.1) Using mode 1, write, enter and RUN a program which displays in bright red:

```
The ships crashed in the fog.
```

6.2) Using mode 0, write, enter and RUN a program that displays in pink:

```
GAME OVER
```

6.3) Using mode 0, write, enter and RUN a program which displays in bright white:

```
Traffic jam ahead.
```

and then in flashing pink/sky blue:

```
Turn off at the next junction.
```

The PAPER statement

Do a software reset and use the NEW statement to forget the previous program. In all your previous screen displays the background colour has always been blue. The PAPER statement allows you to select the background colour from one of your inks. Lets assume that in mode 1 you wish the background colour to be bright red. As bright red is ink number 3, type (noticing the space between PAPER and 3):

```
PAPER 3
```

Press an **ENTER** key. The PAPER 3 statement instructs the computer to fill the background with ink number 3 (i.e. bright red).

The Ready message is now bright yellow on a bright red background but the rest of the screen still has its blue background. The background only changes to the new colour when a character is displayed on the screen. The colour of the cursor which becomes sky blue is controlled by the computer and not by the PAPER statement. Type:

```
PRINT "The background is bright red."
```

Press an **ENTER** key. The computer displays on the screen in bright yellow on a bright red background:

```
The background is bright red.
```

Again notice that the bright red background only appears where characters have been displayed on the screen.

Clear the screen using the CLS statement. The centre of the screen will now be bright red surrounded by a blue border. This border has always been there but was not noticeable since it was the same colour as the centre of the screen. The computer can only write on the centre of the screen.

In the PAPER 3 statement the space after PAPER must not be omitted.

Example 6c

Write a program in mode 0 which displays in black characters on a flashing background of pink and sky blue:

```
GAME OVER
```

Table 6.2 shows that in mode 0:

- flashing pink and sky blue is ink number 15 and hence the statement PAPER 15 will make the background this colour
- black is ink number 5 and hence the statement PEN 5 will fill the computer's pen with this colour.

Do a software reset and use the NEW statement to forget the previous program. The program could be:

```
10  REM End of game display
20  REM Select mode 0 and flashing pink/sky blue background
30  MODE 0
40  PAPER 15
50  REM Clear screen to flashing pink/sky blue
60  CLS
70  REM Select black characters
80  PEN 5
90  REM Display the text
100 PRINT "GAME OVER"
```

Tutorial

Before entering a program, remember to do a software reset and use the NEW statement to forget the previous one.

6.4) Write a program in mode 1 that selects bright yellow characters on a bright red background and displays:

```
My name is John.
```

6.5) Change Example 6c to display GAME OVER on a pastel green background and then add an extra line:

```
DO YOU WISH TO PLAY AGAIN?
```

displayed in bright cyan on the same background.

Multicoloured backgrounds

It is possible to have several background colours on the screen at the same time.

Do a software reset and use the NEW statement to forget the previous program. Let's assume you would like to display on the screen in mode 1:

```
This is a bright cyan background.
This is a bright red background.
```

where the first line should have a bright cyan and the second line a bright red background.

A possible program could be:

```
10 REM Illustration of different backgrounds
20 REM Select mode 1
30 MODE 1
40 REM Select a bright cyan background
50 PAPER 2
60 PRINT "This is a bright cyan background."
70 REM Select a bright red background
80 PAPER 3
90 PRINT "This is a bright red background."
```

Enter and RUN the program. The computer displays on the screen with a bright cyan background:

```
This is a bright cyan background.
```

and then displays on the screen with a bright red background:

```
This is a bright red background.
```

The background of the rest of the screen remains blue.

Example 6d

Write a program to display in mode 1 in bright red on a bright yellow background.

```
Where has the ball gone?
```

It should then display in bright yellow on a bright red background:

```
I think it is over there.
```

Do a software reset and use the NEW statement to forget the previous program. Enter the program which could be:

```
10  REM Example 6.d
20  REM Select mode 1
30  MODE 1
40  REM Select bright yellow background and bright red
characters
50  PAPER 1
60  PEN 3
70  PRINT "Where has the ball gone?"
80  REM Select bright red background and bright yellow
characters
90  PAPER 3
100 PEN 1
110 PRINT "I think it is over there."
```

Tutorial

Before entering a program, remember to do a software reset and use the NEW statement to forget the previous one.

6.6) Write a program in mode 1 which clears the screen in bright cyan and then displays in bright red on a bright yellow background:

```
We are going out now.
```

It should then change to bright cyan on a bright red background and display:

```
Have you switched off the television?
```

6.7) Write a program in mode 0 which clears the screen in bright yellow and then displays in pink:

```
My birthday is on the 19th April.
```

It should then change the background to bright cyan and display in bright green:

```
When is your birthday?
```

Solutions to the tutorials

6.1)
```
10 REM Tutorial 6.1
20 REM Select mode 1
30 MODE 1
40 REM Select bright red characters
50 PEN 3
60 PRINT "The ships crashed in the fog."
```

6.2)
```
10 REM End of game display
20 REM Select mode 0
30 MODE 0
40 REM Select pink characters
50 PEN 11
60 PRINT "GAME OVER"
```

6.3)
```
10 REM Tutorial 6.3
20 REM Select mode 0
30 MODE 0
40 REM Select bright white characters
50 PEN 4
60 PRINT "Traffic jam ahead."
70 REM Select flashing pink/sky blue characters
80 PEN 15
90 PRINT "Turn off at the"
100 PRINT "next junction."
```

6.4)
```
10 REM Tutorial 6.4
20 REM Select bright red background
30 PAPER 3
40 PRINT "My name is John."
```

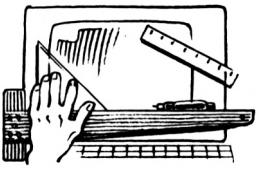
6.5)
```
10 REM End of game display
20 REM Select mode 0 and pastel green background
30 MODE 0
40 PAPER 13
50 REM Clear the screen to pastel green
60 CLS
70 REM Select black characters
80 PEN 5
90 PRINT "GAME OVER"
100 REM Select bright cyan characters
110 PEN 2
120 PRINT "DO YOU WISH TO"
130 PRINT " PLAY AGAIN?"
```

6.6)
```
10 REM Tutorial 6.6
20 REM Select mode 1
30 MODE 1
40 REM Select bright cyan background
50 PAPER 2
60 REM Clears the screen to bright cyan
70 CLS
80 REM Select bright yellow background
90 PAPER 1
100 REM Select bright red characters
110 PEN 3
120 PRINT "We are going out now."
130 REM Select bright red background
140 PAPER 3
150 REM Select bright cyan characters
160 PEN 2
170 PRINT "Have you switched off the television?"
```

6.7)
```
10 REM Tutorial 6.7
20 REM Select mode 0
30 MODE 0
40 REM Select bright yellow background
50 PAPER 1
60 REM Clear the screen to bright yellow
70 CLS
80 REM Select pink characters
90 PEN 11
100 PRINT "My birthday is on"
```

```
110 PRINT "the 19th April."
120 REM Select bright cyan background
130 PAPER 2
140 REM Select bright green characters
150 PEN 12
160 PRINT "When is your"
170 PRINT "  birthday?"
```

Designing your screen layout



Enter and RUN the program:

```
10 PRINT "SPACE GAME"
20 PRINT "by J Smith."
```

The screen displays:

```
SPACE GAME
by J Smith.
```

After the computer had executed the PRINT statement at line 10, it automatically moved the cursor to the beginning of the next line. As the cursor indicates the position of the next character on the screen the phrase:

```
by J Smith.
```

was displayed on this line. A better screen display would be obtained if the phrases had been centred on the screen with a few lines between them. If you could position the cursor on the screen then you would be able to create your own screen layout. The statements introduced in this chapter will enable you to do this.

A screen display planner for mode 1

Screen displays should first be drawn on a screen display planner. As the mode 1 screen contains 25 lines each of 40 characters, a screen display planner can be made by dividing a sheet of A4 paper (29.7 cm. by 21 cm.) into 40 columns and 25 rows as shown in Fig. 7.1. Each box on your screen display planner should be almost a square, representing one character on the screen.

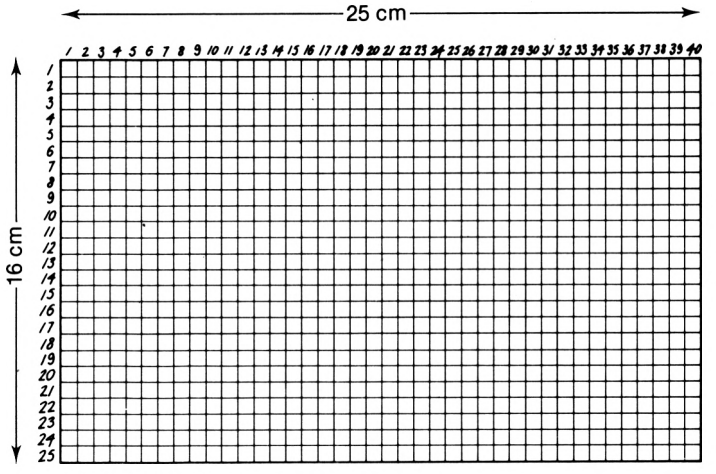


Figure 7.1 Mode 1 screen display planner

Keep this as your master copy and use tracing paper when designing your screen displays.

A screen display planner for mode 0

In mode 0 the screen contains 25 lines each of 20 characters. A screen display planner can be made by dividing a sheet of A4 paper (29.7 cm. by 21 cm.) into 20 columns and 25 rows as

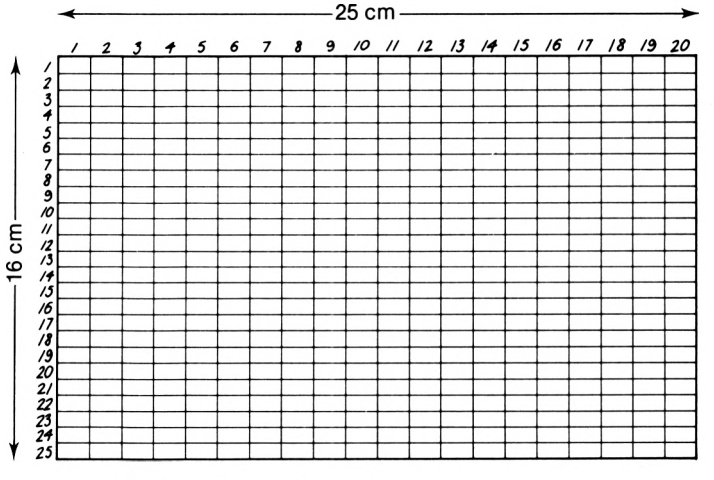


Figure 7.2 Mode 0 screen display planner

shown in Fig. 7.2. Each box on your screen display planner should be approximately twice as wide as it is high and represents one character on the screen.

Keep this as your master copy.

Tutorial

7.1) In the screen display planner shown in Fig. 7.3 determine the column and row number of each character.

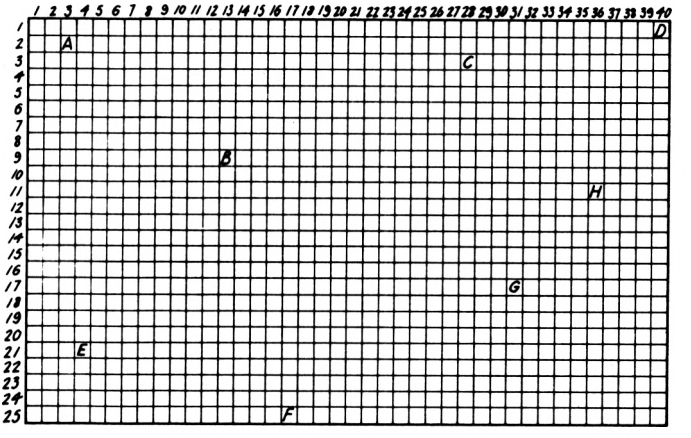


Figure 7.3 Screen display planner for Tutorial 7.1

7.2) Mark the following characters on your mode 0 screen display planner:

Character	Column	Row
A	1	24
B	19	2
H	10	8
,	20	25
4	1	1
p	2	19

The LOCATE statement

The `LOCATE` statement is used to move the cursor to a new

position on the screen. The statement:

```
LOCATE 5,10
```

moves the cursor to column 5 and row 10. We shall call this the screen position 5,10. The space after **LOCATE** in the above statement must not be omitted.

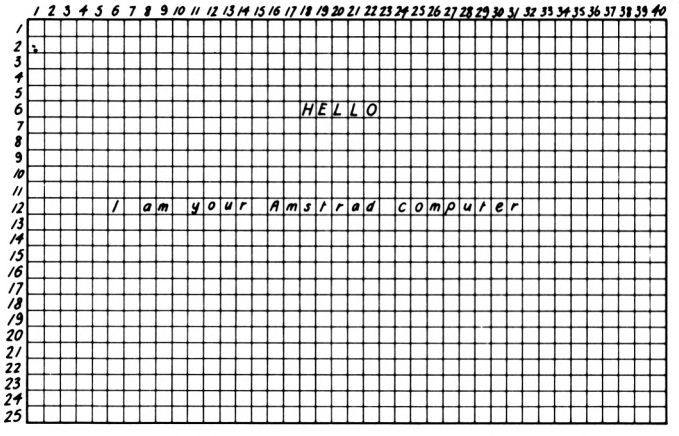


Figure 7.4 Mode 1 screen display of text message

Assume that you wish to create the screen display in mode 1, shown in Fig. 7.4. When creating a screen display you should always start by clearing the screen. As the **H** of **HELLO** is at column 18 and row 6 the cursor should be moved to screen position 18,6 using the **LOCATE** statement:

```
LOCATE 18,6
```

The word **HELLO** can then be displayed using a **PRINT** statement.

As the next line starts at screen position 6,12 a **LOCATE** statement can be used to move the cursor to this position and then the line can be displayed. The complete program to give the screen display could be:

```
10 REM Screen display
20 CLS
30 LOCATE 18,6
40 PRINT "HELLO"
50 LOCATE 6,12
60 PRINT "I am your Amstrad computer"
```

Enter and **RUN** the program. The computer displays the two lines as required but on completing the program the computer also displays:

```
Ready
```

The easiest way to suppress the **Ready** is to display it in the background colour. You could add a line 70:

```
70 PEN 0
```

This would tell the computer to fill its pen with blue ink which is the same as the background colour. Hence the **Ready** would not be visible. Add this line to your program and **RUN** it. The **Ready** is now invisible.

Tutorial

7.3) Change the program of Example 4d to obtain a good screen display.

7.4) Change the program of Example 6c to obtain a good screen display.

Use of the semi-colon in the PRINT statement

Enter and **RUN** the program:

```
10 PRINT "cat"
20 PRINT "dog"
```

The computer displays:

```
cat
dog
```

Now put a semi-colon at the end of line 10 as follows:

```
10 PRINT "cat";
20 PRINT "dog"
```

RUN the program. At line 10 the computer displays:

```
cat
```

but since it is followed by a semi-colon, it does not position the cursor at the start of the next line as usual but leaves it at the end of the word `cat`. Hence at line 20 the word `dog` is displayed next to `cat` and not on the next line as previously.

Placing a semi-colon after the quotation marks in a `PRINT` statement tells the computer not to move the cursor to the beginning of the next line. This can be useful if you wish to display `cat` and `dog` in different colours. Assume `cat` is to be written in bright red and `dog` in bright cyan. The program could be modified to:

```
10 REM Use of the ; in the PRINT statement.
20 REM Select bright red characters
30 PEN 3
40 PRINT "cat";
50 REM Select bright cyan characters
60 PEN 2
70 PRINT "dog"
```

Example 7a

Write a program in mode 0 that displays in the centre of the screen:

```
DANGER!-ICE
```

The computer should display `DANGER!-` in bright red on a bright white background and `ICE` in black on a bright magenta background. The rest of the screen should be blue.

A possible program could be:

```
10 REM Warning sign
20 REM Select mode 0
30 MODE 0
40 REM Select bright red characters and bright white
background
50 PEN 3
60 PAPER 4
70 REM Locate and display DANGER!-
80 LOCATE 4,12
90 PRINT "DANGER!-";
100 REM Select black characters and bright magenta
background
```

```
110 PEN 5
120 PAPER 7
130 REM Display ICE
140 PRINT "ICE"
150 REM Hide the cursor
160 PAPER 0
170 PEN 0
```

Tutorial

7.5) Write a program in mode 0 to display in the centre of the screen:

```
Monday 1st April
```

The computer should display the day in bright yellow on a blue background and the date in black on a blue background.

7.6) Write a program in mode 1 to display in the centre of the screen:

```
The background is bright red.
```

The computer should display the words `The background is` in the usual screen colours and the words `bright red` in bright yellow on bright red.

7.7) Change the program of Example 6a so that the words `bright cyan` are displayed in bright cyan on a blue background and the other words are in bright yellow on a blue background.

7.8) Extend your program for Tutorial 4.9 so that in mode 0

Solutions to the tutorials

7.1)

Character	Column	Row
A	3	2
B	13	9
C	28	3
D	40	1
E	4	21
F	17	25
G	31	17
H	36	11

the display is centred on the screen with the + signs in bright yellow and the text in bright red on a blue background.

7.2)

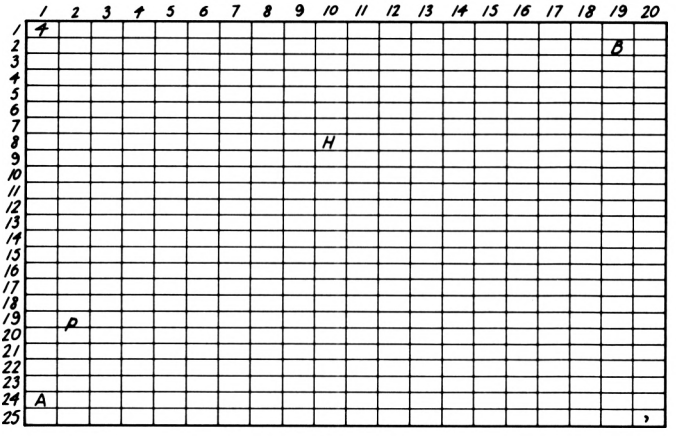


Figure 7.5

7.3)
```
10 REM Making a telephone call
20 REM Select mode 1
30 MODE 1
40 CLS
50 LOCATE 8,10
60 PRINT "Lift the receiver."
70 LOCATE 8,12
80 PRINT "Dial the number."
90 LOCATE 8,14
100 PRINT "Chat to the person."
110 LOCATE 8,16
120 PRINT "Replace the receiver."
```

7.4)
```
10 REM End of game display
20 REM Select mode 0 and flashing pink/sky blue background
30 MODE 0
40 PAPER 15
50 REM Clear the screen to flashing pink/sky blue
60 CLS
70 REM Select black characters
80 PEN 5
90 REM Locate and display the text
```

```
100 LOCATE 5,12
110 PRINT "GAME OVER"
```

7.5)
```
10 REM Displays the date
20 REM Select mode 0
30 MODE 0
40 LOCATE 2,10
50 PRINT "Monday ";
60 REM Select black characters
70 PEN 5
80 PRINT "1st April"
```

7.6)
```
10 REM Tutorial 7.6
20 REM Select mode 1
30 MODE 1
40 LOCATE 5,10
50 PRINT "The background is ";
60 REM Select bright red background
70 PAPER 3
80 PRINT "bright red."
```

7.7)
```
10 REM Tutorial 7.7
20 REM Select mode 1
30 MODE 1
40 LOCATE 12,8
50 PRINT "This is mode 1."
60 LOCATE 2,10
70 PRINT "My pen is filled with ";
80 REM Select bright cyan characters
90 PEN 2
100 PRINT "bright cyan";
110 REM Select bright yellow characters
120 PEN 1
130 PRINT " ink."
```

7.8)
```
10 REM Draws title page of game
20 REM Select mode 0
30 MODE 0
40 REM Draw the + signs first in bright yellow
50 LOCATE 3,8
60 PRINT "++++++++++++++++"
70 LOCATE 3,9
80 PRINT "+              +"
90 LOCATE 3,10
100 PRINT "+              +"
110 LOCATE 3,11
120 PRINT "+              +"
130 LOCATE 3,12
140 PRINT "+              +"
150 LOCATE 3,13
160 PRINT "+              +"
170 LOCATE 3,14
180 PRINT "+              +"
190 LOCATE 3,15
200 PRINT "+              +"
210 LOCATE 3,16
220 PRINT "++++++++++++++++"
230 REM Display the text
240 REM Select bright red characters
250 PEN 3
260 LOCATE 6,10
270 PRINT "SPACEMAZE"
280 LOCATE 9,12
290 PRINT "by"
300 LOCATE 6,14
310 PRINT "J. Smith"
320 REM Move cursor to line 18 and hide it
330 LOCATE 1,18
340 PEN 0
```

Drawing with your Amstrad CPC 664

The building blocks

This chapter will show you how to make your Amstrad CPC 664 draw pictures composed of standard building blocks just as in a mosaic.

Type:

```
MODE 0
```

Press an **ENTER** key. Type (remembering that brackets are the **SHIFT 8** and **9** keys and noticing the space between `PRINT` and `CHR$`):

```
PRINT CHR$(224)
```

Press an **ENTER** key. The screen will display building block number 224 which is a smiling face. A complete list of the building blocks is given in Chapter 7 pages 9 to 20 of your User Instructions Manual. Block 224 is drawn on page 18. Beneath the drawing is the block number (224) followed by two other numbers that you will not require to use.

You will see that the drawing consists of an 8 by 8 pattern of black and white dots. The white dots are represented on your monitor as the background colour (blue) and the black dots as the pen colour (bright yellow). The 8 by 8 pattern occupies one box on your mode 0 screen display planner.

The building blocks numbered 32 to 122 and number 163 are the keyboard characters that you have been using throughout the book. Type:

```
PRINT "h"
```

Press an **ENTER** key. Now type:

```
PRINT CHR$(104)
```

Press an **ENTER** key. Both of these PRINT statements display a lower case h.

The building blocks numbered 123 to 255 are general designs that can be fitted together to make pictures and shapes. However some of them form particular patterns, as shown in Table 8.1,

Block number	Pattern
160–162	French and German accents
163–166	pound, copyright, pilcrow and section signs
168–172	arithmetic symbols
174	hook
176–191	Greek alphabet
224	happy face
225	sad face
226–229	card suits
234,235	male and female symbols
236,237	musical symbols
238	star
239	rocket
240–247	arrows
248–251	dancing man
252	bomb
253	mushroom cloud

Table 8.1

which gives suggested interpretations of the designs, but the image they create in the mind will depend on their surroundings, colour and the thoughts of the viewer. Could block 229 be a spade in a pack of cards, a tree or an aeroplane? You will find that these designs can be used in numerous ways to create figures and pictures, the only limitation being the ingenuity of your mind!

To display building block number 253 type:

```
PRINT CHR$(253)
```

Press an **ENTER** key. Any other block can be displayed by using its number in place of the 253. Display a selection of the building blocks on the screen so that you have an image of them in your mind.

Two characters can be displayed together by typing:

```
PRINT CHR$(224);CHR$(225)
```

Press an **ENTER** key. The semi-colon tells the computer to display the sad face next to the happy one rather than on the next line. If you wish to display the two faces on the same line but with two spaces between them then type:

```
PRINT CHR$(224);"  ";CHR$(225)
```

Press an **ENTER** key. The computer displays the happy and the sad face with two spaces between them.

Designing your own picture

You will find it easier to create a picture if you first plan it on your screen display planner. Assume that you drew a robot as shown in Fig. 8.1. If the picture is to be displayed in pink on a bright green background, then the first few lines of the program would be:

```
10 REM Draw a picture in mode 0
20 MODE 0
30 REM Select bright green background
40 PAPER 12
50 REM Clear the screen to bright green
60 CLS
70 REM Select pink characters
80 PEN 11
```

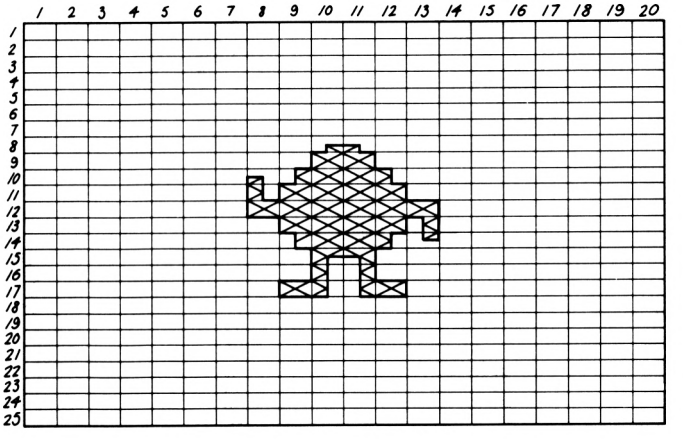


Figure 8.1 Mode 0 screen display planner showing robot

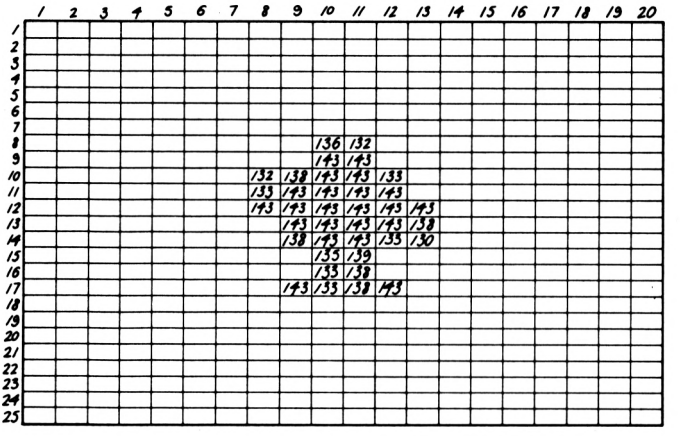


Figure 8.2 Robot figure building-block numbers

The next stage is to convert your drawing into its building block numbers. These can be marked on another screen planner as shown in Fig. 8.2. Each line of the display will be displayed on the screen by using a **LOCATE** and then a **PRINT** statement. As the top line of the drawing starts at screen position 10,8 it could be displayed by the lines:

```
90 REM Draw the robot
100 LOCATE 10,8
110 PRINT CHR$(136);CHR$(132)
```

The next line could be displayed by:

```
120 LOCATE 10,9
130 PRINT CHR$(143);CHR$(143)
```

The other lines in the drawing could be displayed in the same way. The complete program could be:

```
10 REM Draw a picture in mode 0
20 MODE 0
30 REM Select bright green background
40 PAPER 12
50 REM Clear the screen to bright green
60 CLS
70 REM Select pink characters
80 PEN 11
90 REM Draw the robot
100 LOCATE 10,8
```

```
110 PRINT CHR$(136);CHR$(132)
120 LOCATE 10,9
130 PRINT CHR$(143);CHR$(143)
140 LOCATE 8,10
150 PRINT CHR$(132);CHR$(138);CHR$(143);CHR$(143);
CHR$(133)
160 LOCATE 8,11
170 PRINT CHR$(133);CHR$(143);CHR$(143);CHR$(143);
CHR$(143)
180 LOCATE 8,12
190 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(143)
200 LOCATE 9,13
210 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
CHR$(138)
220 LOCATE 9,14
230 PRINT CHR$(138);CHR$(143);CHR$(143);CHR$(133);
CHR$(130)
240 LOCATE 10,15
250 PRINT CHR$(135);CHR$(139)
260 LOCATE 10,16
270 PRINT CHR$(133);CHR$(138)
280 LOCATE 9,17
290 PRINT CHR$(143);CHR$(133);CHR$(138);CHR$(143)
```

Enter the above program and **RUN** it. The computer should display your robot. If anything is wrong then do a software reset before listing the program to look for your mistake.

SAVE the program onto a disk and call it **ROBOT**.

Example 8a

Modify the above program so that the robot has cyan hands.

As his hands are in screen positions 8,10 and 13,14 lines 150 and 230 will require to be changed. Line 150 displays from screen position 8,10 to 12,10. It must be modified so that it displays screen position 8,10 in cyan and the others in pink. The line could be expanded to:

```
150 REM Select cyan characters
151 PEN 8
152 REM Draw his hand
153 PRINT CHR$(132);
154 REM Select pink characters
```

```
155 PEN 11
156 REM Display the rest of the line
157 PRINT CHR$(138);CHR$(143);CHR$(143);CHR$(133)
```

Similarly line 230 could be expanded to:

```
230 REM Display all but his hand
231 PRINT CHR$(138);CHR$(143);CHR$(143);CHR$(133);
232 REM Select cyan characters
233 PEN 8
234 REM Draw his hand
235 PRINT CHR$(130)
236 REM Select pink characters
237 PEN 11
```

Change these lines and RUN the program. The robot's hands should now be cyan.

SAVE the program onto a disk and call it ROBOT8a.

Tutorial

8.1) If you assume that the top two rows of the above robot are his hat, then modify the program called ROBOT8a to make his hat yellow. SAVE the program onto a disk and call it ROBOT81.

8.2) Modify the program ROBOT so that both his hands are pointing upwards.

8.3) If you assume that the last two rows of the above robot are his boots, then modify the program called ROBOT81 to make his boots black. SAVE the program onto a disk and call it ROBOT83.

8.4) Modify the program called ROBOT83 so that the computer displays in black beneath the robot the words:

```
MY ROBOT
```

8.5) Modify the program ROBOT so that the robot holds a pink flag in his right hand as shown in Fig. 8.3.

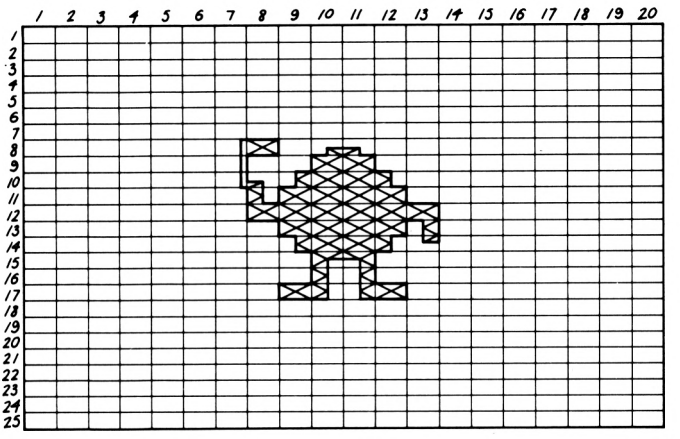


Figure 8.3 Robot figure waving a flag

Transparent printing

Assume that you wish to give the robot eyes by displaying a lower case o in black at screen positions 10,10 and 11,10. This could be achieved by adding the following lines to the program ROBOT83:

```
300 REM Give the robot eyes
310 REM Select black characters
320 PEN 5
330 LOCATE 10,10
340 PRINT "o";"o"
```

Add these lines to the program and RUN it. The computer displays the o's in black but it hides part of the robot's face as it fills the rest of the box with the background colour which is bright green.

A solution is to use transparent printing. This instructs the computer to display only the pattern on the building block and ignore the background. The statement:

```
PRINT CHR$(22)+CHR$(1)
```

instructs the computer to use transparent printing and the statement:

```
PRINT CHR$(22)+CHR$(0)
```

instructs it to return to normal printing.

Add the following lines to the program ROBOT83:

```
300 REM Give the robot eyes
310 REM Select transparent printing
320 PRINT CHR$(22)+CHR$(1)
330 REM Select black characters and display his eyes
340 PEN 5
350 LOCATE 10,10
360 PRINT "o";"o"
370 REM Return to normal printing
380 PRINT CHR$(22)+CHR$(0)
```

RUN the program. The computer will now display the eyes in black but the original pink background remains.

SAVE the program onto a disk and call it BLACKEYE.

Tutorial

8.6) Modify the program ROBOT83 so that the robot's boots have buttons which flash from pink to sky blue. The buttons should be displayed using building block 162 at screen positions 9,17 and 12,17.

8.7) Modify the program BLACKEYE so that the robot has a black nose. The nose should consist of building blocks 194 and 195 at screen positions 10,11 and 11,11.

Designing your own building blocks

One advantage of transparent printing is that the simple building blocks can be superimposed on each other to produce more complicated patterns. Assume you wish to colour the flag that the robot held in Tutorial 8.5. A suggested colour scheme is shown in Fig. 8.4.

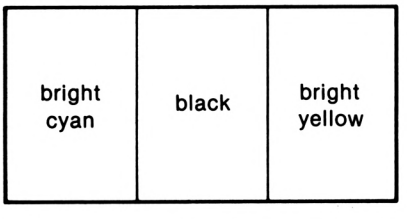


Figure 8.4 Suggested colour scheme for flag

This could be constructed by:

1) Displaying building block 143 in black—this would colour the screen box black.
2) Selecting transparent printing.
3) Selecting bright cyan characters.
4) Superimposing building block 211 on the black screen box—this displays the bright cyan line at the left edge of the screen box.
5) Selecting bright yellow characters.
6) Superimposing building block 209 on the screen box—this displays the bright yellow line at the right edge of the screen box.
7) Returning to normal printing.

The following program would display the flag at screen position 8,8:

```
10 REM Designing your own building blocks
20 REM Select mode 0
30 MODE 0
40 REM Select black characters
50 PEN 5
60 LOCATE 8,8
70 PRINT CHR$(143)
80 REM Select transparent printing
90 PRINT CHR$(22)+CHR$(1)
100 REM Select bright cyan characters
110 PEN 2
120 LOCATE 8,8
130 PRINT CHR$(211)
140 REM Select bright yellow characters
150 PEN 1
160 LOCATE 8,8
170 PRINT CHR$(209)
180 REM Return to normal printing
190 PRINT CHR$(22)+CHR$(0)
```

Enter and RUN the program. The computer should display the desired pattern.

Example 8b

Assume that you wish to add blue eyebrows to the robot's eyes

by superimposing building block 126 on his eyes.

Add the following lines to the program called ROBOT83:

```
300 REM Give the robot eyes
310 REM Select transparent printing
320 PRINT CHR$(22)+CHR$(1)
330 REM Select black characters and display his eyes
340 PEN 5
350 LOCATE 10,10
360 PRINT "o";"o"
370 REM Select blue characters and display his eyebrows
380 PEN 0
390 LOCATE 10,10
400 PRINT CHR$(126);CHR$(126)
410 REM Return to normal printing
420 PRINT CHR$(22)+CHR$(0)
```

Tutorial

8.8) Superimpose the building blocks 204,205 and 144 in mode 0 to make the pattern of building block 203 but with the two diagonals and the centre being different colours.

Line graphics

So far pictures have been created from the Amstrad CPC 664

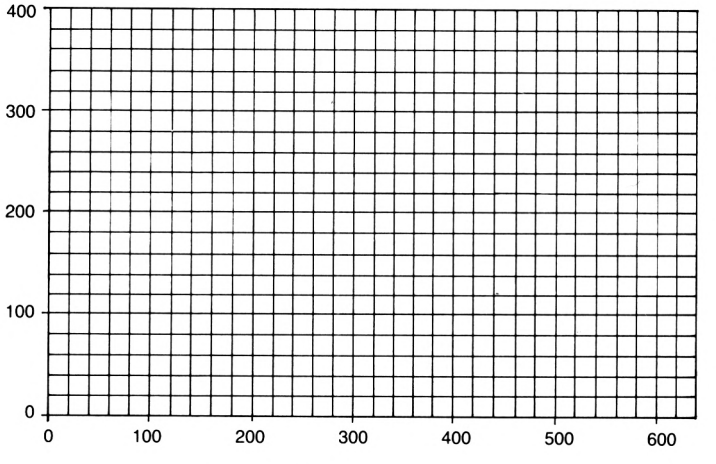


Figure 8.5 Graphics planner

building blocks. It is also possible to create pictures by drawing outline shapes and then colouring them. As before it is best to design your picture on a graphics planner. A graphics planner can be made from a sheet of graph paper as shown in Fig. 8.5. The horizontal direction is divided into 640 points and the vertical direction into 400. Notice that unlike the screen display planners of the previous chapter the vertical axis is numbered from the bottom of the screen upwards.

Example 8c

In the graphics planner shown in Fig. 8.6 find the coordinates of the point A.

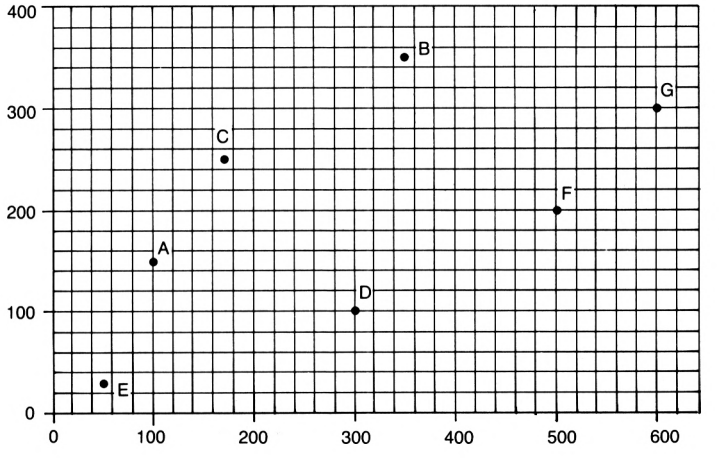


Figure 8.6 Graphics planner for Example 8c

In the horizontal direction the A is at point 100. The point A is said to have an x coordinate of 100. In the vertical direction the A is at point 150. The point A is said to have a y coordinate of 150. The coordinates of point A are written as 100,150.

Tutorial

8.9) State the coordinates of the other points on the graphics planner of Example 8c.

8.10) Plot the following points on a graphics planner:

Name	Coordinates
A	550,350
B	75,100
C	450,0
D	0,100
E	200,375
F	350,250

The graphics colours

The graphics colours are the same as the text colours. The **GRAPHICS PEN** and **GRAPHICS PAPER** statements select the graphics pen and background colours the same way that the **PEN** and **PAPER** statements select the text colours. Type (noticing the space on both sides of **PAPER**):

```
GRAPHICS PAPER 2
```

Press an **ENTER** key. The graphics background colour is now bright cyan but will not be displayed until the graphics screen is cleared. Type:

```
CLG
```

Press an **ENTER** key. The CLear Graphics screen (**CLG**) statement clears the screen to the graphics background colour bright cyan. Notice that the **Ready** message is still in the usual colours as only the graphic colours have been changed and not the text colours.

The MOVE and DRAW statements

The statement:

```
MOVE 10,20
```

instructs the computer to move the graphics cursor to the screen point 10,20. The statement:

```
DRAW 650,30
```

instructs the computer to draw a line from the current position of the graphics cursor to the screen point 650,30 in the graphics

pen colour.

Consider the program:

```
10 MODE 1
20 GRAPHICS PEN 2
30 MOVE 10,20
40 DRAW 650,30
```

Enter and **RUN** the program. Line 10 sets up mode 1 and clears the screen. Line 20 changes the graphics pen colour to bright cyan. At line 30 the graphics cursor is moved to screen position 10,20 but nothing is drawn on the screen. Line 40 instructs the computer to draw a line in the graphics pen colour bright cyan from the current graphics cursor position, i.e. from the point 10,20 to the point 650,30. As the program is now complete the computer displays the usual **Ready** message. Notice that this message is displayed in the text colour (bright yellow) showing that the **GRAPHICS PEN** statement did not affect the text colour. Also the message is displayed at the top left hand corner of the screen showing that the **MOVE** and **DRAW** statements do not affect the position where the text is to be written.

Example 8d

Write a program to draw the outline of a bright yellow house with two blue windows, a bright red door and roof as shown in Fig. 8.7.

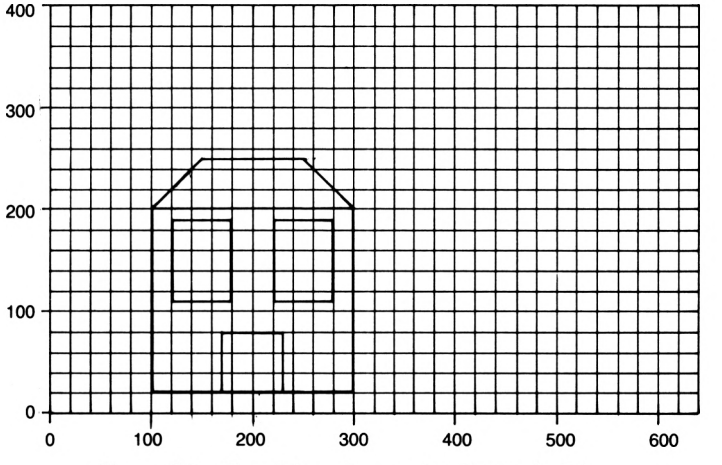


Figure 8.7 Graphics planner for Example 8d

A possible program could be:

```
10 REM Draw the outline of a house
20 REM Select mode 1
30 MODE 1
40 REM Draw the bright yellow house
50 GRAPHICS PEN 1
60 MOVE 100,20
70 DRAW 100,200
80 DRAW 300,200
90 DRAW 300,20
100 DRAW 100,20
110 REM Draw the bright red roof
120 GRAPHICS PEN 3
130 MOVE 100,200
140 DRAW 150,250
150 DRAW 250,250
160 DRAW 300,200
170 DRAW 100,200
180 REM Draw the bright red door
190 MOVE 170,20
200 DRAW 170,80
210 DRAW 230,80
220 DRAW 230,20
230 DRAW 170,20
240 REM Draw the bright cyan windows
250 GRAPHICS PEN 2
260 MOVE 120,110
270 DRAW 120,190
280 DRAW 180,190
290 DRAW 180,110
300 DRAW 120,110
310 MOVE 220,110
320 DRAW 220,190
330 DRAW 280,190
340 DRAW 280,110
350 DRAW 220,110
360 REM Hide the cursor
370 PEN 0
```

SAVE the program onto a disk and call it HOUSE.

Tutorial

8.11) Draw the outline of a large capital A in bright red ink on

a black background using mode 0. **SAVE** the program onto a disk and call it **A**.

Enter and **RUN** the program:

```
10 MODE 1
20 GRAPHICS PEN 2
30 MOVE 10,20
40 DRAW 650,300
```

Now change line 10 to:

```
10 MODE 0
```

and **RUN** the program. Observe that in mode 0 the line is more ragged than in mode 1. Mode 1 is said to have a *higher resolution* than mode 0. Hence any picture is a compromise between the higher resolution giving smoother lines and the restriction of only having four colours.

The FILL statement

The **FILL** statement can be used to fill an area of the screen that is enclosed by drawn lines. Enter the program:

```
10 REM Demonstration of filling an area
20 MODE 1
30 GRAPHICS PEN 3
40 MOVE 200,100
50 DRAW 200,150
60 DRAW 300,150
70 DRAW 300,100
80 DRAW 200,100
```

RUN the program which draws a bright red rectangle on the screen. To fill the rectangle in bright cyan the graphics cursor must first be moved to within the rectangle by the statement:

```
MOVE 250,125
```

The rectangle can then be filled by the statement:

```
FILL 2
```

The complete program becomes:

```
10 REM Demonstration of filling an area
20 MODE 1
30 GRAPHICS PEN 3
40 MOVE 200,100
50 DRAW 200,150
60 DRAW 300,150
70 DRAW 300,100
80 DRAW 200,100
90 MOVE 250,125
100 FILL 2
```

It is important to realise that the outline round the area to be filled must be completely closed by only one colour otherwise the colour will spill out and fill the complete screen.

Example 8e

Draw, in mode 0, a triangle and colour it black on a bright red background.

A suitable program could be:

```
10 REM Use of the FILL command
20 MODE 0
30 REM Select a bright red background
40 GRAPHICS PAPER 3
50 CLG
60 REM Draw the black triangle
70 GRAPHICS PEN 5
80 MOVE 400,10
90 DRAW 500,200
100 DRAW 600,150
110 DRAW 400,10
120 REM Fill the triangle in black
130 MOVE 500,100
140 FILL 5
```

Example 8f

Extend Example 8e so that a bright green rectangle is drawn inside the triangle.

When filling overlapping shapes the background must be

drawn and filled first. Add the following lines to the program of Example 8e:

```
150 REM Draw the rectangle
150 GRAPHICS PEN 12
160 MOVE 500,150
170 DRAW 500,170
180 DRAW 520,170
190 DRAW 520,150
200 DRAW 500,150
210 REM Fill the rectangle in bright green
220 MOVE 510,160
230 FILL 12
```

Tutorial

8.12) Extend the program of Tutorial 8.11 so that the letter A is filled with bright red ink. SAVE the program onto a disk and call it AFILL.

8.13) Extend the program of Example 8d so that the house is coloured.

Writing text on a picture

The TAG statement can be used to position text at any desired position on a picture. Add the following lines to the program HOUSE:

```
370 REM Write text on the picture
380 REM Select graphics colours bright red on bright yellow
390 GRAPHICS PEN 3
400 GRAPHICS PAPER 1
410 TAG
420 MOVE 400,100
430 PRINT "My house";
440 TAGOFF
450 REM Hide the cursor
460 PEN 0
```

The TAG statement instructs the computer to write text at the graphics cursor. As line 420 moves the graphics cursor to point

400,100 line 430 displays:

```
My house
```

so that the top left hand corner of the M is at point 400,100.

Notice that when displaying text at the graphics cursor the PRINT statement must end with a semi-colon. The text is written in the graphics pen colour on the graphics background colour. Thus it is written in bright red on bright yellow. The TAGOFF statement at line 440 instructs the computer to display all future text at the text cursor in the text colour and with the text background colour.

Tutorial

8.14) Extend the program of Tutorial 8.12 so that the sentence:

```
This is an A.
```

is displayed at the foot of the screen in bright red.

Solutions to the tutorials

8.1) Change lines 70 and 80 of the program called ROBOT8a to:

```
70 REM Select yellow characters
80 PEN 9
```

Add the following lines:

```
135 REM Select pink characters
136 PEN 11
```

8.2)
```
10 REM Draw a picture in mode 0
20 MODE 0
30 REM Select bright green background
40 PAPER 12
50 REM Clear the screen to bright green
60 CLS
70 REM Select pink characters
80 PEN 11
90 REM Draw the robot
100 LOCATE 10,8
```

```
110 PRINT CHR$(136);CHR$(132)
120 LOCATE 10,9
130 PRINT CHR$(143);CHR$(143)
140 LOCATE 8,10
150 PRINT CHR$(132);CHR$(138);CHR$(143);CHR$(143);
CHR$(133);CHR$(136)
160 LOCATE 8,11
170 PRINT CHR$(133);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(138)
180 LOCATE 8,12
190 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(143)
200 LOCATE 9,13
210 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143)
220 LOCATE 9,14
230 PRINT CHR$(138);CHR$(143);CHR$(143);CHR$(133)
240 LOCATE 10,15
250 PRINT CHR$(135);CHR$(139)
260 LOCATE 10,16
270 PRINT CHR$(133);CHR$(138)
280 LOCATE 9,17
290 PRINT CHR$(143);CHR$(133);CHR$(138);CHR$(143)
```

8.3) Add the following lines to the program called ROBOT81:

```
255 REM Select black characters
256 PEN 5
```

8.4) Add the following lines to the program called ROBOT83:

```
300 LOCATE 7,20
310 PRINT "MY ROBOT"
```

8.5)

```
10 REM Draw a picture in mode 0
20 MODE 0
30 REM Select bright green background
40 PAPER 12
50 REM Clear the screen to bright green
60 CLS
70 REM Select pink characters
80 PEN 11
90 REM Draw the robot
100 LOCATE 7,8
```

```
110 PRINT CHR$(209);CHR$(143);" ";CHR$(136);CHR$(132)
120 LOCATE 7,9
130 PRINT CHR$(209);"  ";CHR$(143);CHR$(143)
140 LOCATE 7,10
150 PRINT CHR$(209);CHR$(132);CHR$(138);CHR$(143);
CHR$(143);CHR$(133)
160 LOCATE 8,11
170 PRINT CHR$(133);CHR$(143);CHR$(143);
CHR$(143);CHR$(143)
180 LOCATE 8,12
190 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(143)
200 LOCATE 9,13
210 PRINT CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(138)
220 LOCATE 9,14
230 PRINT CHR$(138);CHR$(143);CHR$(143);
CHR$(133);CHR$(130)
240 LOCATE 10,15
250 PRINT CHR$(135);CHR$(139)
260 LOCATE 10,16
270 PRINT CHR$(133);CHR$(138)
280 LOCATE 9,17
290 PRINT CHR$(143);CHR$(133);CHR$(138);CHR$(143)
```

8.6) Add the following lines to the program called ROBOT83:

```
300 REM Give the robot flashing buttons
310 REM Select transparent printing
320 PRINT CHR$(22)+CHR$(1)
330 REM Select flashing pink/sky blue characters and display his
buttons
340 PEN 15
350 LOCATE 9,17
360 PRINT CHR$(162)
370 LOCATE 12,17
380 PRINT CHR$(162)
390 REM Return to normal printing
400 PRINT CHR$(22)+CHR$(0)
```

8.7) Add the following lines to the program called BLACK-EYE:

```
364 REM Give the robot a nose
365 LOCATE 10,11
366 PRINT CHR$(194);CHR$(195)
```

8.8)

```
10 REM Tutorial 8.8
20 REM Clear the screen
30 CLS
40 REM Select transparent printing
50 PRINT CHR$(22)+CHR$(1)
60 REM Select black characters
70 PEN 5
80 LOCATE 10,10
90 PRINT CHR$(204)
100 REM Select bright red characters
110 PEN 3
120 LOCATE 10,10
130 PRINT CHR$(205)
140 REM Select bright yellow characters
150 PEN 1
160 LOCATE 10,10
170 PRINT CHR$(144)
180 REM Return to normal printing
190 PRINT CHR$(22)+CHR$(0)
```

8.9)

Name	Coordinates
B	350,350
C	175,250
D	300,100
E	50,25
F	500,200
G	600,300

8.10)

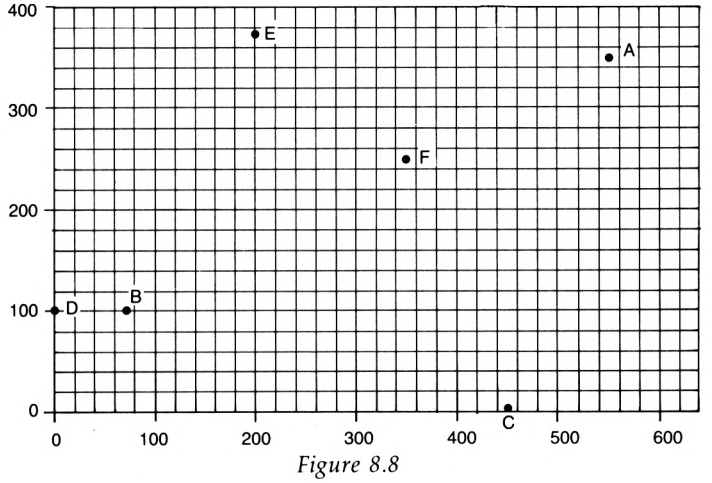


Figure 8.8

8.11)
```
10 REM Capital A
20 MODE 0
30 REM Select bright red on black for the graphics colours
40 GRAPHICS PAPER 5
50 GRAPHICS PEN 3
60 CLG
70 MOVE 100,100
80 DRAW 200,300
90 DRAW 300,100
100 DRAW 250,100
110 DRAW 230,160
120 DRAW 170,160
130 DRAW 150,100
140 DRAW 100,100
150 MOVE 200,250
160 DRAW 225,175
170 DRAW 175,175
180 DRAW 200,250
190 REM Hide the cursor
200 PEN 5
210 PAPER 5
```

8.12) Add the following lines to the program of Tutorial 8.11:

```
190 REM Fill the A in bright red
200 MOVE 200,275
210 FILL 3
220 REM Hide the cursor
230 PEN 5
240 PAPER 5
```

8.13) Add the following lines to the program of Example 8d:

```
101 REM Fill the walls in bright yellow
102 MOVE 200,100
103 FILL 1
171 REM Fill the roof in bright red
172 MOVE 200,225
173 FILL 3
231 REM Fill the door in bright red
232 MOVE 200,50
233 FILL 3
351 REM Fill the windows in bright cyan
352 MOVE 150,150
353 FILL 2
```

```
354 MOVE 250,150
355 FILL 2
```

8.14) Add the following lines to the program of Tutorial 8.12:

```
220 REM Write text on the picture
230 TAG
240 MOVE 50,80
250 PRINT "This is an A";
260 TAGOFF
270 REM Hide the cursor
280 PEN 5
290 PAPER 5
```

Using the computer's memory

The computer's memory can be thought of as a large honeycomb consisting of thousands of memory cells. Each of these memory cells can store numbers, letters, words or phrases.

Storing numbers in a memory cell

When a number is stored in a cell, the cell is given a label to distinguish it from the others, as shown in Fig. 9.1, where:

- the memory cell labelled `age` stores the number 6
- the memory cell labelled `daysinMay` stores the number 31
- the memory cell labelled `average` stores the number 100.

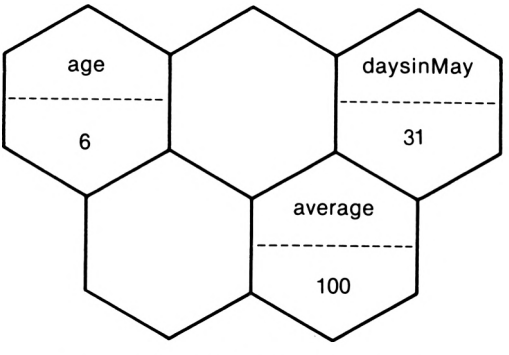


Figure 9.1 Labelled memory cells

When labelling a memory cell it is helpful to choose a label which is meaningful to the content of the cell. The choice of the label `age` for a memory cell would indicate that this cell stores somebody's age. If the ages of several people were to

be stored in memory cells then labels such as `Davidsage` or `Johnsage` could be used.

Although the labels given to the memory cells have been written in lower case letters except for proper nouns, BASIC cannot distinguish between lower and upper case letters. Hence the following labels are considered to be the same:

```
age   AGE   Age
```

We shall use lower case letters for labels as it distinguishes them from BASIC statements which are listed in upper case.

You will notice that one of the cells was labelled `daysinMay` and not `days in May` with spaces between the words. Labels must not contain spaces. However the full stop can be used instead of the spaces so that the label becomes `days.in.May` which is easier to read than `daysinMay`.

A label must start with a letter but can contain numbers if required. Labels `age1` and `age2` would be acceptable whereas `1age` and `2age` would not be allowed.

Labels must not contain punctuation marks other than the full stop. You should also avoid using symbols such as $ £ & +, etc. as these have a special meaning to the computer.

Your labels must not be the same as BASIC statements (e.g. the label `pen` would not be acceptable as it is the same as the BASIC statement `PEN`). Appendix 2 lists all the BASIC statements which will enable you to check if your labels are allowed.

Tutorial

9.1) Which of the following labels would be acceptable?

a)	`number?`	`oldage£`	`Tom's.age`
b)	`Johns age`	`Johns.age`	`Johnsage`
c)	`3rdletter`	`thirdletter`	`letter3`
d)	`number`	`newnumber`	`new.number`
e)	`time`	`minutes`	`stoppage.time`

The age of a boy called Tom, who is 21, can be stored in a memory cell labelled `Toms.age` by typing:

```
Toms.age = 21
```

Press an **ENTER** key. The computer now stores the number 21 in a memory cell labelled `Toms.age`.

To display the contents of the memory cell labelled `Toms.age` on the screen type:

```
PRINT Toms.age
```

Press an **ENTER** key. The computer now searches for the memory cell labelled `Toms.age` and displays a copy of its contents, 21, on the screen. The memory cell still contains the number 21 as only a copy of its contents was displayed on the screen.

Notice that the statement:

```
PRINT Toms.age
```

did not have quotation marks around the label `Toms.age` as the computer was being instructed to display the contents of the memory cell labelled `Toms.age`. If you had typed

```
PRINT "Toms.age"
```

the computer would have displayed:

```
Toms.age
```

which is not what was wanted.

When a memory cell stores a number its label is called a *numeric variable*. The word variable is chosen as the number stored in the memory cell can be altered. At present the memory cell labelled `Toms.age` contains the number 21. Type:

```
Toms.age = 30
```

Press an **ENTER** key. As the computer has already labelled a memory cell `Toms.age` it stores the number 30 in it and forgets the number that was already there.

Example 9a

Write a program which puts the number 9 into a memory cell labelled `number` and then displays its contents on the screen.

A possible program could be:

```
10 REM Example 9.a
20 number = 9
```

```
30 PRINT number
```

Tutorial

9.2) Write a program which puts the number 25 into a memory cell labelled `number` and then displays its contents on the screen.

9.3) Write a program which puts your age into a memory cell labelled `age` and then displays its contents on the screen.

Displaying text with the numbers on the screen

Enter and **RUN** the program of Example 9a. You will notice that the 9 is displayed in the second column of the screen. Add the following line to the program:

```
5 PAPER 3
```

RUN the program. The computer again displays the 9 in the second column of the screen but it has also included a space on both sides of the 9. This shows that when your Amstrad CPC 664 displays the contents of a numeric variable it includes a space on both sides of it.

You will seldom want to display a number on its own but may want to include an explanation, e.g. in Example 9a you would probably want to display:

```
The memory cell contains 9.
```

To include this message the program of Example 9a could become:

```
10 REM Example 9.a
20 number = 9
30 PRINT "The memory cell contains";number
```

Enter and **RUN** the program. At line 30 the computer displays the phrase within the quotation marks followed by the contents of the cell labelled `number` with a space on both sides of it. Hence it displays:

```
The memory cell contains 9
```

Example 9b

Change the program of Tutorial 9.3 to display on the screen:

```
I am ... years old.
```

A possible program could be:

```
10 REM Displays your age
20 age = 32
30 PRINT "I am";age;"years old."
```

Tutorial

9.4) Write a program to store the temperature in a memory cell labelled `degreesC` and then display on the screen:

```
The temperature is ... Centigrade.
```

9.5) Write a program to store the number of marathon competitors in a memory cell labelled `entrants` and then display on the screen:

```
... people entered the marathon.
```

Storing words or phrases in a memory cell

So far only numbers have been stored in the memory cells. Sometimes you may wish to store a word or phrase such as your name or address. This type of information is called a *string*. When a string is stored in a memory cell its label is called a *string variable*. The rules for labelling a string variable are the same as for a numeric variable except that they must end with a $ sign as in Table 9.1.

Cell label	Contents of the cell
name$	John
dog$	spaniel
meal$	lunch
place$	London

Table 9.1

A string can consist of a mixture of numbers, letters or

symbols provided that its label is a string variable, as in Table 9.2.

Cell label	Contents of the cell
date$	1st June 1984
birthday$	1–06–84

Table 9.2

The string consisting of the word `red` is written as the string `"red"`. As quotation marks are used to enclose the contents of the string they cannot be part of it. Hence a string could not consist of the sentence:

```
"I do not know" he said.
```

Tutorial

9.6) Which of the following statements could be true for the Amstrad CPC 664 computer?:

a) The memory cell labelled `Toms.age` contains the number 21.
b) The memory cell labelled `Toms.age$` contains the string `"twenty one"`.
c) The memory cell labelled `seconds` contains the string `"thirty five"`.
d) The memory cell labelled `Toms.age$` contains the string `"21st year"`.
e) The memory cell labelled `address$` contains the string `"12 Holm Brae"`.
f) The memory cell labelled `age` contains the number 13.

9.7) Suggest suitable labels for memory cells that contain the following information:

green	Edinburgh	25th December
6am	9 o'clock	

9.8) Match the cell contents to a suitable label in the following:

Cell label	Cell contents
type.of.bird$	London bridge
date$	47
age	Kenneth
age$	10 years old
time$	9–9–90
place$	4.45pm
year	canary
name$	1984

The string `"red"` can be stored in a memory cell labelled `colour$` by typing:

```
colour$ = "red"
```

Press an **ENTER** key. The computer has been instructed to store the characters within the quotation marks (i.e. `red`) in a memory cell labelled `colour$` (Fig. 9.2). Remember that the quotation marks are not part of the string.

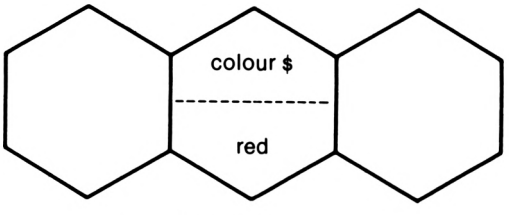


Figure 9.2 Storing a string

The contents of the memory cell labelled `colour$` can be displayed on the screen by typing:

```
PRINT colour$
```

Press an **ENTER** key. The computer displays the contents of the memory cell labelled `colour$` on the screen but, unlike a numeric variable, it does not insert a space at both ends of the string. Hence the computer displays on the screen:

```
red
```

Example 9c

Write a program that puts today's date into a memory cell and then displays its contents on the screen.

A possible program could be:

```
10 REM Today's date
20 date$ = "1.01.99"
30 PRINT date$
```

Enter and RUN the program. Line 20 instructs the computer to store the string "1.01.99" in the memory cell labelled date$. At line 30 the contents of this cell are displayed on the screen.

Change line 20 to:

```
20 date$ = "1st January 1999"
```

RUN the program. This time the computer displays on the screen:

```
1st January 1999
```

Tutorial

9.9) Write a program to store your telephone number in a memory cell and then display its contents on the screen.

9.10) Write a program to store the name of your street in a memory cell and then display its contents on the screen.

Displaying text with strings

As with numbers you will probably want to include some explanation when displaying the contents of a memory cell on the screen. For example the program of Example 9c could be extended so that the computer displays:

```
Today is 1st January 1999
```

This would require the PRINT statement at line 30 in the program of Example 9c to be extended so that the computer displays on the same line both the phrase Today is and the contents of the memory cell labelled date$. You might expect line 30 to become:

```
30 PRINT "Today is";date$
```

Modify line 30 and RUN the program. The computer displays:

```
Today is1st January 1999
```

The computer has written the contents of the cell `date$` immediately after `Today is`. The missing space on the screen can be obtained by including a space in line 30 between the `is` and the quotation mark:

```
30 PRINT "Today is ";date$
```

Modify line 30 and `RUN` the program again. The desired display is now obtained.

Example 9d

Write a program that stores your name in a memory cell and then displays on the screen:

```
I am called .....
```

A possible program could be:

```
10 REM Displaying your name on the screen
20 name$ = "Jack"
30 PRINT "I am called ";name$;"."
```

Tutorial

9.11) Extend Tutorial 9.9 so that the computer displays on the screen:

```
My telephone number is ...
```

9.12) Extend Tutorial 9.10 so that the computer displays on the screen:

```
I live in ...
```

Sometimes more than one variable will be required in a program as demonstrated in the following example.

Example 9e

Write a program to store your name, your birthplace, your date of birth and your age in separate memory cells and then display the contents of these cells on the screen.

A possible program could be:

```
10 REM Stores and displays your name , birthplace, date of birth
```

```
and age
20 REM Clear the screen
30 CLS
40 name$ = "John Smith"
50 place$ = "Timbuctoo"
60 date$ = "2nd June 1968"
70 age = 14
80 REM Display the information
90 LOCATE 8,8
100 PRINT "My name is ";name$
110 LOCATE 8,10
120 PRINT "and I was born on ";date$
130 LOCATE 8,12
140 PRINT "at ";place$;"."
150 LOCATE 8,14
160 PRINT "I am now";age;"years old."
```

Tutorial

9.13) Write a program to store the title, the author and the publisher of a book in separate memory cells. It should then display in mode 1 the contents of these cells in bright yellow, bright cyan and bright red respectively on lines 8,10 and 12 of the screen.

9.14) Write a program to store the following information about a car sale in memory cells and then display in mode 1 the contents of these memory cells suitably spaced on the screen.

Month	January
Make of car	Ford
Model	Escort estate
Colour	Blue
Selling price	£4500

Solutions to the tutorials

9.1) *a*) `number?` and `Tom's.age` are not acceptable as they contain the punctuation marks ? and '. `oldage£` is not acceptable as it contains the symbol £.

b) `Johns age` is not acceptable as it has a space in it.

c) `3rdletter` is not acceptable as it starts with a number.
d) All acceptable.
e) `time` is not acceptable as it is the same as the BASIC statement `TIME`.

9.2)
```
10 REM Tutorial 9.2
20 number = 25
30 PRINT number
```

9.3)
```
10 REM Tutorial 9.3
20 age = 28
30 PRINT age
```

9.4)
```
10 REM Displays the temperature
20 degreesC = 24
30 PRINT "The temperature is";degreesC;"Centigrade."
```

9.5)
```
10 REM Number of marathon competitors
20 entrants = 2567
30 PRINT entrants;"people entered the marathon."
```

9.6) Statements *a b d e* and *f* could be true.
Statement *c* is wrong as the memory cell labelled `seconds` must contain a number and not a string.

9.7)

Cell label	Cell contents
colour$	green
town$	Edinburgh
date$	25th December
hour$	6am
arrival.time$	9 o'clock

9.8)

Cell label	Cell contents
type.of.bird$	canary
date$	9–9–90
age	47
age$	10 years old
time$	4.45pm
place$	London bridge
year	1984
name$	Kenneth

9.9)
```
10 REM Telephone number
20 phone.number$ = "0532-636311"
```

```
30 PRINT phone.number$
```

9.10)
```
10 REM Name of street
20 street$ = "Fletcher Grove"
30 PRINT street$
```

9.11)
```
10 REM Telephone number
20 phone.number$ = "0532-636311"
30 PRINT "My telephone number is ";phone.number$
```

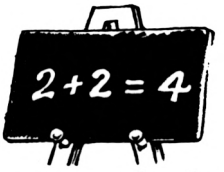
9.12)
```
10 REM Name of street
20 street$ = "Fletcher Grove"
30 PRINT "I live in ";street$;"."
```

9.13)
```
10 REM Details of a book
20 title$ = "Advancing with the Electron"
30 author$ = "Peter Seal"
40 publisher$ = "Micro Press"
50 REM Clear the screen
60 CLS
70 REM Display title in bright yellow
80 PEN 1
90 LOCATE 6,8
100 PRINT title$
110 REM Display author in bright cyan
120 PEN 2
130 LOCATE 12,10
140 PRINT "by ";author$
150 REM Display publisher in bright red
160 PEN 3
170 LOCATE 12,12
180 PRINT "published by ";publisher$
190 REM Hide the cursor
200 PEN 0
```

9.14)
```
10 REM Information about a car
20 month$ = "January"
30 make$ = "Ford"
40 model$ = "Escort estate"
50 colour$ = "Blue"
60 price$ = "£4500"
70 REM Clear the screen
80 CLS
90 REM Display the information
```

```
100 LOCATE 5,6
110 PRINT "Month          ";month$
120 LOCATE 5,8
130 PRINT "Make of car    ";make$
140 LOCATE 5,10
150 PRINT "Model          ";model$
160 LOCATE 5,12
170 PRINT "Colour         ";colour$
180 LOCATE 5,14
190 PRINT "Selling price ";price$
200 REM Hide the cursor
210 PEN 0
```

Sums with the Amstrad CPC 664



Using the computer as a calculator

Type (noticing the space between PRINT and 5+4):

```
PRINT 5+4
```

Press an **ENTER** key. The computer calculates 5+4 and displays the answer 9 on the screen.

Type:

```
PRINT 5-4
```

Press an **ENTER** key. The computer calculates 5−4 and displays the answer 1 on the screen.

Remembering that * means multiply, type:

```
PRINT 5*4
```

Press an **ENTER** key. The computer calculates five times four and displays the answer 20 on the screen.

Remembering that / means divide, type:

```
PRINT 5/4
```

Press an **ENTER** key. The computer calculates five divided by four and displays the answer 1.25 on the screen.

Tutorial

10.1) Use the computer to work out the following sums:

a) 79−5.6
b) 123.34+145.98
c) 1.34*2.75

In the above examples the computer has displayed only the answer to the calculation on the next line. Sometimes it is more convenient to display the problem and the answer as

```
5+4 = 9
```

This can be achieved by typing:

```
PRINT "5+4 =";5+4
```

Press an **ENTER** key. The part of the PRINT statement in quotation marks is displayed on the screen. Because of the semi-colon the result of the calculation 5+4 is also displayed on the same line giving:

```
5+4 = 9
```

Example 10a

Using a PRINT statement ask the computer to calculate 25 times 33 and display the answer as:

```
25*33 = ...
```

Type:

```
PRINT "25*33 =";25*33
```

Press an **ENTER** key. The computer then displays on the screen:

```
25*33 = 825
```

Tutorial

10.2) Using the PRINT statement ask the computer to calculate 6/4 and display the answer as:

```
6/4 = ...
```

10.3) Using the PRINT statement ask the computer to calculate 48+81+93 and display the answer as:

```
48+81+93 = ...
```

10.4) Using the PRINT statement ask the computer to calculate 245*73 and display the answer as:

```
2:*73 = ...
```

10.5) Using the PRINT statement ask the computer to calculate 36−12 and display the answer as:

```
36-12 = ...
```

Remembering the answer to a calculation

The answer to a calculation does not have to be displayed on the screen, it can be stored in a memory cell. Type:

```
result = 5+4
```

Press an **ENTER** key. The computer now adds 5 and 4 to get 9. The 9 is stored in a memory cell labelled result. The contents of this cell can be displayed on the screen by typing:

```
PRINT result
```

Press an **ENTER** key. The computer now displays the contents of the cell result on the screen.

The above two statements can be included in a program:

```
10 REM Displays the sum of 5 and 4
20 result = 5+4
30 PRINT result
```

Enter and RUN the program. The computer displays a 9 on the screen. One of its memory cells will be labelled result and contain the number 9.

The contents of memory cells containing numbers can also be used in arithmetic operations. Consider the program?

```
10 REM Addition of the contents of two memory cells
20 first.number = 70.2
30 second.number = 34.6
40 result = first.number + second.number
40 PRINT first.number;"+";second.number;"=";result
```

Enter and RUN the program. The computer displays:

```
70.2 + 34.6 = 104.8
```

Three of the computer's memory cells are now labelled as in Fig. 10.1.

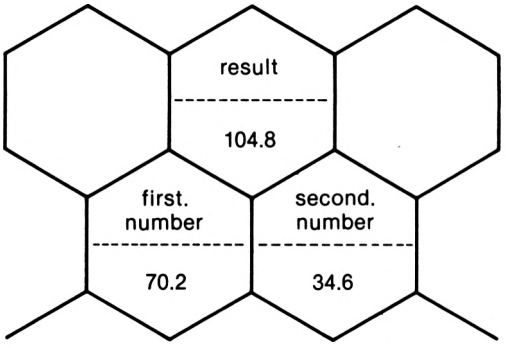


Figure 10.1 Storing two numbers and their sum

Example 10b

Write a program to calculate the fine on an overdue library book. Assume that the book is 19 days overdue and fines are charged at 2p per day.

A possible program could be:

```
10 REM Calculates the fine on an overdue book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
40 days.overdue = 19
50 fine = days.overdue * charge.per.day
60 PRINT "The amount to be paid is";fine;"pence."
```

Example 10c

Write a program to display the cost of three tins of beans if one tin costs 15 pence.

A possible program could be:

```
10 REM Displays cost of three tins of beans
20 cost.of.beans = 15
30 three.bean = 3* cost.of.beans
40 PRINT "Cost of 3 tins of beans is";three.bean;"pence."
```

Tutorial

10.6) An average of 625 people attended a flower show each day. Write a program to display on the screen the total number of people attending the show if it lasted for

seven days.

10.7) An information office is open for three hours each day. Write a program to calculate the total number of hours that it is open in June.

10.8) A school has 262 boys and 200 girls. Write a program to calculate the average number of pupils in each class assuming that there are 14 classes.

10.9) Write a program to calculate the monthly repayments on a loan of £200 for nine months at zero interest. The computer should display on the screen:

```
The monthly repayment is .... pounds.
```

The ROUND statement

You will have noticed that the computer displayed the answer to Tutorial 10.9 as:

```
The monthly repayment is 22.2222222 pounds.
```

Normally this would be quoted to the nearest penny as 22.22 pounds. As only two numbers are retained after the decimal point it is said to be rounded off to two decimal places. The computer can be instructed to ROUND the payment to two decimal places by including the instruction:

```
45 real.payment = ROUND(payment,2)
```

The 2 indicates that the contents of the memory cell labelled *payment* is to be rounded to two decimal places. The result is stored in the memory cell labelled real.payment.

The program becomes:

```
10 REM Calculate the repayments at zero interest rate
20 capital = 200
30 month = 9
40 payment = capital / month
45 real.payment = ROUND(payment,2)
50 PRINT "The monthly payment is"
60 PRINT real.payment;"pounds."
```

Example 10d

Write a program to display the yearly interest on £226.35 at a rate of 8.5% per annum.

A possible program could be:

```
10 REM Yearly interest on £226.35 at 8.5% per annum
20 interest.rate = 8.5/100
30 capital = 226.35
40 interest = capital * interest.rate
50 real.interest = ROUND(interest,2)
60 PRINT "Yearly interest on £226.35 at 8.5% is"
70 PRINT real.interest;"pounds."
```

Enter and RUN the program. The computer displays on the screen:

```
Yearly interest on £226.35 at 8.5% is
19.24 pounds.
```

Tutorial

10.10) If VAT is charged at 15%, write a program to calculate the VAT payable on goods costing £22.60 and display at the centre of the screen:

```
VAT on £22.60 is .... pounds.
```

10.11) Write a program to calculate the twelve monthly repayments on a loan. Assume that the loan is for £2000 and the interest rate is 8.25% per annum payable on the total sum borrowed. Display the result at the centre of a mode 1 screen in bright red on a bright yellow background with the rest of the screen blue.

10.12) Write a program to display in mode 1 the cost of five litres of petrol at 42.8 pence a litre. The cost should be in bright red and any text in bright yellow on a blue background.

Displaying the pound sign

In Tutorial 10.10 the computer displayed:

```
The VAT on £22.60 is 3.39 pounds.
```

You might prefer the computer to display:

```
The VAT on £22.60 is £3.39
```

If the computer displayed the £ sign on the screen and then the number 3.39, the £ sign would be overwritten by the space that the computer places in front of a number. Hence the computer must display:

```
The VAT on £22.60 is  3.39
```

It can then display the £ sign at the appropriate position. Notice the two spaces between the word `is` and `3.39`. The computer will display one of them along with the 3.39 but the other must be included in the `PRINT` statement. The program would become:

```
10 REM Calculate VAT charge
20 cost = 22.60
30 VAT.rate = 15/100
40 VAT = cost * VAT.rate
50 real.VAT = ROUND(VAT,2)
60 REM Clear the screen
70 CLS
80 LOCATE 3,12
90 PRINT "The VAT on £22.60 is ";real.VAT
100 LOCATE 24,12
110 PRINT "£"
```

Enter and `RUN` the program. The computer displays:

```
The VAT on £22.60 is £3.39
```

Tutorial

10.13) Extend the program of Tutorial 10.11 so that the computer displays:

```
The monthly payment is £...
```

Random numbers

The statement:

```
INT(RND*4)+1
```

will generate a random whole number from 1 to 4, i.e. it selects one of the following numbers 1,2,3,4. The term random means that if the statement is repeated a large number of times then each of the four numbers will be selected equally often. The random number selected can either be displayed on the screen using the **PRINT** statement or stored in a memory.
Type:

```
PRINT INT(RND*4)+1
```

Press an **ENTER** key. The computer now selects one of the numbers 1,2,3 or 4 and displays it on the screen. Repeat this several times and you will see the various numbers being selected.

Example 10e

Write a program to act as a die.
A possible program could be:

```
10 REM Program to simulate a die.
20 CLS
30 random.number = INT(RND*6)+1
40 PRINT "The die shows";random.number
```

SAVE this program onto a disk and call it **DIE**.

Tutorial

10.14) Write a program in mode 0 to simulate the throw of two dice. It should then display the result at the centre of the screen in black on a bright white background with the rest of the screen blue.

The RANDOMIZE statement

Repeat the following sequence several times:

1) Switch off the computer.
2) Switch the computer on.
3) **LOAD** the program **DIE** from disk.

4) RUN it 10 times and note the random number generated.

You will find that when the program of Example 10e is entered and RUN immediately after the computer is switched on then it always generates the same sequence of random numbers. Different sequences can be obtained by including at the beginning of the program the statement:

```
RANDOMIZE TIME
```

The program of Example 10e would become:

```
10 REM Program to simulate a die
20 REM Vary the sequence of random numbers
30 RANDOMIZE TIME
40 CLS
50 random.number = INT(RND*6)+1
60 PRINT "The die shows";random.number
```

Finding the faults in a program using the STOP and CONT (continue) statements

If your program does not produce the desired results then there must be a mistake in your typing, your logic or both. Let's assume that in the program of Tutorial 10.10 you typed line 40 as

```
40 VAT = cost * VATrate
```

and did not notice the missing full stop in VATrate. Enter the program and RUN it. The computer displays:

```
The VAT rate on £22.60 is 0 pounds.
```

The STOP and CONT statements are useful in finding and removing the error. This is called *debugging* the program. Insert two STOP statements into the program at lines 35 and 85:

```
35 STOP

85 STOP
```

RUN the program. When the computer executes line 35 the STOP statement instructs it to stop executing the program. The computer displays:

```
Break in 35
Ready
```

At this stage the computer should have stored 22.60 and 0.15 in the memory cells labelled `cost` and `VAT.rate`. This can now be checked by instructing it to display the contents of these cells by typing:

```
PRINT cost
```

Press an **ENTER** key. The computer displays 22.60 followed by the usual `Ready` message. Now type:

```
PRINT VAT.rate
```

Press an **ENTER** key. The computer displays 0.15 followed by the usual `Ready` message. The results show that the program has functioned correctly up to line 35.

The computer can be told to continue executing the program by typing:

```
CONT
```

Press an **ENTER** key. When the computer reaches line 85 it again stops executing the program. As before you can instruct it to display the contents of the memory cells labelled `VAT` and `real.VAT`. You will find that both of these are 0 which suggests that there is a fault at the line where they were set up, i.e. at lines 40 and 50. You would probably notice the error at this stage, correct it, remove the `STOP` statements and `RUN` the program.

Solutions to the Tutorials

10.1) *a*) 73.4
b) 269.32
c) 3.685

10.2) `PRINT "6/4 =";6/4`

10.3) `PRINT "48+81+93 =";48+81+93`

10.4) `PRINT "245*73 =";245*73`

10.5) `PRINT "36-12 =";36-12`

10.6)
```
10 REM Attendance at a flower show
20 daily.attendance = 625
30 number.of.days = 7
40 total = daily.attendance * number.of.days
50 PRINT total;"people attended the flower show."
```

10.7)
```
10 REM Total hours of opening
20 daily.hours = 3
30 number.of.days = 30
40 total = daily.hours * number.of.days
50 PRINT "The information office is open for"
60 PRINT total;"hours in June."
```

10.8)
```
10 REM Average number of pupils per class
20 boys = 262
30 girls = 200
40 total = boys + girls
50 classes = 14
60 average = total / classes
70 PRINT "There are";average;"pupils per class."
```

10.9)
```
10 REM Calculate the repayments at zero interest rate
20 capital = 200
30 month = 9
40 payment = capital / month
50 PRINT "The monthly payment is"
60 PRINT payment;"pounds."
```

10.10)
```
10 REM Calculate VAT charge
20 cost = 22.60
30 VAT.rate = 15/100
40 VAT = cost * VAT.rate
50 real.VAT = ROUND(VAT,2)
60 REM Clear the screen
70 CLS
80 LOCATE 3,12
90 PRINT "The VAT on £22.60 is";real.VAT;"pounds."
```

10.11)
```
10 REM Calculate the repayments at 8.25% interest
20 capital = 2000
30 interest.rate = 8.25/100
40 interest = capital * interest.rate
50 total.repayment = capital + interest
```

```
60 monthly.repayment = total.repayment / 12
70 real.monthly.repayment = ROUND(monthly.repayment,2)
80 REM Clear the screen to blue
90 CLS
100 REM Select bright yellow background
110 PAPER 1
120 REM Select bright red characters
130 PEN 3
140 LOCATE 2,12
150 PRINT "The monthly payment is";real.monthly.repayment;"pound
s."
160 REM Hide the cursor
170 PAPER 0
180 PEN 0
```

10.12)

```
10 REM Cost of petrol
20 cost.litre = 42.8
30 number.of.litres = 5
40 cost = cost.litre * number.of.litres / 100
50 real.cost = ROUND(cost,2)
60 REM Clear the screen to blue
70 CLS
80 REM Select bright yellow characters
90 PEN 1
100 LOCATE 2,10
110 PRINT "The cost of five litres of petrol is"
120 REM Select bright red characters
130 PEN 3
140 LOCATE 13,12
150 PRINT real.cost;
160 REM Select bright yellow characters
170 PEN 1
180 PRINT "pounds."
190 REM Hide the cursor
200 PEN 0
210 PAPER 0
```

10.13)

```
10 REM Calculate the repayments at 8.25% interest
20 capital = 2000
30 interest.rate = 8.25/100
40 interest = capital * interest.rate
50 total.repayment = capital + interest
```

```
60 monthly.repayment = total.repayment / 12
70 real.monthly.repayment = ROUND(monthly.repayment,2)
80 REM Clear the screen to blue
90 CLS
100 REM Select bright yellow background
110 PAPER 1
120 REM Select bright red characters
130 PEN 3
140 LOCATE 2,12
150 PRINT "The monthly payment is ";real.monthly.repayment
160 REM Display the £ sign
170 LOCATE 27,12
180 PRINT "£"
190 REM Hide the cursor
200 PAPER 0
210 PEN 0
```

10.14)

```
10 REM Simulates two dice
20 dice1 = INT(RND*6)+1
30 dice2 = INT(RND*6)+1
40 REM Select mode 0
50 MODE 0
60 REM Select bright white background
70 PAPER 4
80 REM Select black characters
90 PEN 5
100 LOCATE 9,12
110 PRINT dice1;dice2
120 REM Hide the cursor
130 PEN 0
140 PAPER 0
```

You speak to the computer

In Chapter 10 you wrote the following program to calculate the fine on an overdue library book.

```
10 REM  Calculates the fine on an overdue book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
40 days.overdue = 19
50 fine = days.overdue * charge.per.day
60 PRINT "The amount to be paid is";fine;"pence."
```

Every time an overdue book was returned the librarian would have to:

1) List the program.
2) Change line 40 to the appropriate value.
3) Run the program to calculate the fine due.

It would obviously be more convenient if the program could ask the librarian to enter the number of overdue days while it was running. This is done by means of the INPUT statement. This statement instructs the computer to cease executing the program temporarily and wait until information is entered at the keyboard. When the computer receives this information it continues with the program. The information entered at the keyboard can either be a number or a string.

Use of the INPUT statement to enter a number

Type (noticing the space between INPUT and number):

```
INPUT number
```

Press an **ENTER** key. The computer displays on the screen:

```
? ▮
```

and then waits for you to type (i.e. INPUT) a number.

Type in 86 and you will see that it is also displayed on the screen. Press an **ENTER** key. This informs the computer that you have finished typing the number and that it can now continue. The number 86 is stored in the memory cell labelled number. The computer then displays the usual Ready message showing that it is waiting for another command. You can verify this by typing:

```
PRINT number
```

Press an **ENTER** key. The computer now displays 86 on the screen.

The INPUT statement can be used as part of a program as follows:

```
10 REM Send a number to the computer
20 INPUT number
30 PRINT "You typed in";number
```

Enter and RUN the program. The computer waits at line 20 until you type in a number (say 21) and press an **ENTER** key. It then proceeds to line 30 where it displays:

```
You typed in 21
```

Example 11a

Write a program to INPUT the cost of a tin of beans and display:

```
The cost of a tin of beans is .. pence.
```

The program could be:

```
10 REM Cost of beans
20 INPUT price
30 PRINT "The cost of a tin of beans is";price;"pence."
```

Tutorial

11.1) Write a program to INPUT a number into a memory

cell labelled choice, clear the screen and display:

```
The number you picked was ..
```

11.2) Write a program to INPUT the cost of a tin of beans into a memory cell labelled price, clear the screen and display:

```
The cost of 3 tins of beans is ..pence.
```

11.3) Write a program to INPUT the number of days in February into a memory cell labelled number, clear the screen and display:

```
February has .. days this year.
```

Let's rewrite the program to calculate the fine on an overdue library book, but this time include an INPUT statement at line 40.

```
10 REM  Calculates the fine on an overdue book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
40 INPUT days.overdue
50 fine = days.overdue * charge.per.day
60 PRINT "The amount to be paid is";fine;"pence."
```

Enter and RUN the program. The computer prompts you to enter a number from the keyboard at line 40 by displaying on the screen:

```
? 
```

Unfortunately it gives you no information about the number it requires. This can be overcome by inserting a PRINT statement immediately before the INPUT statement.

The program now becomes:

```
10 REM  Calculates the fine on an overdue book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
35 PRINT"How many days is the book overdue?"
40 INPUT days.overdue
50 fine = days.overdue * charge.per.day
60 PRINT "The amount to be paid is";fine;"pence."
```

Add line 35 to your program and RUN it. The screen now

displays:

```
How many days is the book overdue?
? ■
```

The computer now waits for you to type (i.e. INPUT) the number of days (say 5) and press an **ENTER** key. The computer now displays on the screen:

```
The amount to be paid is 10 pence.
```

Example 11b

Write a program that asks your age, clears the screen and displays:

```
You are .. years old.
```

The program could be:

```
10 REM This program asks your age
20 PRINT "What is your age?"
30 INPUT age
40 REM Clear the screen
50 CLS
60 PRINT "You are";age;"years old."
```

Tutorial

11.4) Write a program that asks how many people attended a car show, clears the screen and displays:

```
.. people attended the car show.
```

11.5) Write a program that asks how many litres of petrol at 42.8 pence per litre you wish to purchase. It should then clear the screen in mode 1 and display at the centre of the screen in bright red on blue:

```
You have asked for .. litres.
The cost is £..
```

11.6) Expand Example 11b so that the screen displays:

```
You are .. years old and in 12 years time you will be ..
```

Use of the INPUT statement to enter a string

It is often necessary to INPUT a word or a phrase (e.g. your name or address) into a program. You will remember from Chapter 9 that words or phrases are strings and that the label given to the memory cell that stores a string must end with a $, e.g. *name$, address$*.

The INPUT statement for a string behaves the same as for a number. Type:

```
INPUT name$
```

Press an **ENTER** key. The computer displays on the screen:

```
? 
```

and then waits for you to type (i.e. INPUT) a string. Type in your name (say John but do not put quotation marks around it). You will see that it is also displayed on the screen. Press an **ENTER** key. This informs the computer that you have finished typing the string and that it can now continue.

The string "John" is stored in the memory cell labelled name$. The computer then displays the usual Ready message showing that it is waiting for another instruction. You can verify this by typing:

```
PRINT name$
```

Press an **ENTER** key. The computer now displays on the screen:

```
John
```

The INPUT statement for a string can also be used in a program as demonstrated by the following example.

Example 11c

Write a program that asks your name and then displays on the screen:

```
Hello ... I am the Amstrad CPC 664.
```

A possible program could be:

```
10 REM Asks your name
```

```
20 PRINT "What is your name?"
30 INPUT name$
40 PRINT "Hello "¡name$¡" I am the Amstrad CPC 664."
```

Tutorial

11.7) Write a program that asks which day of the week it is, clears the screen and displays:

```
Today is ...
```

11.8) Write a program that clears the screen, asks your favourite colour and then displays:

```
Your favourite colour is ...
```

11.9) Write a program that asks what you would like for lunch, clears the screen and displays:

```
... is served.
```

More than one INPUT statement can be used in a program as demonstrated in the following example.

Example 11d

Write a program that clears the screen, asks your name and age and then displays:

```
Hello ... You are .. years old.
```

A possible program could be:

```
10 REM Asks your name and age
20 REM Clear the screen
30 CLS
40 PRINT "What is your name?"
50 INPUT name$
60 PRINT "What is your age?"
70 INPUT age
80 PRINT "Hello "¡name$¡". You are"¡age¡"years old."
```

Tutorial

11.10) Write a program to ask which grade of petrol you wish to buy followed by the number of litres you require. It should then display centred on the screen:

```
You have selected grade .. and have asked for .. litres.
```

11.11) Write a program to ask your name, town and telephone number. It should then display centred on the screen:

```
Your name is ...
You live in ...
Your telephone number is ...
```

11.12) Write a program to ask the day of the week, the date, the month and the year. It should then display centred on the screen:

```
Today is ... ... ... ...
```

Combining the PRINT and INPUT statements

Let's look back at lines 40 and 50 of Example 11d:

```
40 PRINT "What is your name?"
50 INPUT name$
```

These lines gave the following screen display:

```
What is your name?
? 
```

The name typed in (say John) is displayed at the cursor so that the screen now displays:

```
What is your name?
? John 
```

Lines 40 and 50 can be combined into one statement namely:

```
40 INPUT "What is your name?",name$
```

This gives the following screen display:

```
What is your name? 
```

When the name John is typed it is displayed at the cursor so the screen now displays:

```
What is your name?John 
```

When an **ENTER** key is pressed the computer knows that you have finished typing the string and that it can continue. It

stores the string "John" in the memory cell labelled name$ and then proceeds to the next line of the program.

If you require a space between the ? and John on the screen display, line 40 should be:

```
40 INPUT "What is your name? ",name$
```

Notice the extra space between the ? and the ".

In a similar way lines 60 and 70 of Example 11d can be changed from:

```
60 PRINT "What is your age?"
70 INPUT age
```

to

```
60 INPUT "What is your age? ",age
```

The program in Example 11d would now be:

```
10 REM Asks your name and age
20 REM Clear the screen
30 CLS
40 INPUT "What is your name? ",name$
60 INPUT "What is your age? ",age
70 PRINT "Hello ";name$;". You are";age;"years old."
```

The LOCATE statement can be used with the INPUT statement allowing you to plan your screen displays. Assume that you wish to extend the above program so that the INPUT questions are displayed at screen positions 6,10 and 8.12. The computer should then clear the screen and display at screen position 1,10:

```
Hello ... You are .. years old.
```

The program would become:

```
10 REM Asks your name and age
20 REM Clear the screen
30 CLS
40 REM INPUT the information
50 LOCATE 6,10
60 INPUT "What is your name? ",name$
70 LOCATE 8,12
80 INPUT "What is your age? ",age
90 REM Clear the screen
```

```
100 CLS
110 REM Display the information
120 LOCATE 1,10
130 PRINT "Hello ";name$;". You are";age;"years old."
```

Tutorial

11.13) Extend the library book program so that the PRINT and INPUT statements are combined.

11.14) Write a program using a good screen display that combines the PRINT and INPUT statements in Tutorial 11.11 into one INPUT statement and then displays in mode 1:

```
Your name is ....
You live in ...
Your telephone number is ...
```

These lines should be displayed in bright yellow, bright red and bright cyan respectively on a blue background.

Solutions to the tutorials

11.1)
```
10 REM Inputs a number
20 INPUT choice
30 REM Clear the screen
40 CLS
50 PRINT "The number you picked was";choice
```

11.2)
```
10 REM Cost of a tin of beans
20 INPUT price
30 three.bean = 3* price
40 REM Clear the screen
50 CLS
60 PRINT "The cost of 3 tins of beans
is";three.bean;"pence."
```

11.3)
```
10 REM Number of days in February
20 INPUT number
30 REM Clear the screen
40 CLS
50 PRINT "February has";number;"days this year."
```

11.4)
```
10 REM Numbers attending a car show
20 PRINT "How many people attended the car show today?"
30 INPUT attendance
40 REM Clear the screen
50 CLS
60 PRINT attendance;"people attended the car show."
```

11.5)
```
10 REM Purchase of petrol
20 REM Select mode 1
30 MODE 1
40 REM INPUT number of litres of petrol
50 PRINT "How many litres of petrol at 42.8 pence"
60 PRINT "per litre do you wish to purchase?"
70 INPUT quantity
80 REM Clear the screen
90 CLS
100 REM Select bright red characters
110 PEN 3
120 REM Calculate and display the cost
130 cost.per.litre = 42.8
140 cost.pounds = cost.per.litre * quantity / 100
150 real.cost = ROUND(cost.pounds,2)
160 LOCATE 10,10
170 PRINT "You have asked for";quantity;"litres."
180 LOCATE 11,12
190 PRINT "The cost is ";real.cost
200 REM Insert the £ sign
210 LOCATE 23,12
220 PRINT "£"
230 REM Hide the cursor
240 PEN 0
```

11.6)
```
10 REM This program asks your age
20 PRINT "What is your age?"
30 INPUT age
40 later.age = age +12
50 REM Clear the screen
60 CLS
70 PRINT "You are";age;"years old and in 12 years"
80 PRINT "time you will be";later.age
90 REM Hide the cursor
100 PEN 0
```

11.7)
```
10 REM Day of the week
20 PRINT "Which day is it?"
30 INPUT day$
40 REM Clear the screen
50 CLS
60 PRINT "Today is ";day$
```

11.8)
```
10 REM Your favourite colour
20 REM Clear the screen
30 CLS
40 PRINT "What is your favourite colour?"
50 INPUT colour$
60 PRINT "Your favourite colour is ";colour$;"."
```

11.9)
```
10 REM Lunch menu
20 PRINT "What would you like for lunch?"
30 INPUT food$
40 REM Clear the screen
50 CLS
60 PRINT food$;" is served."
```

11.10)
```
10 REM Purchase of petrol
20 REM INPUT the information
30 PRINT "Which grade of petrol do you wish?"
40 INPUT star
50 PRINT "How many litres of petrol do you wish"
60 PRINT "to purchase?"
70 INPUT quantity
80 REM Clear the screen
90 CLS
100 REM Display the information
110 LOCATE 6,8
120 PRINT "You have selected grade";star
130 LOCATE 5,10
140 PRINT "and have asked for";quantity;"litres."
150 REM Hide the cursor
160 PEN 0
```

11.11)
```
10 REM Address and telephone number
20 REM INPUT the information
30 PRINT "What is your name?"
40 INPUT name$
50 PRINT "Which town do you live in?"
```

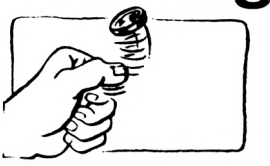
```
60 INPUT town$
70 PRINT "What is your telephone number?"
80 INPUT telephone$
90 REM Clear the screen
100 CLS
110 REM Display the information
120 LOCATE 4,8
130 PRINT "Your name is ";name$
140 LOCATE 4,10
150 PRINT "You live in ";town$
160 LOCATE 1,12
170 PRINT "Your telephone number is ";telephone$
180 REM Hide the cursor
190 PEN 0
```

11.12)
```
10 REM Asks the day and date
20 REM INPUT the information
30 PRINT "What day is it?"
40 INPUT day$
50 PRINT "What is the date?"
60 INPUT date$
70 PRINT "What month is it?"
80 INPUT month$
90 PRINT "What year is it?"
100 INPUT year
110 REM Clear the screen
120 CLS
130 REM Display the information
140 LOCATE 1,12
150 PRINT "Today is ";day$;" ";date$;" ";month$;year
160 REM Hide the cursor
170 PEN 0
```

11.13)
```
10 REM Calculates the fine on an overdue book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
40 REM INPUT the days overdue
50 INPUT "How many days are the book overdue? ",
days.overdue
60 fine = days.overdue * charge.per.day
70 REM Display the fine
80 PRINT "The amount to be paid is";fine;"pence."
```

11.14)
```
10 REM Address and telephone number
20 REM Clear the screen
30 CLS
40 REM INPUT the information
50 LOCATE 4,8
60 INPUT "What is your name? ",name$
70 LOCATE 4,10
80 INPUT "Which town do you live in? ",town$
90 LOCATE 1,12
100 INPUT "What is your telephone number? ",telephone$
100 REM Clear the screen
110 CLS
120 REM Display the information
130 LOCATE 4,8
140 PRINT "Your name is ";name$
150 REM Select bright red characters
160 PEN 3
170 LOCATE 4,10
180 PRINT "You live in ";town$
190 REM Select bright cyan characters
200 PEN 2
210 LOCATE 1,12
220 PRINT "Your telephone number is ";telephone$
230 REM Hide the cursor
240 PEN 0
```

Decision making



The IF...THEN...ELSE statement

Think about the following everyday situations.

- IF it is sunny THEN we shall go to the beach ELSE we shall go to the zoo.
- IF it is 10 o'clock THEN you must go to bed ELSE you may watch the television.
- IF anyone rings the bell THEN go to the door ELSE continue reading your book.
- IF the letter is addressed to you THEN open it ELSE leave it for me.

Each of these situations involves a decision of the following type **IF** some condition is true **THEN** take the appropriate action **ELSE** take some other action. This is called an IF... THEN...ELSE statement and can be used to instruct the computer to make decisions similar to those above.

Use of the IF...THEN...ELSE statement to compare numbers

Assume that you wish to instruct the computer to make the following decision: if the contents of the memory cells labelled `first.number` and `second.number` are equal then display on the screen:

```
The contents of the two memory cells are equal.
```

otherwise display on the screen:

```
The contents of the two memory cells are ... and ...
```

The BASIC statement for this decision could be:

```
IF first.number = second.number THEN PRINT "The contents
of the two memory cells are equal." ELSE PRINT "The
contents of the two memory cells are";first.number;"and";
second.number
```

This statement can be included in a program as follows:

```
10 first.number = 4
20 second.number = 4
30 IF first.number = second.number THEN PRINT "The
contents of the two memory cells are equal." ELSE PRINT
"The contents of the two memory cells are";first.number;
"and";second.number
```

Enter the program noticing the space after the IF and the spaces on both sides of THEN and ELSE. RUN the program. When the computer executes line 30 it finds that the condition is true as the contents of the two memory cells are equal. Hence it displays on the screen:

```
The contents of the two memory cells are equal.
```

Change line 10 to:

```
10 first.number = 7
```

RUN the program again. When the computer executes line 30 it finds that the condition is not true as the contents of the two memory cells are not equal. Hence it displays on the screen:

```
The contents of the two memory cells are 7 and 4
```

Example 12a

Write a program to INPUT two numbers and test if they are equal.

A possible program could be:

```
10 REM Demonstration of the IF...THEN...ELSE statement
20 INPUT "Type a number ",first.number
30 INPUT "Type another number ",second.number
40 IF first.number = second.number THEN PRINT "The two
numbers are equal." ELSE PRINT "The numbers you typed
were";first.number;"and";second.number
```

Enter and RUN the program. Input the number 10 for both the

first and the second numbers. The computer displays on the screen:

```
The two numbers are equal.
```

RUN the program again but this time **INPUT** the number 3 for the first number and 5 for the second. The computer now displays on the screen:

```
The numbers you typed were 3 and 5
```

Tutorial

12.1) Write a program to **INPUT** a number and test if it is equal to 7. Depending on the result of the test the computer should display on the screen:

```
You entered the correct number.
```

or

```
You entered the wrong number.
```

12.2) Write a program to generate a random number between 1 and 10. It should ask you to guess this number and then tell you whether or not your guess is correct.

12.3) Write a program to generate and display a random number between 1 and 10. It should then ask you to **INPUT** two numbers which add up to this number and tell you whether or not you are correct.

The computer can also test the conditions in Table 12.1.

Condition	BASIC symbol
equal to	=
not equal to	<>
greater than	>
less than	<
less than or equal to	<= or =>
greater than or equal to	>= or =>

Table 12.1

Remember that the less than < and greater than > signs are the **SHIFT ,** and **SHIFT .** keys.

Example 12b

Write a program that asks you to INPUT two different numbers and then displays:

```
.... is greater than ....
```

The program could be:

```
10 REM Comparison of two numbers
20 INPUT "Type a number ",first.number
30 INPUT "Type a different number ",second.number
40 IF first.number > second.number THEN
PRINT first.number;"is greater than";second.number ELSE
PRINT second.number;"is greater than";first.number
```

Enter and RUN the program. If you INPUT the numbers 7 and 12 when requested, the computer will display on the screen:

```
12 is greater than 7
```

RUN the program again but this time INPUT the numbers 10 and 5. The computer displays:

```
10 is greater than 5
```

Tutorial

12.4) Repeat Example 12b but this time the computer should display:

```
... is less than ...
```

12.5) Write a program that generates a random number between 1 and 5 and displays whether or not it is less than or equal to 3.

Use of the IF . . . THEN . . . ELSE statement to compare strings

When the computer compares two strings it can only test whether or not they are equal. For the strings to be equal they

must be identical.

Tutorial

12.6) State whether or not the following pairs of strings are equal:

"Hawthorn Avenue"	"Hawthorn Avenue"
"Fletcher Grove"	"Fletcher Avenue"
"Amstrad computer"	"Amstrad computer"
"dog"	"dog "
"John"	"john"
"Come here."	"Come here"

The IF...THEN...ELSE statement for comparing strings is the same as for comparing numbers as shown in the following example:

Example 12c

Write a program which asks you to INPUT the password to enter a top security room. The computer should then display ENTER or NO ENTRY as appropriate.

A possible program could be:

```
10 REM Comparison of passwords
20 password$="Fido"
30 INPUT "What is the password? ",answer$
40 IF answer$ = password$ THEN PRINT "ENTER" ELSE PRINT "NO
ENTRY"
```

At line 40 the computer compares the contents of the string variables password$ and answer$. It displays ENTER or NO ENTRY depending on whether or not they are identical.

Tutorial

12.7) Write a program that asks you to guess a letter. The computer should then display either:

```
Correct.
```

or

```
Wrong.
```

12.8) Write a program that asks you to **INPUT** the capital of Norway. The computer should then display either:

```
Correct.
```

or

```
The capital of Norway is Oslo.
```

The ELSE is not essential

So far all the decisions have involved statements of the form: **IF...THEN...ELSE**. However some decisions do not require an **ELSE**. Consider the decisions:

- IF your hands are dirty THEN wash them.
- IF it rains THEN put up your umbrella.
- IF the grass is too long THEN cut it.

Each of these situations involve decisions of the following type: IF some condition is true THEN take the appropriate action. This is called an **IF...THEN** statement and can be used to instruct the computer to make decisions similar to those above. As before the computer can compare either two numbers or two strings as shown in the following example:

Example 12d

Change the program of Tutorial 12.2 so that the computer displays whether your guess is too high, too low or correct.

A possible program could be:

```
10 REM Guess the number between 1 and 10
20 REM Vary the sequence of random numbers
30 RANDOMIZE TIME
40 REM Create the random number
50 number = INT(RND*10)+1
60 REM Clear the screen
70 CLS
80 REM Ask for the person's guess
90 LOCATE 2,10
100 INPUT "Please guess the number 1 to 10 ",guess
110 REM Test if the guess is too high, too low or correct
120 IF number = guess THEN PRINT "You guessed correctly."
```

```
130 IF number < guess THEN PRINT "Your guess was too high."
140 IF number > guess THEN PRINT "Your guess was too low."
```

At line 120 the computer compares the contents of the memory cells labelled `number` and `guess`. If they are not equal it starts to execute line 130 immediately. If they are equal it displays:

```
You guessed correctly.
```

and then proceeds to execute line 130.

Tutorial

12.9) Explain the action of the computer when it executes lines 130 and 140 in the program of Example 12d.

12.10) Extend the library book program of Tutorial 11.13 so that there is a minimum fine of 6 pence and a maximum fine of 26 pence.

12.11) Write a simple calculator program that, using a good screen display, asks you to enter two numbers followed by a + − * or / to indicate whether the numbers have to be added, subtracted, multiplied or divided. It should then display the appropriate answer.

Solutions to the tutorials

12.1)
```
10 REM Tutorial 12.1
20 REM INPUT a number
30 INPUT "Enter a number ",number
40 IF number = 7 THEN PRINT "You entered the correct number."
ELSE PRINT "You entered the wrong number."
```

12.2)
```
10 REM Guess the number
20 REM Generate a random number from 1 to 10
30 RANDOMIZE TIME
40 random.number = INT(RND*10)+1
50 INPUT "Guess the number from 1 to 10 ",guess
60 IF guess = random.number THEN PRINT "You guessed the correct
number." ELSE PRINT "You guessed the wrong number."
```

12.3)
```
10 REM Addition test
20 REM Generate a random number from 1 to 10
30 RANDOMIZE TIME
40 random.number = INT(RND*10)+1
50 PRINT "Enter two numbers that add to";random.number
60 INPUT "First number = ",first.number
70 INPUT "Second number = ",second.number
80 total = first.number + second.number
90 IF total = random.number THEN PRINT "Correct" ELSE PRINT
"Wrong"
```

12.4)
```
10 REM Comparison of two numbers
20 INPUT "Type a number ",first.number
30 INPUT "Type a different number ",second.number
40 IF first.number < second.number THEN
PRINT first.number;"is less than";second.number ELSE PRINT
second.number;"is less than";first.number
```

12.5)
```
10 REM Tutorial 12.5
20 REM Clear the screen
30 CLS
40 REM Generate a random number between 1 and 5
50 RANDOMIZE TIME
60 random.number = INT(RND*5)+1
70 IF random.number <= 3 THEN PRINT "Random number less
than or equal to 3." ELSE PRINT "Random number greater
than 3."
```

12.6) The string "Hawthorn Avenue" is equal to the string "Hawthorn Avenue".
The string "Fletcher Grove" is not equal to the string "Fletcher Avenue".
The string "Amstrad computer" is not equal to the string "Amstrad computer" as the second one has an extra space between the words.
The string "dog" is not equal to the string "dog " as the second one has a space after the g of dog.
The string "John" is not equal to the string "john" as the first one starts with a capital letter.
The string "Come here." is not equal to the string "Come here" as the first one ends in a full stop.

12.7)
```
10 REM Guess the letter
20 REM Correct letter is c
30 letter$ = "c"
40 INPUT "Guess a letter ",guess$
50 IF guess$ = letter$ THEN PRINT "Correct." ELSE PRINT "Wrong."
```

12.8)
```
10 REM Capital of Norway
20 INPUT "What is the capital of Norway ",capital$
30 IF capital$ = "Oslo" THEN PRINT "Correct." ELSE PRINT "The
capital of Norway is Oslo."
```

12.9) At line 130 the computer tests if the content of the memory cell labelled number is less than that of the memory cell labelled guess. If it is less it displays:

```
Your guess was too high.
```

and then proceeds to execute line 140. Otherwise it executes line 140 immediately. At line 140 the computer tests if the content of the memory cell labelled number is greater than that of the memory cell labelled guess. If it is greater it displays:

```
Your guess was too low.
```

The program then displays the usual Ready message. Otherwise it displays the usual Ready message immediately.

12.10)
```
10 REM  Calculates the fine on an overdue book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
40 REM INPUT the days overdue
50 INPUT "How many days are the book overdue? ",
days.overdue
60 fine = days.overdue * charge.per.day
70 REM Now make minimum fine 6p and maximum fine 26p
80 IF fine < 6 THEN fine = 6
90 IF fine > 26 THEN fine = 26
100 REM Display the fine
110 PRINT "The amount to be paid is";fine;"pence."
```

12.11)
```
10 REM Calculator program
20 REM Clear the screen
```

```
30 CLS
40 REM INPUT the information
50 LOCATE 11,8
60 INPUT "First number = ",first.number
70 LOCATE 10,10
80 INPUT "Second number = ",second.number
90 REM Decide on type of sum
100 LOCATE 2,12
110 INPUT "Enter + - / * for the type of sum ",type$
120 REM Clear the screen
130 CLS
140 REM Display the result
150 LOCATE 10,12
160 IF type$ = "+" THEN PRINT first.number;"+";
second.number;"=";first.number + second.number
170 IF type$ = "-" THEN PRINT first.number;"-";
second.number;"=";first.number - second.number
180 IF type$ = "/" THEN PRINT first.number;"/";
second.number;"=";first.number / second.number
190 IF type$ = "*" THEN PRINT first.number;"*";
second.number;"=";first.number * second.number
200 REM Hide the cursor
210 PEN 0
```

Again and again

Loops

Look back at the program for Tutorial 11.13 where the fine for an overdue library book was calculated. It has the disadvantage that each time an overdue book was returned the librarian would have to **RUN** the program. It would save time in a busy library if the program ran continuously and the librarian only had to enter the number of overdue days. This would occur if the computer after executing line 80 automatically went back to line 40 so that it executed the lines in the following order: 10 20 30 40 50 60 70 80 40 50 60 70 80 40 50 60 70 80 40, etc.

As lines 40 to 80 would constantly be repeated they could be said to form a loop. A loop can be created using the **WHILE...WEND** statement.

The WHILE...WEND statement

The **WHILE** is used to mark the start of the loop and the **WEND** the end of it. The **WHILE** is followed by a condition. The computer executes the loop if this condition is true and escapes from it if the condition is false. The library book program would have the following structure:

```
10
20
30
34 REM Create a WHILE...WEND loop from which the computer cannot
escape
35 dummy.test = 0
36 WHILE dummy.test = 0
40
50
```

```
60
70
80
90 WEND
```

The WHILE at line 36 and the WEND at line 90 mark the beginning and the end of the loop which contains the lines 40, 50, 60, 70, and 80 as required.

If the condition:

```
dummy.test = 0
```

is true then the computer executes the loop.

Line 35 stores the number zero in the memory cell labelled dummy.test. Hence at line 36 the condition:

```
dummy.test = 0
```

is true and the loop is executed. The WEND statement at line 90 instructs the computer to go back to line 36 where the condition is again tested. As the contents of the memory cell labelled dummy.test have not been altered the condition is still true and the loop is executed again. In this way the loop is repeated indefinitely.

The complete program becomes:

```
10 REM Calculates the fine on an overdue library book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
34 REM Create a WHILE...WEND loop from which the computer cannot
escape
35 dummy.test = 0
36 WHILE dummy.test = 0
40   REM INPUT the days overdue
50   INPUT "How many days is the book overdue? ",
     days.overdue
60   fine = days.overdue * charge.per.day
70   REM Display the fine
80   PRINT "The amount to be paid is";fine;"pence."
90 WEND
```

The statements within the WHILE...WEND loop have been indented two spaces as this makes the program easier to read.

Enter and RUN the program noticing the space between the WHILE and dummy.test at line 36. Each time the compu-

ter executes the loop it asks you, at line 50, to **INPUT** the number of days the book is overdue and then at line 80 displays the fine on the screen. As the program is in an endless loop it can only be stopped by using the escape key which is the blue key labelled **ESC**. Stop the program by pressing the escape key twice. The computer will display:

```
Break in ..
Ready
```

The display:

```
Break in ..
```

indicates the line that the computer was executing when the program was stopped. The program is still stored in the computer's memory.

Example 13a

Extend the program of Tutorial 12.2 so that it constantly asks you to guess another number.

The program of Tutorial 12.2 would constantly ask you to guess another number if the following lines were added:

```
34 REM Create a WHILE...WEND loop from which the computer cannot
escape
35 dummy.test = 0
36 WHILE dummy.test = 0
70 WEND
```

Tutorial

13.1) Write an autobank program that constantly asks you how much money you wish to withdraw and then displays:

```
You wish to withdraw .... pounds.
```

13.2) Change the die program of Example 10e so that it displays another throw of the die whenever you press an **ENTER** key.

Counting the number of times the computer has executed the loop

Let's extend the above library book program so that it also indicates the number of overdue books that have been returned. The number of books will be the same as the number of times the computer has executed the loop. This number will be stored in a memory cell labelled `counter`. The contents of this memory cell can be increased by the statement:

```
counter = counter + 1
```

This instructs the computer to add one to the contents of the memory cell labelled `counter` and then store the result in the same memory cell. Hence if the number 5 was stored in the memory cell labelled `counter` then the instruction:

```
counter = counter + 1
```

would increase this number to 6.

The library book program would become:

```
10 REM Calculates the fine on an overdue library book
20 REM Fines are charged at 2p per day
30 charge.per.day = 2
32 REM The memory cell labelled counter stores the number of
overdue books that have been returned
33 counter = 0
34 REM Create a WHILE...WEND loop from which the computer cannot
escape
35 dummy.test = 0
36 WHILE dummy.test = 0
40   REM INPUT the days overdue
50   INPUT "How many days is the book overdue? ",
     days.overdue
60   fine = days.overdue * charge.per.day
70   REM Display the fine
80   PRINT "The amount to be paid is";fine;"pence."
84   REM Increase the overdue book count by one
85   counter = counter + 1
86   PRINT counter;"overdue books have been returned."
90 WEND
```

The computer enters the loop from lines 36 to 90 with the contents of the memory cell labelled `counter` equal to zero.

Every time the computer goes round the loop the contents of this memory cell will be increased by one at line 85 and displayed on the screen at line 86.

Tutorial

13.3) Change the program of Example 13a so that the computer displays the number of times that you guessed correctly and the total number of guesses.

13.4) Change the program of Tutorial 13.2 so that the computer also displays the number of times the die has been rolled.

How to get out of a loop

In the previous examples the computer is trapped in the loop and can never escape. In some cases, e.g. the library book program, this would not matter as you would want the program to **RUN** continuously until you switched off in the evening. Sometimes however you may wish the computer to go round the loop a given number of times and then escape from it.

Assume that you wished to modify the program of Tutorial 13.3 so that the computer escaped from the loop after 10 guesses and then displayed the number correct.

Consider the program:

```
10 REM Guess the number
 20 REM Vary the random number sequence
 30 RANDOMIZE TIME
 40 REM The memory cell labelled total.number stores the number
of guesses
 50 REM The memory cell labelled number.correct stores the
number of correct guesses
 60 total.number = 0
 70 number.correct = 0
 80 WHILE total.number < 10
 90   REM Generate a random number from 1 to 10
100   random.number = INT(RND*10)+1
110   INPUT "Guess the number ",guess
120   IF guess = random.number THEN PRINT "You guessed the
```

```
      correct number." ELSE PRINT "You guessed the wrong
      number."
 130  IF guess = random.number THEN number.correct =
      number.correct + 1
 140  total.number = total.number + 1
 150 WEND
 160 PRINT "You have had";number.correct;"correct guesses"
 170 PRINT "out of";total.number
```

The WHILE at line 80 and the WEND at line 150 mark the beginning and the end of the loop. If the condition:

```
total.number < 10
```

is true then the computer executes the loop.

The computer enters the loop with the content of the memory cell labelled total.number equal to zero. Each time the computer executes the loop the content of this memory is increased by one. This is repeated until the tenth execution of the loop when the content of this memory cell is increased to 10. This time when the computer goes back to line 80 the condition is found to be false and therefore the computer escapes from the loop by jumping to line 160 which is the first line after the WEND statement.

Example 13b

Modify the program of Example 12d so that you are given three attempts to guess the number.

A possible program could be:

```
 10 REM Guess the number between 1 and 10
 20 REM Vary the random number sequence
 30 RANDOMIZE TIME
 40 REM Create the random number
 50 number = INT(RND*10)+1
 60 REM Clear the screen
 70 CLS
 80 REM Give the person three guesses
 90 REM The memory cell labelled guesses.left stores the number
of guesses the person has left
100 guesses.left = 3
110 REM If the memory cell labelled repeat.loop$ contains "NO"
then escape from the loop
120 REM If the memory cell labelled repeat.loop$ contains "YES"
```

```
then execute the loop
130 repeat.loop$ = "YES"
140 WHILE repeat.loop$ = "YES"
150   REM Ask for the person's guess .
160   INPUT "Please guess the number 1 to 10 ",guess
170   REM Test if the guess is too high, too low or
      correct
180   IF guess = number THEN PRINT "You guessed
      correctly."
190   IF guess < number THEN PRINT "Your guess was too
      low."
200   IF guess > number THEN PRINT "Your guess was too
      high."
210   REM Update the number of guesses left
      and repeat.loop$
220   guesses.left = guesses.left - 1
230   IF guesses.left = 0 THEN repeat.loop$ = "NO"
240   IF guess = number THEN repeat.loop$ = "NO"
250 WEND
260 IF guess <> number THEN PRINT "The number was";number
```

Why should you always win if you are given four guesses?

Tutorial

13.5) Modify the program of Example 12c so that you have two attempts to enter the password.

13.6) Modify the program of Tutorial 13.1 so that you enter your wage. Each time you withdraw money the program tells you how much is left but will not let you be overdrawn.

13.7) Write a program to calculate the twelve monthly repayments on loans of £100, £200, £300 and £400 at an interest rate of 8% per annum payable on the total sum borrowed.

Loops within loops

It is possible to have a loop which is entirely contained within another loop. In this case the loops are said to be *nested*. Let's assume that the program of Tutorial 13.7 has to be extended so

that it displays the repayments on the capital sums for interest rates of 8%, 9% and 10%.

In Tutorial 13.7 the interest rate was fixed at 8%. A WHILE...WEND loop could be used to make the interest rate 8%, 9% or 10% as required. The program would then have the structure:

```
 10 REM Monthly repayments on a loan
 20 percentage.interest = 8
 30 WHILE percentage.interest <= 10
 40   REM Clear the screen
 50   CLS
 60   interest.rate = percentage.interest / 100
 70
  .
  . Insert lines 30 to 130 of the program
  .       for tutorial 13.7
  .
170
180   REM Change the  percentage interest for the
      next loop
190   percentage.interest = percentage.interest + 1
200   REM Wait until an ENTER key is pressed
210   INPUT "Press an ENTER key.",a$
220 WEND
```

The WHILE...WEND loop from lines 30 to 220 would make the computer execute the program of Tutorial 13.7 three times with interest rates of 8%, 9% and 10%. The complete program would be:

```
 10 REM Monthly repayments on a capital sum
 20 percentage.interest = 8
 30 WHILE percentage.interest <= 10
 40   REM Clear the screen
 50   CLS
 60   interest.rate = percentage.interest / 100
 70   loan = 100
 80   WHILE loan <= 400
 90     interest = loan * interest.rate
100     total.repayment = interest + loan
110     monthly.repayment = total.repayment / 12
120     real.monthly.repayment =
```

```
        ROUND(monthly.repayment,2)
130     PRINT "monthly repayment on a";loan;"pound loan"
140     PRINT "at";percentage.interest;"per cent is";
        real.monthly.repayment;"pounds."
150     REM Change the loan for the next loop
160     loan = loan + 100
170   WEND
180   REM Change the percentage interest for the next loop
190   percentage.interest = percentage.interest + 1
200   REM Wait until an ENTER key is pressed
210   INPUT "Press an ENTER key.",a$
220 WEND
```

Example 13c

Write a program to display the multiplication tables.

A possible program could be:

```
 10 REM Multiplication tables
 20 table = 1
 30 WHILE table <= 10
 40   number = 1
 50   WHILE number <= 10
 60     product = number * table
 70     PRINT number;"*";table;"=";product
 80     REM Increase number for the next loop
 90     number = number + 1
100   WEND
110   REM Increase table for the next loop
120   table = table +1
130   REM Press an ENTER key to continue
140   INPUT "Press an ENTER key.",a$
150 WEND
```

Tutorial

13.8) Modify the program of Example 13b so that the computer continually gives you three attempts to guess a number.

Solutions to the tutorials

13.1)
```
10 REM Auto bank program
20 REM Clear the screen
30 CLS
40 REM Create a WHILE...WEND loop from which the computer
cannot escape
50 dummy.test = 0
60 WHILE dummy.test = 0
70   REM INPUT the amount to be withdrawn
80   INPUT "How much do you wish to withdraw? ",cash
90   REM Display amount to be withdrawn
100  PRINT "You wish to withdraw";cash;"pounds."
110 WEND
```

13.2)
```
10 REM Program to simulate a die
20 REM Vary the random number sequence
30 RANDOMIZE TIME
40 REM Create a WHILE...WEND loop from which the computer
cannot escape
50 dummy.test = 0
60 WHILE dummy.test = 0
70   CLS
80   random.number = INT(RND*6)+1
90   LOCATE 12,10
100  PRINT "The die shows";random.number
110  REM Wait until an ENTER key is pressed
120  LOCATE 1,12
130  INPUT "Press ENTER for the next roll of die ",a$
140 WEND
```

As line 130 is an INPUT statement the computer will wait until you press an **ENTER** key.

13.3)
```
10 REM Guess the number
20 REM Vary the random number sequence
30 RANDOMIZE TIME
40 REM The memory cell labelled total.number stores the number
of guesses
50 REM The memory cell labelled number.correct stores the
number of correct guesses
60 total.number = 0
70 number.correct = 0
```

```
 80 REM Create a WHILE...WEND loop from which the computer
cannot escape
 90 dummy.test = 0
100 WHILE dummy.test = 0
110   REM Generate a random number from 1 to 10
120   random.number = INT(RND*10)+1
130   INPUT "Guess the number ",guess
140   IF guess = random.number THEN PRINT "You guessed the
      correct number." ELSE PRINT "You guessed the
      wrong number."
150   IF guess = random.number THEN number.correct =
      number.correct + 1
160   total.number = total.number + 1
170   PRINT "You have had";number.correct;"correct
      guesses"
180   PRINT "out of";total.number
190 WEND
```

13.4)

```
 10 REM Program to simulate a die
 20 REM Vary the sequence of random numbers
 30 RANDOMIZE TIME
 40 REM The memory cell labelled number.of.throws stores the
number of times the die has been thrown
 50 number.of.throws = 0
 60 REM Create a WHILE...WEND loop from which the computer
cannot escape
 70 dummy.test = 0
 80 WHILE dummy.test = 0
 90   REM Clear the screen
100   CLS
110   random.number = INT(RND*6)+1
120   LOCATE 12,10
130   PRINT "The die shows";random.number
140   REM Update number of throws and display it
150   number.of.throws = number.of.throws + 1
160   LOCATE 7,12
170   PRINT "The die has been rolled";number.of.throws;
      "times."
180   REM Wait until an ENTER key is pressed
190   LOCATE 1,14
200   INPUT "Press ENTER for the next roll of die",a$
210 WEND
```

13.5)
```
10 REM Comparison of passwords
20 REM Clear the screen
30 CLS
40 password$ = "Fido"
50 REM If the memory cell labelled repeat.loop$ contains "NO"
then escape from the loop
60 REM If the memory cell labelled repeat.loop$ contains "YES"
then execute the loop
70 repeat.loop$ = "YES"
80 REM The memory cell labelled tries.left stores the number of
remaining attempts to enter the password
90 tries.left = 2
100 WHILE repeat.loop$ = "YES"
110   REM INPUT the password
120   INPUT "What is the password? ",answer$
130   IF answer$ = password$ THEN repeat.loop$ = "NO"
140   tries.left = tries.left - 1
150   IF tries.left = 0 THEN repeat.loop$ = "NO"
160 WEND
170 IF answer$ = password$ THEN PRINT "ENTER" ELSE
PRINT "NO ENTRY"
```

13.6)
```
10 REM Auto bank program
20 REM Clear the screen
30 CLS
40 REM INPUT your wage
50 INPUT "What is your wage? ",wage
60 REM Your current balance is stored in the memory cell
labelled money
70 money = wage
80 REM No money left is indicated by the memory cell labelled
repeat.loop$ containing "NO"
90 REM While there is money left the memory cell labelled
repeat.loop$ contains "YES"
100 repeat.loop$ = "YES"
110 WHILE repeat.loop$ = "YES"
120   REM INPUT the amount to be withdrawn
130   INPUT "How much do you wish to withdraw? ",cash
140   REM Display amount to be withdrawn
150   PRINT "You wish to withdraw";cash;"pounds."
160   REM The memory cell labelled money.left stores the
      amount of money you would have after this
      transaction
```

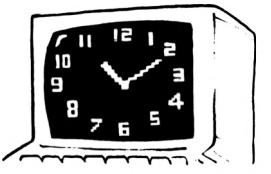
```
170   money.left = money - cash
180   REM Test if still in the black
190   IF money.left = 0 THEN repeat.loop$ = "NO"
200   IF money.left > 0 THEN PRINT "You have";money.left;
      "pounds in the bank."
210   IF money.left < 0 THEN PRINT "You do not have
      enough money." ELSE money = money.left
220 WEND
230 PRINT "You have no money left now."
```

13.7)
```
10 REM Monthly repayments on a loan
20 interest.rate = 8 / 100
30 loan = 100
40 WHILE loan <= 400
50   interest = loan * interest.rate
60   total.repayment = interest + loan
70   monthly.repayment = total.repayment / 12
80   real.monthly.repayment = ROUND(monthly.repayment,2)
90   PRINT "monthly repayment on";loan;"pound loan is"
100  PRINT real.monthly.repayment;"pounds."
110  REM Change loan for the next loop
120  loan = loan + 100
130 WEND
```

13.8)
```
10 REM Guess the number between 1 and 10
20 REM Vary the random number sequence
30 RANDOMIZE TIME
40 REM Create a WHILE...WEND loop from which the computer
cannot escape
50 dummy.test = 0
60 WHILE dummy.test = 0
70   REM Create the random number
80   number = INT(RND*10)+1
90   REM Give the person three guesses
100  REM The memory cell labelled guesses.left stores the
     number of guesses the person has left
110  guesses.left = 3
120  REM If the memory cell labelled repeat.loop$
     contains "NO" then escape from the loop
130  REM If the memory cell labelled repeat.loop$
     contains "YES" then execute the loop
```

```
140   repeat.loop$ = "YES"
150   WHILE repeat.loop$ = "YES"
160     REM Ask for the person's guess
170     INPUT "Please guess the number 1 to 10 ",guess
180     REM Test if the guess is too high, too low
        or correct
190     IF guess = number THEN PRINT "You guessed
        correctly."
200     IF guess < number THEN PRINT "Your guess was too
        low."
210     IF guess > number THEN PRINT "Your guess was too
        high."
220     REM Update the number of guesses left
        and repeat.loop$
230     guesses.left = guesses.left - 1
240     IF guesses.left = 0 THEN repeat.loop$ = "NO"
250     IF guess = number THEN repeat.loop$ = "NO"
260   WEND
270   IF guess <> number THEN PRINT "The number
      was";number
280 WEND
```

Making use of the Amstrad CPC 664's clock



Enter, renumber and **RUN** the overdue library book program of Chapter 13.

```
 10 REM Calculates the fine on an overdue library book
 20 REM Fines are charged at 2p per day
 30 charge.per.day = 2
 40 REM The memory cell labelled counter stores the number of
overdue library books that have been returned
 50 counter = 0
 60 REM Create a WHILE...WEND loop from which the computer
cannot escape
 70 dummy.test = 0
 80 WHILE dummy.test = 0
 90   REM INPUT the days overdue
100   INPUT "How many days is the book overdue? ",
      days.overdue
110   fine = days.overdue * charge.per.day
120   REM Display the fine
130   PRINT "The amount to be paid is";fine;"pence."
140   REM Increase the overdue book count by one
150   counter = counter + 1
160   PRINT counter;"overdue books have been returned."
170 WEND
```

The screen presentation is not ideal as all the previous fines are also displayed with the current one. If the computer had been instructed to wait for five seconds after displaying the fine at line 130 the screen could then have been cleared. The five second delay would have given the librarian time to read

the fine. Delays can be created by using the TIME statement.

The TIME statement

Your Amstrad CPC 664 contains a counter, labelled TIME, which is set to zero when the computer is switched on. Thereafter its value is automatically increased by one every 1/300 of a second. Table 14.1 shows the increase in the value of this counter in a given time.

Time delay		Increase in the value of the TIME counter
1/300	second	1
1/20	second	15
1/2	second	150
1	second	300
3	second	900
5	second	1500
30	second	9000
1	minute	18000
1	hour	1080000

Table 14.1 The TIME counter

The value of the TIME counter can be displayed on the screen by typing:

```
PRINT TIME
```

Press an **ENTER** key. The computer now displays the value of the TIME counter at the instant the **ENTER** key was pressed.

Example 14a

In this example you can check that the TIME counter increases as shown in Table 14.1. An accurate watch will be required. Type:

```
PRINT TIME
```

Note the time on your watch and at the same instant press an **ENTER** key. The computer displays on the screen the current value of the TIME counter. Type:

```
PRINT TIME
```

When one minute has elapsed, press an **ENTER** key. The computer now displays on the screen the value of the TIME counter after the one minute delay. Check that the difference between the numbers displayed by the computer is approximately 18000 in agreement with Table 14.1.

Creating a delay in a program

Table 14.1 shows that a five second delay can be created by instructing the computer to wait until the value of the TIME counter has increased by 1500. The following sequence would achieve this:

Add 1500 to the current value of the TIME counter and store the result in a memory cell labelled end.time using the statements:

```
10 REM Creates a five second delay
20 end.time = TIME + 1500
```

Now trap the computer in a loop until the value of the TIME counter has increased to greater than the content of the memory cell labelled end.time. This will take five seconds and could be achieved with an empty WHILE...WEND loop:

```
30 WHILE TIME < end.time
40 WEND
```

The complete delay is:

```
10 REM Creates a five second delay
20 end.time = TIME + 1500
30 WHILE TIME < end.time
40 WEND
```

Enter and RUN the program. Five seconds pass before the Ready message is displayed on the screen showing that the computer takes five seconds to execute the program.

A five second delay can be created after line 130 of the above overdue library book program by inserting the lines:

```
131   REM Create a five second delay
132   end.time = TIME + 1500
```

```
133   WHILE TIME < end.time
134   WEND
```

Before the next question is asked the screen can be cleared by the lines:

```
135   REM Clear the screen
136   CLS
```

Insert these lines and RUN the program.

Example 14b

Modify the password program of Example 12c so that the computer displays the ENTER or NO ENTRY for three seconds before requesting the password from the next person.

A possible program could be:

```
 10 REM comparison of passwords
 20 REM Create a WHILE...WEND loop from which the computer
cannot escape
 30 dummy.test = 0
 40 WHILE dummy.test = 0
 50   REM Clear the screen
 60   CLS
 70   password$ = "Fido"
 80   REM INPUT the password
 90   INPUT "What is the password? ",answer$
100   IF answer$ = password$ THEN PRINT "ENTER" ELSE PRINT
      "NO ENTRY"
110   REM Create a three second delay
120   end.time = TIME + 900
130   WHILE TIME < end.time
140   WEND
150 WEND
```

Tutorial

14.1) Modify the program of Tutorial 11.5 so that the cost of petrol is displayed for six seconds before the screen is cleared and the program repeats itself.

14.2) Modify the password program of Tutorial 13.5 so that the computer displays the ENTER or NO ENTRY for

three seconds before requesting the password from the next person.

14.3) Modify the die program of Tutorial 13.4 so that it rolls the die every twenty seconds rather than when an ENTER key is pressed.

14.4) Modify the library book program so that the fine is displayed for four seconds. The computer should also centre the text on the screen and display it in bright yellow on blue except for the fine which should be displayed in bright red on blue.

Solutions to the Tutorials

14.1)
```
10 REM Purchase of petrol
20 REM Select mode 1
30 MODE 1
40 REM Create a WHILE...WEND loop from which the computer
cannot escape
50 dummy.test = 0
60 WHILE dummy.test = 0
70   LOCATE 1,8
80   PRINT "How many litres of petrol do you wish"
90   LOCATE 12,10
100  PRINT "to purchase";
110  REM INPUT number of litres of petrol
120  INPUT quantity
130  REM Clear the screen
140  CLS
150  REM Select bright red characters
160  PEN 3
170  REM Calculate and display the cost
180  cost.per.litre = 42.8
190  cost.pounds = cost.per.litre * quantity / 100
200  real.cost = ROUND(cost.pounds,2)
210  LOCATE 5,10
220  PRINT "You have asked for";quantity;"litres."
230  LOCATE 11,12
240  PRINT "The cost is ";real.cost
250  REM Insert the £ sign
260  LOCATE 23,12
270  PRINT "£"
280  REM Create a delay of six seconds
```

```
290   end.time = TIME + 1000
300   WHILE TIME < end.time
310   WEND
320   REM Clear the screen
330   CLS
340 WEND
```

14.2)

```
 10 REM comparison of passwords
 20 REM Create a WHILE...WEND loop from which the computer
cannot escape
 30 dummy.test = 0
 40 WHILE dummy.test = 0
 50   REM Clear the screen
 60   CLS
 70   password$ = "Fido"
 80   REM If the memory cell labelled repeat.loop$
      contains "NO" then escape from the loop
 90   REM If the memory cell labelled repeat.loop$
      contains "YES" then execute the loop
100   repeat.loop$ = "YES"
110   REM The memory cell labelled tries.left stores the
      number of remaining attempts to enter the password
120   tries.left = 2
130   WHILE repeat.loop$ = "YES"
140     REM INPUT the password
150     INPUT "What is the password? ",answer$
160     IF answer$ = password$ THEN repeat.loop$ = "NO"
170     tries.left = tries.left - 1
180     IF tries.left = 0 THEN repeat.loop$ = "NO"
190   WEND
200   IF answer$ = password$ THEN PRINT "ENTER" ELSE PRINT
      "NO ENTRY"
210   REM Create a three second delay
220   end.time = TIME + 900
230   WHILE TIME < end.time
240   WEND
250 WEND
```

14.3)

```
 10 REM Program to simulate a die
 20 REM Vary the random number sequence
```

```
 30 RANDOMIZE TIME
 40 REM The memory cell labelled number.of.throws stores the
number of times the die has been thrown
 50 number.of.throws = 0
 60 REM Create a WHILE...WEND loop from which the computer
cannot escape
 70 dummy.test = 0
 80 WHILE dummy.test = 0
 90   REM Clear the screen
100   CLS
110   random.number = INT(RND*6)+1
120   LOCATE 12,10
130   PRINT "The die shows";random.number
140   REM Update the number of throws and display it
150   number.of.throws = number.of.throws + 1
160   LOCATE 7,12
170   PRINT "The die has been rolled";number.of.throws;
      "times."
180   REM Create a 20 second delay
190   end.time = TIME + 6000
200   WHILE TIME < end.time
210   WEND
220 WEND
```

14.4)

```
 10 REM Calculates the fine on an overdue library book
 20 REM Clear the screen
 30 CLS
 40 REM Fines are charged at 2p per day
 50 charge.per.day = 2
 60 REM The memory cell labelled counter stores the number of
overdue library books that have been returned
 70 counter = 0
 80 REM Create a WHILE...WEND loop from which the computer
cannot escape
 90 dummy.test = 0
100 WHILE dummy.test = 0
110   REM INPUT the days overdue
120   LOCATE 1,10
130   INPUT "How many days is the book overdue? ",
      days.overdue
140   fine = days.overdue * charge.per.day
150   REM Display the fine
```

```
160   LOCATE 3,12
170   PRINT "The amount to be paid is";
180   REM Select bright red characters
190   PEN 3
200   PRINT fine;
210   REM Select bright yellow characters
220   PEN 1
230   PRINT "pence."
240   REM Create a four second delay
250   end.time = TIME + 1200
260   WHILE  TIME < end.time
270   WEND
280   REM Clear the screen
290   CLS
300   REM Increase the overdue book count by one
310   counter = counter + 1
320   REM Display the number of overdue books returned
330   LOCATE 2,8
340   PRINT counter;"overdue books have been returned."
350 WEND
```

Chapter Fifteen

Making your pictures move

Animation

In Chapter 8 the Amstrad CPC 664's building blocks were used to create pictures on the screen. In this chapter the pictures will be animated which means that they will be made to move. Animation with the computer uses the same techniques as animation in cartoons. In these, apparent movement is suggested to the viewer by showing a series of still pictures each portraying the next stage in the movement.

If building blocks 248 and 249 are displayed alternately on the screen then an image of a man doing leg astride exercises can be obtained. The following program will display these blocks at screen position 8,10:

```
 10 REM Legs astride exercise
 20 REM Select mode 0
 30 MODE 0
 40 REM Create a WHILE...WEND loop from which the computer
cannot escape
 50 dummy.test = 0
 60 WHILE dummy.test = 0
 70   REM Display building block 248 at screen
      position 8,10
 80   LOCATE 8,10
 90   PRINT CHR$(248)
100   REM Display building block 249 at screen
      position 8,10
110   LOCATE 8,10
120   PRINT CHR$(249)
130 WEND
```

Enter and **RUN** the program. The computer displays the man but he is doing his exercises too fast. Press the escape key twice to terminate the program. The man can be slowed down by inserting a delay after each image is drawn on the screen (i.e. after lines 90 and 120). The following lines would produce delays of ¼ of a second:

```
 91   REM Create a delay of 1/4 second
 92   end.time = TIME + 75
 93   WHILE TIME < end.time
 94   WEND
121   REM Create a delay of 1/4 second
122   end.time = TIME + 75
123   WHILE TIME < end.time
124   WEND
```

Add these lines to the program and **RUN** it. The man now does his leg astride exercises at a reasonable speed. Experiment with other time delays.

Example 15a

Using building blocks 248 and 250, write a program that displays a man swinging his left leg to the side and then back again.

A possible program could be:

```
 10 REM Man doing left leg swings
 20 REM Select mode 0
 30 MODE 0
 40 REM Create a WHILE...WEND loop from which the computer
cannot escape
 50 dummy.test = 0
 60 WHILE dummy.test = 0
 70   REM Display building block 248 at screen
      position 8,10
 80   LOCATE 8,10
 90   PRINT CHR$(248)
100   REM Create a delay of 1/4 second
110   end.time = TIME + 75
120   WHILE TIME < end.time
130   WEND
140   REM Display building block 250 at screen
      position 8,10
```

```
150   LOCATE 8,10
160   PRINT CHR$(250)
170   REM Create a delay of 1/4 second
180   end.time = TIME + 75
190   WHILE TIME < end.time
200   WEND
210 WEND
```

Tutorial

15.1) Using building blocks 248 and 251, write a program that displays a man swinging his right leg to the side and then back again.

15.2) Using building blocks 248, 250, and 251, write a program that displays a man swinging one leg and then the other to the side and back again.

Before a picture can be moved across the screen some of the BASIC statements must be re-examined.

Using labels as parameters

In the previous chapters, statements such as LOCATE, PEN, etc. have always been followed by one or two numbers:

```
LOCATE 6,10
PEN 3
```

These numbers are called the *parameters* of the statement. It is possible to use the labels given to memory cells as the parameters. Look at the following program:

```
10 REM Demonstration of the use of labels given to memory
cells as parameters
20 colour = 3
30 PEN colour
40 PRINT "The pen colour is now bright red."
50 colour = 2
60 PEN colour
70 PRINT "The pen colour is now bright cyan."
```

Enter and RUN the program. Since, at line 20, the number 3 is stored in the memory cell labelled colour, line 30 is

equivalent to:

```
PEN 3
```

Hence line 40 is displayed on the screen in bright red. Similarly lines 50 and 60 change the pen colour to bright cyan.

Example 15b

Extend the program of Example 15a so that the man changes colour randomly.

Add the following lines to the program of Example 15a

```
15 REM Vary the random number sequence
16 RANDOMIZE TIME
61   REM Select a random colour
62   colour = INT(RND*15)+1
63   PEN colour
```

Tutorial

15.3) Extend the program of Example 15b so that the man and his background change colour randomly with the rest of the screen blue.

Creating movement across the screen

A multiplication sign can be moved across the screen by drawing the sign in column one, waiting a short time, deleting the sign in column one, and then repeating this process in the other columns until the * is across the screen.

The following program will move the * across row 8 of the screen in mode 0:

```
 10 REM Moves a star across the screen
 20 REM Select mode 0
 30 MODE 0
 40 REM The memory cell labelled column stores the screen column
where the * is currently displayed
 50 column = 1
 60 REM Lines 70 to 200 form the loop that moves the * across
the screen
 70 WHILE column <= 20
```

```
 80   REM Draw the * on the screen
 90   LOCATE column,8
100   PRINT "*"
110   REM Delay for 1/4 second
120   end.time = TIME + 75
130   WHILE TIME < end.time
140   WEND
150   REM Delete the * from the screen
160   LOCATE column,8
170   PRINT " "
180   REM Increase the screen column position by one
190   column = column + 1
200 WEND
```

Enter and RUN the program. The * should now move across the screen. Experiment with different delays at line 120. Also try adding a second delay after line 170 as follows:

```
171   REM delay for 1/4 second
172   end.time =TIME + 75
173   WHILE TIME < end.time
174   WEND
```

RUN the program and notice the pulsating motion.

Example 15c

Write a program in mode 0 to move a * diagonally from screen position 1,1 to 20,20.

Add the following lines to the last program:

```
 41 REM The memory cell labelled row stores the screen row where
the * is currently displayed
 51 row = 1
191 row = row + 1
```

and change the lines 90, 160 and 180:

```
 90   LOCATE column,row
160   LOCATE column,row
180   REM Increase the screen column and row position by one
```

Tutorial

15.4) Write a program that displays in mode 0 a * moving vertically down the screen in column 10.

15.5) Write a program in mode 0 that displays a * moving up the screen in column 10.

15.6) Extend the program of Tutorial 15.4 so that the * changes colour randomly.

Moving larger pictures

The * that was moved around the screen in the last section only occupied one screen position. However a similar principle can be used to move drawings that consist of several building blocks.

Let's move a space ship vertically up the screen. The section of a mode 0 screen display planner in Fig. 15.1 shows the space ship which will be bright red on a blue background.

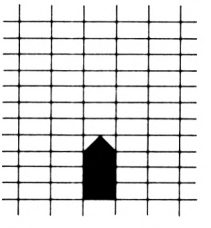


Figure 15.1 Screen display planner showing a spaceship

The following program will draw the space ship:

```
 10 REM Draw the space ship
 20 REM Select mode 0
 30 MODE 0
 40 REM The memory cell labelled row stores the screen row of
the display. Start with the top of the space ship at row 19
 50 row = 19
 60 REM Select bright red characters
 70 PEN 3
 80 REM Draw the space ship
 90 REM The memory cell labelled counter stores the number of
times the WHILE...WEND loop has been executed. During each
execution of the loop the computer will display a row of the
space ship.
100 counter = 0
110 WHILE counter < 4
```

```
120   REM The memory cell labelled block stores the
      building block being displayed
130   IF counter = 0 THEN block = 244 ELSE block = 143
140   LOCATE 10,row
150   PRINT CHR$(block)
160   counter = counter + 1
170   REM Increase the screen row for the next line of the
      space ship
180   row = row + 1
190 WEND
```

Enter and RUN the program. A bright red space ship should be displayed on the screen.

The space ship can be moved up the screen by constantly repeating the following steps:

1) Drawing the space ship one row further up the screen.
2) Deleting the bottom character of the previous space ship. It is only necessary to delete the last row of the space ship as the other rows will be deleted when the space ship is redrawn.
3) Waiting a short time.

This can be achieved by modifying the program that drew the space ship to give it the following structure:

```
 10
  .
  .
 50
 51 REM Repeat this loop until the space ship reaches the top of
the screen
 52 WHILE row >=1
 60
  .
  .
190
200   REM Delete last row of the previous space ship
210   LOCATE 10,row
220   PRINT " "
230   REM Delay a 1/8 second
240   end.time = TIME + 38
250   WHILE TIME < end.time
260   WEND
```

```
270   REM Decrease the screen row by five to set it to the
      top of the next space ship to be displayed
280   row = row - 5
290 WEND
```

The WHILE...WEND loop from lines 52 to 290 moves the space ship up the screen by:

1) Drawing the space ship, in lines 60 to 190, with its top at screen position 10,row.
2) Deleting, in lines 200 to 220, the last row of the previous space ship.
3) Waiting ⅛ of a second before repeating the loop.

Add these extra lines to the program that drew the space ship and RUN it. The space ship will move up the screen. SAVE the program onto a disk and call it ROCKET.

Example 15d

Extend the above program so that the space ship's exhaust fuels are visible. Building block 253 displayed in bright yellow will be used to represent the fuel.

A possible program could be:

```
 10 REM Draw the space ship with fuel burn
 20 REM Select mode 0
 30 MODE 0
 40 REM The memory cell labelled row stores the screen row of
the display. Start with the top of the space ship at row 19
 50 row = 19
 60 REM Repeat this loop until the space ship reaches the top of
the screen
 70 WHILE row >= 1
 80   REM Select bright red characters
 90   PEN 3
100   REM Draw the space ship
110   REM The memory cell labelled counter stores the
      number of times the WHILE...WEND loop has been
      executed
120   counter = 0
130   WHILE counter < 4
140     REM The memory cell labelled block stores the
        building block being displayed
```

```
150     IF counter = 0 THEN block = 244 ELSE block = 143
160     LOCATE 10,row
170     PRINT CHR$(block)
180     counter = counter + 1
190     REM Increase the screen row for next line of the
        space ship
200     row = row + 1
210   WEND
220   REM Draw the fuel in bright yellow
230   PEN 1
240   LOCATE 10,row
250   PRINT CHR$(253)
260   REM Increase the screen row for the next line
270   row =row + 1
280   REM Delete last row of the previous space ship
290   LOCATE 10,row
300   PRINT " "
310   REM Delay a 1/8 second
320   end.time = TIME + 38
330   WHILE TIME < end.time
340   WEND
350   REM Decrease the screen row by six to set it to the
      top of the next drawing
360   row = row - 6
370 WEND
```

You may wish to vary the time delay at lines 320 to 340. SAVE the program onto a disk and call it FUEL.

Tutorial

15.7) Modify the program of Example 15d so that the fuel ceases to be seen half way up the screen. SAVE the program onto a disk and call it HALFUEL.

15.8) Modify the program of Tutorial 15.7 so that it writes the letters USA down the rocket.

Solutions to the tutorials

15.1) The program is the same as for Example 15a except for the lines:

```
140 Display building block 251 at screen position 8,10
```

```
160 PRINT CHR$(251)
```

15.2)

```
 10 REM Man doing left and right leg swings
 20 REM Select mode 0
 30 MODE 0
 40 REM Create a WHILE...WEND loop from which the computer
cannot escape
 50 dummy.test = 0
 60 WHILE dummy.test = 0
 70   REM Display building block 248 at screen
      position 8,10
 80   LOCATE 8,10
 90   PRINT CHR$(248)
100   REM Create a delay of 1/4 second
110   end.time = TIME + 75
120   WHILE TIME < end.time
130   WEND
140   REM Display building block 250 at screen
      position 8,10
150   LOCATE 8,10
160   PRINT CHR$(250)
170   REM Create a delay of 1/4 second
180   end.time = TIME + 75
190   WHILE TIME < end.time
200   WEND
210   REM Display building block 248 at screen
      position 8,10
220   LOCATE 8,10
230   PRINT CHR$(248)
240   REM Create a delay of 1/4 second
250   end.time = TIME + 75
260   WHILE TIME < end.time
270   WEND
280   REM Display building block 251 at screen
      position 8,10
290   LOCATE 8,10
300   PRINT CHR$(251)
310   REM Create a delay of 1/4 second
320   end.time = TIME + 75
330   WHILE TIME < end.time
340   WEND
350 WEND
```

15.3) Add the following lines to the program of Example 15b:

```
 64   REM Select a random colour
 65   colour = INT(RND*15)+1
 66   PAPER colour
```

15.4)

```
 10 REM Moves a star down the screen
 20 REM Select mode 0
 30 MODE 0
 40 REM The memory cell labelled row stores the screen row where
the * is currently displayed
 50 row = 1
 60 REM Lines 70 to 200 form the loop that moves the * down the
screen
 70 WHILE row <= 25
 80   REM Draw the * on the screen
 90   LOCATE 10,row
100   PRINT "*"
110   REM Delay for 1/4 second
120   end.time = TIME + 75
130   WHILE TIME < end.time
140   WEND
150   REM Delete the * from the screen
160   LOCATE 10,row
170   PRINT " "
180   REM Increase the screen row position by one
190   row = row + 1
200 WEND
```

15.5) The program is the same as for Tutorial 15.4 except for the following lines:

```
 10 REM Moves a star up the screen
 50 row = 25
 60 REM Lines 70 to 200 form the loop that moves the * up the
screen
 70 WHILE row >= 1
180 REM Decrease the screen row position by one
190 row = row - 1
```

15.6) Add the following lines to the program of Tutorial 15.4:

```
 15 REM Vary the random number sequence
 16 RANDOMIZE TIME
 71   REM Select a random colour
 72   colour = INT(RND*15)+1
```

```
73  PEN colour
```

15.7) Change line 250 of the program called FUEL of Example 15d to:

```
250  IF row > 12 THEN PRINT CHR$(253) ELSE PRINT " "
```

15.8) Add the following lines to the program called HALFUEL of Tutorial 15.7:

```
301  REM Transparent printing
302  PRINT CHR$(22)+CHR$(1)
303  REM Decrease row to the base of the body of the
space ship
304  row = row - 2
305  REM Display the A, the S and then the U
306  LOCATE 10,row
307  PRINT "A"
308  row = row -1
309  LOCATE 10,row
310  PRINT "S"
311  row = row -1
312  LOCATE 10,row
313  PRINT "U"
314  REM Normal printing
315  PRINT CHR$(22)+CHR$(0)
```

As line 310 has been overwritten, make it line 316:

```
316  REM Delay a 1/8 second
```

Change lines 350 and 360 to:

```
350  REM Decrease the screen row by two to set it to the top of
the next drawing
360  row = row - 2
```

Sound effects with the Amstrad CPC 664

Playing musical notes

Your Amstrad CPC 664 is able to play tunes which are composed of individual notes that can be generated by the **SOUND** statement. Turn the volume control, which is in front of the **ON–OFF** switch at the right hand edge of the computer, fully up. Type (noticing the space between **SOUND** and **1,478**):

```
SOUND 1,478
```

Press an **ENTER** key. The computer now plays middle C. The first number in the **SOUND** statement (i.e. 1) will be discussed later. The second number (i.e. 478) determines the note as shown in Table 16.1.

Note	Sound number	
G	1276	Octave −2
G#	1204	
A	1136	
A#	1073	
B	1012	
C	956	Octave −1
C#	902	
D	851	
D#	804	
E	758	
F	716	
F#	676	
G	638	
G#	602	
A	568	
A#	536	
B	506	

Note	Sound number	
middle C	478	
C#	451	
D	426	
D#	402	
E	379	
F	358	Octave 0
F#	338	
G	319	
G#	301	
A	284	
A#	268	
B	253	
C	239	
C#	225	
D	213	
D#	201	
E	190	Octave 1
F	179	
F#	169	
G	159	
G#	150	
A	142	

Table 16.1 Sound number values

Example 16a

Write a program to play the scale of C which consists of C,D,E,F,G,A,B in octave 0 and C in octave 1.

A possible program could be:

```
10 REM Scale of C
20 SOUND 1,478
30 SOUND 1,426
40 SOUND 1,379
50 SOUND 1,358
60 SOUND 1,319
70 SOUND 1,284
80 SOUND 1,253
90 SOUND 1,239
```

Enter and **RUN** the program. The computer plays the scale of C. When the computer executed line 20 it started to play

middle C. While playing this note it executed line 30 where it was told to play D. Since it was playing a note it put the note D into a queue and carried on to execute line 40 and so on. When it finished playing each note the computer immediately started to play the next note in the queue. As it takes longer to play the notes than to execute the statements the computer finished the program and displayed the usual `Ready` message while the scale was still being played. `RUN` the program again and observe this.

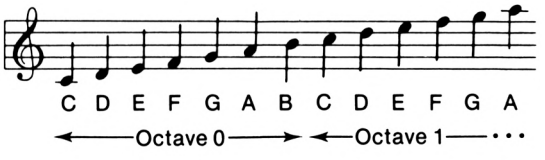
Tutorial

16.1) Write a program to play D,F#,A in octave 0 and D in octave 1.

16.2) Write a program to play A,E,C# in octave 1 and A in octave 0.

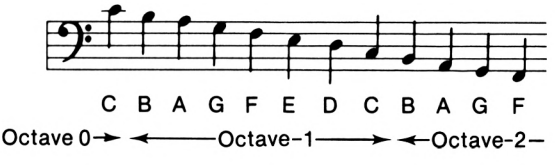
Reading music

The positions of the notes in the treble clef are:



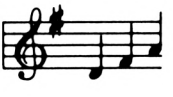
On the piano these notes are played with the right hand.

The positions of the notes in the bass clef are:



On the piano these notes are played with the left hand.

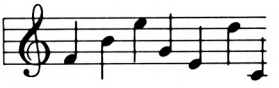
If there are any sharps (i.e. #) at the beginning of the clef then the note is sharpened.



These notes would be D,F#A. The F in all octaves would be sharpened.

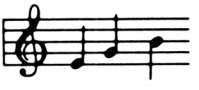
Tutorial

16.3) List the following notes:



Example 16b

Write a program to play the following notes:

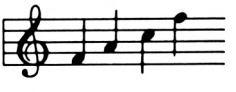


The notes are E,G,B in octave 0. A possible program could be:

```
10 REM Example 16.b
20 SOUND 1,379
30 SOUND 1,319
40 SOUND 1,253
```

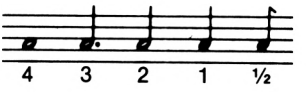
Tutorial

16.4) Write a program to play the following notes:

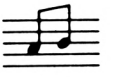


The duration of notes

A tune consists of notes that are played for different lengths of time. The duration of each note is described in terms of its number of beats. The notes are written as follows:



Two half beat notes are often written as:



The duration of each note can be controlled by the **SOUND** statement. Type:

```
SOUND 1,478,20
```

Press an **ENTER** key. The computer will now play middle C for 1/5 of a second. Type:

```
SOUND 1,478,100
```

Press an **ENTER** key. This time the computer plays middle C for one second.

The third number in the **SOUND** statement determines the duration of the note. The duration of the note in seconds is this number divided by 100. Therefore in the statement:

```
SOUND 1,478,20
```

the duration of the note is 20/100 which is 1/5 of a second. In the statement:

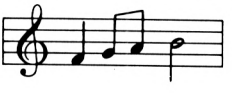
```
SOUND 1,478,100
```

the duration of the note is 100/100 which is one second.

If the computer is not given the duration of the note as in Example 16a then it makes the note last for 1/5 of a second.

Example 16c

Write a program to play the following notes:



If a one beat note lasts for one second the notes will be: F for one second, G for ½ of a second, A for ½ of a second and B for two seconds.

A possible program could be:

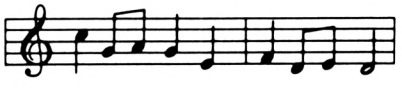
```
10 REM Example 16.c
20 SOUND 1,358,100
30 SOUND 1,319,50
40 SOUND 1,284,50
50 SOUND 1,253,200
```

Enter and **RUN** the program. If you feel that the speed is too fast or too slow then alter the third number in the **SOUND** statements.

Tutorial

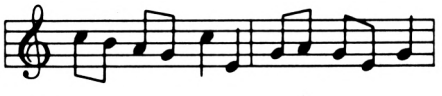
16.5) Write a program to play the scale of C so that each note lasts for one second

16.6) Write a program to play the following tune:



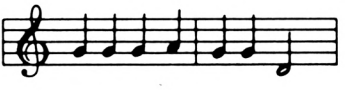
Make a one beat note last for ½ of a second. Notice that the music has been divided into bars each of which have the same number of beats.

16.7) Write a program to play the following extract from 'Sing a Song of Sixpence':



Repeated notes

Let's look at the first few bars of 'Good King Wenceslas':



A possible program could be:

```
10 REM Good King Wenceslas
20 SOUND 1,319,50
30 SOUND 1,319,50
40 SOUND 1,319,50
50 SOUND 1,284,50
60 SOUND 1,319,50
70 SOUND 1,319,50
80 SOUND 1,426,100
```

Enter and **RUN** the program. As the first three notes are all the same the computer runs them together which distorts the tune.

The best way to separate the notes is to insert an extra note of a short duration (1/20 of a second) between the notes. If you make these extra notes a very high frequency then the speaker will not be able to play them and they will not be heard. A suitable note would be:

```
SOUND 1,2,5
```

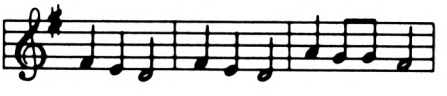
With these extra notes the above program becomes:

```
10 REM Good King Wenceslas
20 SOUND 1,319,50
25 SOUND 1,2,5
30 SOUND 1,319,50
35 SOUND 1,2,5
40 SOUND 1,319,50
50 SOUND 1,284,50
60 SOUND 1,319,50
65 SOUND 1,2,5
70 SOUND 1,319,50
80 SOUND 1,426,100
```

Add the extra notes to the program and **RUN** it.

Tutorial

16.8) Write a program to play the first few notes of 'Three Blind Mice'.



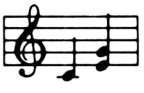
Several notes at once

Your Amstrad CPC 664 has three separate sound channels A, B and C which allows you to play up to three notes at the same time. So far you have only used channel A.

The first number in the **SOUND** statement is the channel number and determines the channel in which the note is to be played as follows:

Channel	Channel number
A	1
B	2
C	4

Let's write a program to play the following notes with each lasting for 2 seconds:



The C and G will be played in channel A and the E in channel B. The C will be played in channel A by the statements:

```
10 REM Play C on channel A
20 SOUND 1,478,200
```

You must now tell the computer to play G on channel A and E on channel B at the same time. The notes are said to *rendezvous*. When the computer is to play two or three notes at the same time then the channel numbers are changed as in Table 16.2.

In our example we want G on channel A to rendezvous with E on channel B. This is achieved with the statements:

```
30 REM Play G on channel A but rendezvous with the
note on channel B
40 SOUND 17,319,200
50 REM Play E on channel B but rendezvous with the
note on channel A
60 SOUND 10,379,200
```

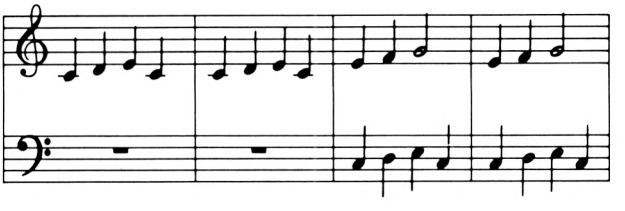
Channel number	Action
1	Send note to channel A only
2	Send note to channel B only
4	Send note to channel C only
17	Send note to channel A and play the channel B rendezvous note at the same time.
33	Send note to channel A and play the channel C rendezvous note at the same time.
10	Send note to channel B and play the channel A rendezvous note at the same time.
34	Send note to channel B and play the channel C rendezvous note at the same time.
12	Send note to channel C and play the channel A rendezvous note at the same time.
36	Send note to channel C and play the channel B rendezvous note at the same time.
49	Send note to channel A and play the channel B and C rendezvous notes at the same time.
42	Send note to channel B and play the channel A and C rendezvous notes at the same time.
28	Send note to channel C and play the channel A and B rendezvous notes at the same time.

Table 16.2 Rendezvous channel numbers

Notice that the channel numbers of both channel A and B are changed according to Table 16.2 when the notes are to be played simultaneously. Enter and **RUN** the program.

Example 16d

Write a program for the following music with a beat lasting for ½ of a second:



The notes of the treble clef will be played in channel A and for the bass clef in channel B. Suitable program lines for bar 1 would be:

```
10 REM Plays Frere Jacques
```

```
20 SOUND 1,478,50
30 SOUND 1,426,50
40 SOUND 1,379,50
50 SOUND 1,478,50
```

Bar 2 could continue:

```
60 REM Dummy note separates repeated C
70 SOUND 1,2,5
80 SOUND 1,478,50
90 SOUND 1,426,50
100 SOUND 1,379,50
110 SOUND 1,478,50
```

Bar 3 could continue:

```
120 REM Rendezvous note E in channel A with C in channel B
130 SOUND 17,379,50
140 SOUND 10,956,50
150 REM Rendezvous note F in channel A with D in channel B
160 SOUND 17,358,50
170 SOUND 10,851,50
180 REM Rendezvous note G in channel A with E in channel B
190 SOUND 17,319,100
200 SOUND 10,758,50
210 SOUND 2,956,50
```

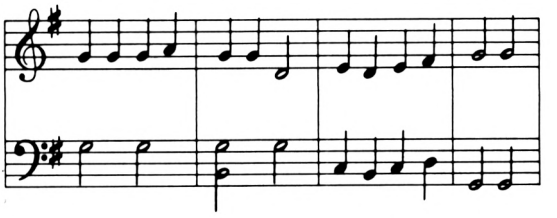
Bar 4 is a repeat of bar 3 except for the first note of channel B which is the dummy separator:

```
220 REM Separator note
230 SOUND 2,2,5
240 REM Rendezvous note E in channel A with C in channel B
250 SOUND 17,379,50
260 SOUND 10,956,50
270 REM Rendezvous note F in channel A with D in channel B
280 SOUND 17,358,50
290 SOUND 10,851,50
300 REM Rendezvous note G in channel A with E in channel B
310 SOUND 17,319,100
320 SOUND 10,758,50
330 SOUND 2,956,50
```

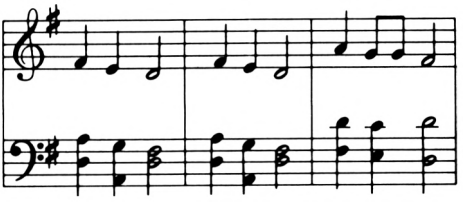
Enter and RUN the program.

Tutorial

16.9) Write a program to play the following tune:



16.10) Write a program to play the following tune:



Solutions to the tutorials

16.1)
```
10 REM Plays D,F#,A in octave 0 and D in octave 1
20 SOUND 1,426
30 SOUND 1,338
40 SOUND 1,284
50 SOUND 1,213
```

16.2)
```
10 REM Plays A,E,C# in octave 1 and A in octave 0
20 SOUND 1,142
30 SOUND 1,190
40 SOUND 1,225
50 SOUND 1,284
```

16.3) The notes are
F octave 0
B octave 0
E octave 1
G octave 0
E octave 0
D octave 1
C octave 0 (middle C)

16.4)
```
10 REM Tutorial 16.4
```

```
20 SOUND 1,358
30 SOUND 1,284
40 SOUND 1,239
50 SOUND 1,179
```

16.5)
```
10 REM Scale of C
20 SOUND 1,478,100
30 SOUND 1,426,100
40 SOUND 1,379,100
50 SOUND 1,358,100
60 SOUND 1,319,100
70 SOUND 1,284,100
80 SOUND 1,253,100
90 SOUND 1,239,100
```

16.6)
```
10 REM Wee Willie Winkie
20 SOUND 1,239,50
30 SOUND 1,319,25
40 SOUND 1,284,25
50 SOUND 1,319,50
60 SOUND 1,379,50
70 SOUND 1,358,50
80 SOUND 1,426,25
90 SOUND 1,379,25
100 SOUND 1,426,100
```

16.7)
```
10 REM Sing a Song of Sixpence
20 SOUND 1,239,20
30 SOUND 1,253,20
40 SOUND 1,284,20
50 SOUND 1,319,20
60 SOUND 1,239,40
70 SOUND 1,379,40
80 SOUND 1,319,20
90 SOUND 1,284,20
100 SOUND 1,319,20
110 SOUND 1,379,20
120 SOUND 1,319,80
```

16.8)
```
10 REM Three Blind Mice
20 SOUND 1,338,50
30 SOUND 1,379,50
40 SOUND 1,426,100
```

```
50 SOUND 1,338,50
60 SOUND 1,379,50
70 SOUND 1,426,100
80 SOUND 1,284,50
90 SOUND 1,319,25
100 SOUND 1,2,5
110 SOUND 1,319,25
120 SOUND 1,338,100
```

16.9)
```
10 REM Plays Good King Wenceslas
20 SOUND 17,319,50
30 SOUND 10,638,100
40 SOUND 1,2,5
50 SOUND 1,319,50
60 SOUND 1,2,5
70 SOUND 17,319,50
80 SOUND 10,638,100
90 SOUND 1,568,50
100 SOUND 1,2,5
110 SOUND 49,319,50
120 SOUND 42,638,100
130 SOUND 28,1012,100
140 SOUND 1,2,5
150 SOUND 1,319,50
160 SOUND 1,2,5
170 SOUND 17,426,100
180 SOUND 10,638,100
190 SOUND 17,379,50
200 SOUND 10,956,50
210 SOUND 17,426,50
220 SOUND 10,1012,50
230 SOUND 17,379,50
240 SOUND 10,956,50
250 SOUND 17,338,50
260 SOUND 10,851,50
270 SOUND 17,319,100
280 SOUND 10,1276,100
290 SOUND 1,2,5
300 SOUND 17,319,100
310 SOUND 10,1276,100
```

16.10)
```
10 REM Plays Three Blind Mice
```

```
20 SOUND 49,338,60
30 SOUND 42,568,60
40 SOUND 28,851,60
50 SOUND 49,379,60
60 SOUND 42,638,60
70 SOUND 28,1136,60
80 SOUND 49,426,120
90 SOUND 42,676,120
100 SOUND 28,851,120
110 SOUND 4,2,5
120 SOUND 49,338,60
130 SOUND 42,568,60
140 SOUND 28,851,60
150 SOUND 49,379,60
160 SOUND 42,638,60
170 SOUND 28,1136,60
180 SOUND 49,426,120
190 SOUND 42,676,120
200 SOUND 28,851,120
210 SOUND 49,284,60
220 SOUND 42,426,60
230 SOUND 28,676,60
240 SOUND 49,319,30
250 SOUND 42,478,60
260 SOUND 28,758,60
270 SOUND 1,2,5
280 SOUND 1,319,30
290 SOUND 49,338,120
300 SOUND 42,426,120
310 SOUND 28,851,120
```

Programs within programs

Consider the program of Example 14b to check a password:

```
 10 REM Comparison of passwords
 20 REM Create a WHILE...WEND loop from which the computer
cannot escape
 30 dummy.test = 0
 40 WHILE dummy.test = 0
 50   REM Clear the screen
 60   CLS
 70   password$ = "Fido"
 80   REM INPUT the password
 90   INPUT "What is the password? ",answer$
100   IF answer$ = password$ THEN PRINT "ENTER" ELSE PRINT
     "NO ENTRY"
110   REM Create a three second delay
120   end.time = TIME + 900
130   WHILE TIME < end.time
140   WEND
150 WEND
```

Instead of displaying ENTER, at line 100, for the correct password you may wish to clear the screen and display at the centre of the screen in bright cyan:

```
Please wait. Enter when the door is fully open.
```

Also for a wrong password you may wish to display at the centre of the screen in bright red on a bright yellow background:

```
NO ENTRY
```

This would require line 100 to become:

```
100 IF answer$ = password$ THEN CLS
                               REM Select bright cyan pen
                               PEN 2
                               LOCATE 14,10
                               PRINT "Please wait."
                               LOCATE 3,12
                               PRINT "Enter when the door
                               is fully open."
                               REM Select bright yellow pen
                               PEN 1
                          ELSE REM Select bright red pen
                               PEN 3
                               REM Select bright yellow
                               background
                               PAPER 1
                               CLS
                               LOCATE 17,12
                               PRINT "NO ENTRY"
                               REM Select blue background
                               and bright yellow pen
                               PAPER 0
                               PEN 1
```

Unfortunately with BASIC it is not possible to have several statements after a THEN or an ELSE. However the program could be written:

```
 10
 20
  .
  .
 90
100 IF answer$ = password$ THEN execute lines 300 to 380
                           and then continue onto line 110
                           ELSE execute lines 500 to 590
                           and then continue onto line 110
110
  .
  .
  .
150
300 CLS
310 REM Select bright cyan pen
```

```
320 PEN 2
330 LOCATE 14,10
340 PRINT "Please wait."
350 LOCATE 3,12
360 PRINT "Enter when the door is fully open."
370 REM Select bright yellow pen
380 PEN 1
500 REM Select right red pen
510 PEN 3
520 REM Select bright yellow background
530 PAPER 1
540 CLS
550 LOCATE 17,12
560 PRINT "NO ENTRY"
570 REM Select blue background and bright yellow pen
580 PAPER 0
590 PEN 1
```

The nine statements that have to be executed if the condition at line 100 is true have been given line numbers 300 to 380. Also the ten statements that have to be executed if the condition at line 100 is false have been given line numbers 500 to 590. Lines 300 to 380 and 500 to 590 are called *subroutines*. The lines 10 to 150 are called the *main program*. The main program must always come before the subroutines so that its line numbers must always be smaller than those of the subroutines.

The BASIC statement that instructs the computer to stop executing the program and start executing a subroutine is:

```
GOSUB ...
```

where the . . . is the line number of the start of the subroutine and is separated from the GOSUB by a space. Hence line 100 becomes:

```
100  IF answer$ = password$ THEN GOSUB 300 ELSE GOSUB 500
```

The BASIC statement that instructs the computer that it has reached the end of the subroutine and that it should go back to the line after the GOSUB statement is:

```
RETURN
```

The following lines would thus be required at the end of the subroutines:

```
390 RETURN
600 RETURN
```

These lines would instruct the computer to go back to the line after the **GOSUB** statement at line 100, that is, to line 110.

The computer must also be instructed when it has completed the main program by inserting an **END** statement after its last line as follows:

```
160 END
```

The complete program becomes:

```
 10 REM Comparison of passwords
 20 REM Create a WHILE...WEND loop from which the computer
cannot escape
 30 dummy.test = 0
 40 WHILE dummy.test = 0
 50   REM Clears the screen
 60   CLS
 70   password$ = "Fido"
 80   REM INPUT the password
 90   INPUT "What is the password? ",answer$
100   IF answer$ = password$ THEN GOSUB 300 ELSE GOSUB 500
110   REM Create a three second delay
120   end.time = TIME + 900
130   WHILE TIME < end.time
140   WEND
150 WEND
160 END
300 CLS
310 REM Select bright cyan pen
320 PEN 2
330 LOCATE 14,10
340 PRINT "Please wait."
350 LOCATE 3,12
360 PRINT "Enter when the door is fully open."
370 REM Select bright yellow pen
380 PEN 1
390 RETURN
500 REM Select bright red pen
510 PEN 3
520 REM Select bright yellow background
```

```
530 PAPER 1
540 CLS
550 LOCATE 17,12
560 PRINT "NO ENTRY"
570 REM Select blue background and bright yellow pen
580 PAPER 0
590 PEN 1
600 RETURN
```

Example 17a

Modify the program of Tutorial 14.4 so that the fine is waived if it is less than 6 pence.

A possible program could be:

```
 10 REM Calculates the fine on an overdue library book
 20 REM Clear the screen
 30 CLS
 40 REM Fines are charged at 2p per day after three days
 50 charge.per.day = 2
 60 REM The memory cell labelled counter stores the number of
overdue library books that have been returned
 70 counter = 0
 80 REM Create a WHILE...WEND loop from which the computer
cannot escape
 90 dummy.test = 0
100 WHILE dummy.test = 0
110   REM INPUT the days overdue
120   LOCATE 1,10
130   INPUT "How many days is the book overdue? ",
      days.overdue
140   IF days.overdue < 3 THEN GOSUB 500 ELSE GOSUB 300
150   REM Create a four second delay
160   end.time = TIME + 1200
170   WHILE TIME < end.time
180   WEND
190   REM Clear the screen
200   CLS
210   REM Increase the overdue book count by one
220   counter = counter + 1
230   REM Display the number of overdue books returned
240   LOCATE 2,8
250   PRINT counter;"overdue books have been returned."
260 WEND
270 END
```

```
300 REM Subroutine to calculate and display the fine
310 fine = days.overdue * charge.per.day
320 REM Display the fine
330 LOCATE 3,12
340 PRINT "The amount to be paid is";
350 REM Select bright red characters
360 PEN 3
370 PRINT fine;
380 REM Select bright yellow characters
390 PEN 1
400 PRINT "pence."
410 RETURN
500 REM Subroutine to display no fine
510 LOCATE 3,12
520 PRINT "There is no fine on this book."
530 RETURN
```

Tutorial

17.1) Modify the program of Tutorial 13.6 so that when you try to overdraw from your account it displays:

```
You do not have enough money for this transaction.
You have only ... pounds in the bank.
```

It is possible for the same subroutine to be called from different places in the main program as shown in the following example.

Example 17b

Modify the program of Example 15a so that the time delays form a subroutine.

A possible program could be:

```
 10 REM Man doing left leg swings
 20 REM Select mode 0
 30 MODE 0
 40 REM Create a WHILE...WEND loop from which the computer
cannot escape
 50 dummy.test = 0
 60 WHILE dummy.test = 0
 70   REM Display building block 248 at screen
```

```
      position 8,10
  80  LOCATE 8,10
  90  PRINT CHR$(248)
 100  REM Call the time delay subroutine
 110  GOSUB 300
 120  REM Display building block 250 at screen
      position 8,10
 130  LOCATE 8,10
 140  PRINT CHR$(250)
 150  REM Call the time delay subroutine
 160  GOSUB 300
 170 WEND
 180 END
 300 REM This subroutine create a delay of 1/4 second
 310 end.time = TIME + 75
 320 WHILE TIME < end.time
 330 WEND
 340 RETURN
```

Enter and **RUN** the program. At line 110 the computer is instructed to execute the subroutine starting at line 300. At line 340 the **RETURN** statement instructs it to go back to the line after the **GOSUB** statement, that is, line 120. A similar procedure occurs at line 160 but this time the **RETURN** at line 340 instructs the computer to go back to line 170 as the **GOSUB** statement that sent it to the subroutine is at line 160.

Tutorial

17.2) Rewrite the solution to Tutorial 15.2 using subroutines.

17.3) Modify the program of Tutorial 17.2 so that there are four men doing the exercise vertically below each other.

Subroutines can be very useful in writing a long program as they assist you to subdivide it into sections. The program for each section can then be written independently of the others, giving you less to think about at any one time. This use of subroutines will be demonstrated in the next chapter.

Solutions to the tutorials

17.1) Change line 210 of the program for Tutorial 13.6 to:

```
210 IF money.left < 0 THEN GOSUB 300 ELSE money =
money.left
```

Add the following lines:

```
240 END
300 REM This subroutine indicates that you are trying to
overdraw your account
310 PRINT "You do not have enough money for this"
320 PRINT "transaction."
330 PRINT "You have only";money;"pounds in the bank."
340 RETURN
```

17.2)

```
 10 REM Man doing left and right leg swings
 20 REM Select mode 0
 30 MODE 0
 40 REM Create a WHILE...WEND loop from which the computer
cannot escape
 50 dummy.test = 0
 60 WHILE dummy.test = 0
 70   REM Create a loop to display the exercise
 80   REM The memory cell labelled stage stores
      the current part of the exercise
 90   stage = 0
100   WHILE stage < 4
110     REM Draw the appropriate stage of the exercise
120     GOSUB 300
130     REM Wait for 1/4 second
140     GOSUB 500
150     REM Change stage for the next loop
160     stage = stage + 1
170   WEND
180 WEND
190 END
300 REM This subroutine draws the appropriate stage of the
exercise
310 REM The memory cell labelled block stores
the building block number to be displayed
320 block = 248
330 IF stage = 1 THEN block = 250
340 IF stage = 3 THEN block = 251
350 LOCATE 8,10
360 PRINT CHR$(block)
370 RETURN
500 REM This subroutine creates a delay of 1/4 second
510 end.time = TIME + 75
520 WHILE TIME < end.time
```

```
530 WEND
540 RETURN
```

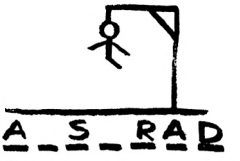
17.3) Add the following lines to the program of Tutorial 17.2:

```
341 REM Create a loop to draw the four men
342 REM The memory cell labelled row stores the screen row of
the man
343 row = 3
344 WHILE row <= 21
361   REM Change row for the next loop
362   row = row + 6
363 WEND
```

Also change line 350 to:

```
350 LOCATE 8,row
```

Hangman game



Program description

In this chapter we shall design a program that plays a hangman game of guess the number. First it is necessary to decide exactly what we want the program to do. It should:

1) Display an introductory screen for five seconds.
2) Play the game where it:
   a) chooses a random number from 1 to 99
   b) displays a hangman frame
   c) repeats the next three steps until you have had six guesses or you have guessed correctly:
      i) asks you to guess the number
      ii) indicates the maximum and minimum possible values of your guess
      iii) draws the next part of the man if the guess is wrong
   d) either plays 'Oh dear! What can the matter be?', hangs the man and displays the answer if you are not correct after six attempts
   or plays a dance and releases the man if you are correct.
3) Ask if you wish to play again.
4) Either play again
   or display a goodbye message.

Structure of the program

This program is too complicated to think about all of it at once. The best technique is to divide the above program description into sections that can be written independently of each other. Each of these sections can be further subdivided so that the task becomes one of writing several short routines. The complete program is obtained by finally fitting these together.

The above program description divides naturally into four parts, each of which will become a subroutine:

1) Introductory screen.
2) Playing the game with appropriate end of game display.
3) Ask if you wish to play again.
4) Goodbye message.

A `WHILE...WEND` loop will be used to give you the option of playing again. It is also wise to do a software reset at the beginning of the program as this puts the computer in a known condition independent of what it was doing previously. The structure of the program will be:

```
 10 REM Hangman game
 20 REM Do a software reset
 30 CALL -17409:CALL -17586
 40 REM Play the game if the memory cell labelled
play.again$ contains "YES"
 50 play.again$ = "YES"
 60 WHILE play.again$ = "YES"
 70   REM Introductory screen
 80   GOSUB 1000
 90   REM Plays the game with appropriate end of game
      display
100   GOSUB 3000
110   REM Asks if you wish to play again
120   GOSUB 8000
130 WEND
140 REM Goodbye message
150 GOSUB 9000
160 REM Hide the cursor
170 PEN 0
180 END
```

The subroutines at lines 1000, 3000, 8000 and 9000 will now be developed.

Introductory screen

Assume that the mode 0 introductory screen, shown in Fig. 18.1, is required. The writing should be in black with the stars

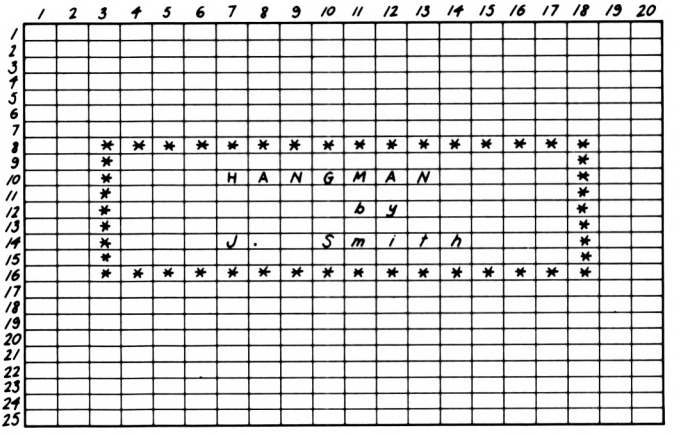


Figure 18.1 Hangman introductory screen

in bright red. The background of the stars and the surrounding screen should be blue whereas the background within the stars should be bright yellow. This screen display is similar to that of Tutorial 7.8. The subroutine starting at line 1000 could be:

```
1000 REM Introductory screen display
1010 REM Select mode 0
1020 MODE 0
1030 REM Draw the * signs first in bright red
1040 PEN 3
1050 LOCATE 3,8
1060 PRINT "*****************"
1070 LOCATE 3,9
1080 PRINT "*               *"
1090 LOCATE 3,10
1100 PRINT "*               *"
1110 LOCATE 3,11
1120 PRINT "*               *"
1130 LOCATE 3,12
1140 PRINT "*               *"
1150 LOCATE 3,13
1160 PRINT "*               *"
1170 LOCATE 3,14
1180 PRINT "*               *"
1190 LOCATE 3,15
1200 PRINT "*               *"
1210 LOCATE 3,16
1220 PRINT "*****************"
```

```
1230 REM Display the text
1240 REM Select bright yellow background and black text
1250 PAPER 1
1260 PEN 5
1270 LOCATE 4,9
1280 PRINT "              "
1290 LOCATE 4,10
1300 PRINT "  HANGMAN     "
1310 LOCATE 4,11
1320 PRINT "              "
1330 LOCATE 4,12
1340 PRINT "      by      "
1350 LOCATE 4,13
1360 PRINT "              "
1370 LOCATE 4,14
1380 PRINT "  J. Smith    "
1390 LOCATE 4,15
1400 PRINT "              "
1410 REM Select normal colours
1420 PAPER 0
1430 PEN 1
1440 REM Create a five second delay
1450 end.time = TIME + 1500
1460 WHILE TIME < end.time
1470 WEND
1480 REM Clear the screen at the end of the subroutine
1490 CLS
1500 RETURN
```

This subroutine can now be tested before continuing. As the subroutines called from lines 100, 120 and 150 have not been written insert a line 85 that stops the program:

```
85 END
```

Enter and RUN the program. The computer should produce the desired screen display for five seconds before it clears the screen and displays the usual Ready message. Delete line 85 and SAVE the program onto a disk calling it TEST1.

The subroutine that plays the game

Let's assume that the screen display for the game is to be in mode 0 as shown in Fig. 18.2.

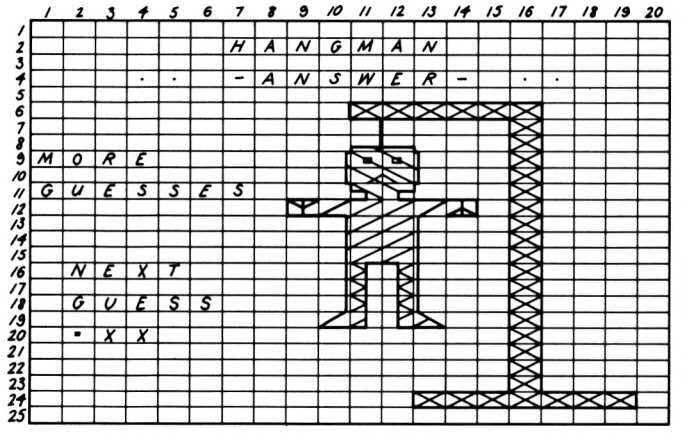
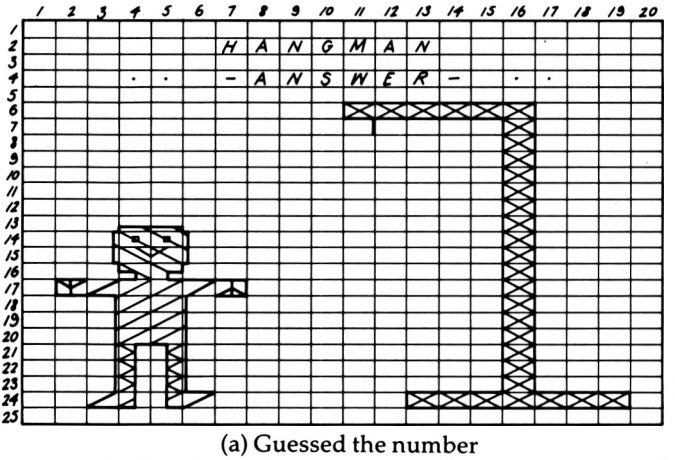
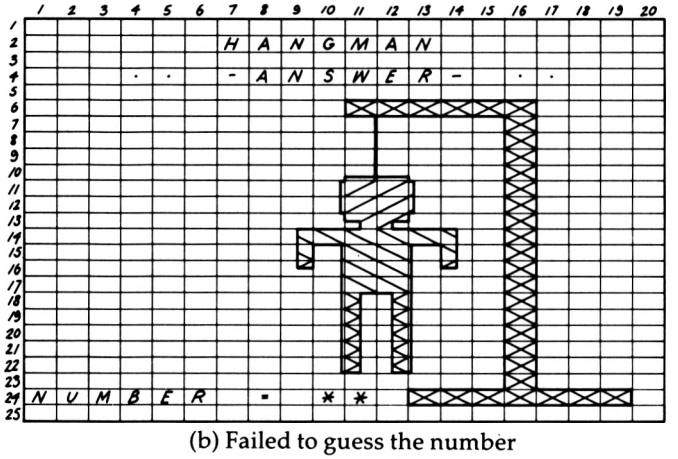


Figure 18.2 Hangman game screen



(a) Guessed the number



(b) Failed to guess the number

Figure 18.3 End of game display

This subroutine has to:

1) Choose a random number from 1 to 99.
2) Display the hangman frame in bright yellow and the title in bright white.
3) Create a loop so that the program gives you six attempts to guess the number. This can be achieved as in Example 13b. However in this case the maximum and minimum possible values of the guess are also to be displayed.
4) Produce an end-of-game display with the appropriate music.

Let's assume that the end-of-game displays shown in Fig. 18.3 are required:

The program for this subroutine could be:

```
3000 REM Plays the game
3010 REM Part 1 of the subroutine
3020 REM Vary the random number sequence
3030 RANDOMIZE TIME
3040 REM Create a random number from 1 to 99
3050 number = INT(RND*99)+1
3060 REM Part 2 of the subroutine
3070 REM Draw the top of the hangman frame
3080 LOCATE 11,6
3090 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(143)
3100 REM Draw the post of the hangman frame
3110 counter = 7
3120 WHILE counter < 24
3130   LOCATE 16,counter
3140   PRINT CHR$(143)
3150   counter = counter + 1
3160 WEND
3170 REM Draw the base of the hangman frame
3180 LOCATE 13,24
3190 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(143);CHR$(143)
3200 REM Display the title in bright white
3210 PEN 4
3220 LOCATE 7,2
3230 PRINT "HANGMAN"
3240 REM Select bright yellow characters
```

```
3250 PEN 1
3260 REM Part 3 of the subroutine
3270 REM Create a WHILE...WEND loop that gives you six
attempts to guess the number
3280 REM The memory cell labelled guesses.left stores the
number of guesses the person has left
3290 guesses.left = 6
3300 REM The memory cell labelled max.value stores the
maximum possible value of your guess
3310 REM The memory cell labelled min.value stores the
minimum possible value of your guess
3320 max.value = 99
3330 min.value = 1
3340 REM If the memory cell labelled repeat.loop$ contains
"NO" then escape from the loop
3350 REM If the memory cell labelled repeat.loop$ contains
"YES" then execute the loop
3360 repeat.loop$ = "YES"
3370 WHILE repeat.loop$ = "YES"
3380   REM Make the contents of the loop into a subroutine
3390   GOSUB 3600
3400   REM Update the number of guesses left
       and repeat.loop$
3410   guesses.left = guesses.left - 1
3420   IF guesses.left = 0 THEN repeat.loop$ = "NO"
3430   IF guess = number THEN repeat.loop$ = "NO"
3440 WEND
3450 REM Part 4 of the subroutine
3460 REM End of game display and music
3470 REM Delete the man and the text at the left of the
screen
3480 row = 7
3490 WHILE row <= 21
3500   LOCATE 1,row
3510   PRINT "              "
3520   row = row + 1
3530 WEND
3540 IF guess = number THEN GOSUB 5300 ELSE GOSUB 7000
3550 RETURN
```

The contents of the loop in part 3 and the two end-of-game displays in part 4 of this subroutine are quite complicated. Hence they have been placed in subroutines starting at lines 3600, 5300 and 7000 so that they can be considered individually

after this subroutine has been tested. Add the lines 3000 to 3550 to the end of the program called TEST1. The part of the program that has been written will now be tested.

As the main function of the subroutine called from line 3390 will be to INPUT a guess, replace this line with:

```
3390 INPUT "Guess = ",guess
```

As the subroutines at lines 5300 and 7000 have not been written delete line 3540. Also as the subroutines at lines 8000 and 9000 have not been written insert a line 105 that stops the program:

```
105 END
```

RUN the program. The initial screen display should be followed by the display for playing the game. This should display the title and the frame and then ask you to make a guess. If after six guesses you are not correct or if you guess correctly the program should stop and the computer will display the usual Ready message.

Remove line 105 and retype lines 3390 and 3540. Save the program onto a disk calling it TEST2.

The subroutines that start at lines 3600, 5300 and 7000 will now be developed.

The subroutine starting at line 3600 and called from line 3390

This subroutine will:

1) Display the maximum and minimum possible values of your guess.
2) Ask you to guess the number.
3) Update the maximum and minimum possible values of your guess.
4) Draw the next part of the man if the guess is wrong.

The program for this subroutine could be:

```
3600 REM Display the maximum and minimum values of the
guess
3610 LOCATE 3,4
```

```
3620 PRINT min.value;"-ANSWER-";max.value
3630 REM Display the number of guesses left
3640 LOCATE 1,7
3650 IF guesses.left = 6 THEN PRINT "SIX"
3660 IF guesses.left = 5 THEN PRINT "FIVE"
3670 IF guesses.left = 4 THEN PRINT "FOUR"
3680 IF guesses.left = 3 THEN PRINT "THREE"
3690 IF guesses.left = 2 THEN PRINT "TWO  "
3700 IF guesses.left = 1 THEN PRINT "ONE  "
3710 LOCATE 1,9
3720 PRINT "MORE"
3730 LOCATE 1,11
3740 PRINT "GUESSES"
3750 REM Display the words NEXT GUESS
3760 LOCATE 2,16
3770 PRINT "NEXT"
3780 LOCATE 2,18
3790 PRINT "GUESS"
3800 REM Delete the last guess
3810 LOCATE 2,20
3820 PRINT "    "
3830 REM INPUT the next guess
3840 LOCATE 2,20
3850 INPUT "=",guess
3860 REM Update the contents of the memory cells labelled
min.value and max.value
3870 IF guess > number THEN GOSUB 3950
3880 IF guess < number THEN GOSUB 4000
3890 REM Draw the next part of the man if the guess is
wrong
3900 IF guess <> number THEN GOSUB 4100
3910 RETURN
```

Notice again that complicated tasks of updating the maximum and minimum value of the guess and drawing the next part of the man have been placed into subroutines so that they can be considered separately.

Updating the maximum possible value of the guess

The subroutine to update the maximum possible value of your guess starts at line 3950. A possible subroutine could be:

```
3950 REM Update the contents of the memory cell labelled
```

```
max.value
3960 IF guess < max.value THEN max.value = guess
3970 RETURN
```

Updating the minimum possible value of the guess

The subroutine to update the minimum possible value of the guess starts at line 4000. A possible subroutine could be:

```
4000 REM Update the contents of the memory cell labelled
min.value
4010 IF guess > min.value THEN min.value = guess
4020 RETURN
```

Drawing the man

This subroutine starts at line 4100 and depending on the number of guesses left calls another subroutine that draws the appropriate part of the man.

```
4100 REM Draws the man
4110 REM Draw the head
4120 IF guesses.left = 6 THEN GOSUB 4300
4130 REM Draw the body
4140 IF guesses.left = 5 THEN GOSUB 4650
4150 REM Draw the left arm
4160 IF guesses.left = 4 THEN GOSUB 4750
4170 REM Draw the right arm
4180 IF guesses.left = 3 THEN GOSUB 4850
4190 REM Draw the left leg
4200 IF guesses.left = 2 THEN GOSUB 4950
4210 REM Draw the right leg
4220 IF guesses.left = 1 THEN GOSUB 5100
4230 RETURN


4300 REM Draws the rope and head
4310 REM Select black characters
4320 PEN 5
4330 REM Draw the rope
4340 LOCATE 11,7
4350 PRINT CHR$(209)
4360 LOCATE 11,8
4370 PRINT CHR$(209)
```

```
4380 REM Select pink characters
4390 PEN 11
4400 REM Draws the face
4410 LOCATE 10,9
4420 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
4430 LOCATE 10,10
4440 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
4450 LOCATE 11,11
4460 PRINT CHR$(139);CHR$(135)
4470 REM Transparent printing
4480 PRINT CHR$(22)+CHR$(1)
4490 LOCATE 11,8
4500 PRINT CHR$(140);CHR$(140)
4510 REM Select black characters
4520 PEN 5
4530 REM Draws the eyes and sad mouth
4540 LOCATE 11,9
4550 PRINT CHR$(144);CHR$(144)
4560 LOCATE 11,10
4570 PRINT CHR$(194);CHR$(195)
4580 REM Normal printing
4590 PRINT CHR$(22)+CHR$(0)
4600 REM Select bright yellow characters
4610 PEN 1
4620 RETURN


4630 REM Draws the body
4640 LOCATE 10,12
4670 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
4680 LOCATE 10,13
4690 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
4700 LOCATE 10,14
4710 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
4720 LOCATE 10,15
4730 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
4740 RETURN


4750 REM Draws the left arm
4760 LOCATE 13,12
4770 PRINT CHR$(143);
4780 REM Select pink background to draw the hand
4790 PAPER 11
```

```
4800 PRINT CHR$(133)
4810 REM Select blue background
4820 PAPER 0
4830 RETURN


4850 REM Draws the right arm
4860 LOCATE 9,12
4870 REM Select pink background to draw the hand
4880 PAPER 11
4890 PRINT CHR$(138);CHR$(143)
4900 REM Select blue background
4910 PAPER 0
4920 RETURN


4950 REM Draws the left leg
4960 REM Select bright green characters for the trousers
4970 PEN 12
4980 LOCATE 12,16
4990 PRINT CHR$(138);CHR$(211)
5000 LOCATE 12,17
5010 PRINT CHR$(138);CHR$(211)
5020 LOCATE 12,18
5030 PRINT CHR$(138);CHR$(211)
5040 REM Select bright red characters for the shoe
5050 PEN 3
5060 LOCATE 12,19
5070 PRINT CHR$(138);CHR$(143)
5080 PEN 1
5090 RETURN


5100 REM Draws the right leg
5110 REM Select bright green characters for the trousers
5120 PEN 12
5130 LOCATE 10,16
5140 PRINT CHR$(209);CHR$(133)
5150 LOCATE 10,17
5160 PRINT CHR$(209);CHR$(133)
5170 LOCATE 10,18
5180 PRINT CHR$(209);CHR$(133)
5190 REM Select bright red characters for the shoe
5200 PEN 3
```

```
5210 LOCATE 10,19
5220 PRINT CHR$(143);CHR$(133)
5230 PEN 1
5240 REM Create a delay of 1 second
5250 end.time = TIME + 300
5260 WHILE TIME < end.time
5270 WEND
5280 RETURN
```

Add lines 3600 to 5280 to the program called TEST2. The part of the program that has been written will now be tested. As the subroutines at lines 5300, 7000, 8000 and 9000 have not been written delete line 3540 and insert a line 105 that stops the program:

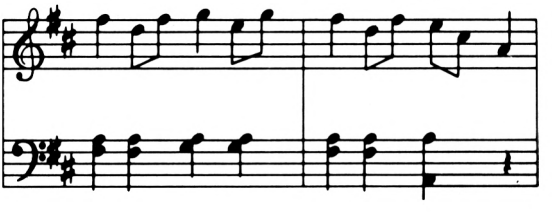
```
105 END
```

RUN the program. With each wrong guess the computer should display another part of the man and update the maximum and minimum possible values of your guess. After six guesses or when you guess correctly the program stops and the computer displays the usual Ready message.

Remove line 105 and retype line 3540. SAVE the program onto a disk calling it TEST3.

The subroutine starting at line 5300 and called from line 3540

This subroutine releases the man and plays:



A possible program for the subroutine could be:

```
5300 REM End of game display and tune for the correct
guess
5310 REM Draws the released man
5320 REM Draws the face
```

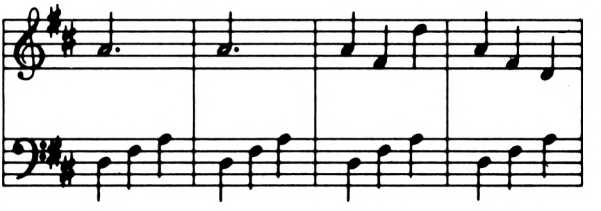
```
5330 REM Select pink characters for the face
5340 PEN 11
5350 LOCATE 4,13
5360 PRINT CHR$(140);CHR$(140)
5370 LOCATE 3,14
5380 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
5390 LOCATE 3,15
5400 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
5410 LOCATE 4,16
5420 PRINT CHR$(139);CHR$(135)
5430 REM Select bright yellow characters for the body
5440 PEN 1
5450 REM Draws the body
5460 LOCATE 2,17
5470 PRINT CHR$(138);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(133)
5480 LOCATE 3,18
5490 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
5500 LOCATE 3,19
5510 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
5520 LOCATE 3,20
5530 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
5540 REM Draws the legs
5550 REM Select bright green characters for the trousers
5560 PEN 12
5570 LOCATE 3,21
5580 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
5590 LOCATE 3,22
5600 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
5610 LOCATE 3,23
5620 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
5630 REM Select bright red characters for the shoes
5640 PEN 3
5650 REM Draws the shoes
5660 LOCATE 3,24
5670 PRINT CHR$(143);CHR$(133);CHR$(138);CHR$(143)
5680 REM Transparent printing
5690 PRINT CHR$(22)+CHR$(1)
5700 REM Select black characters
5710 PEN 5
5720 REM Draws the eyes and happy mouth
5730 LOCATE 4,14
5740 PRINT CHR$(144);CHR$(144)
5750 LOCATE 4,15
```

```
5760 PRINT CHR$(193);CHR$(192)
5770 REM Select pink characters
5780 PEN 11
5790 REM Draws the hands
5800 LOCATE 2,17
5810 PRINT CHR$(133);"    ";CHR$(138)
5820 REM Normal printing
5830 PRINT CHR$(22)+CHR$(0)
5840 REM Select bright yellow characters
5850 PEN 1
5860 REM Plays the dance
5870 REM Bar 1
5880 SOUND 49,169,40
5890 SOUND 42,568,40
5900 SOUND 28,676,40
5910 SOUND 49,213,20
5920 SOUND 2,2,5
5930 SOUND 42,568,40
5940 SOUND 28,676,40
5950 SOUND 1,169,20
5960 SOUND 49,159,40
5970 SOUND 2,2,5
5980 SOUND 42,568,40
5990 SOUND 28,676,40
6000 SOUND 49,190,20
6010 SOUND 2,2,5
6020 SOUND 42,568,40
6030 SOUND 28,638,40
6040 SOUND 1,159,20
6050 REM Bar 2
6060 SOUND 49,169,40
6070 SOUND 2,2,5
6080 SOUND 42,568,40
6090 SOUND 28,676,40
6100 SOUND 49,213,20
6110 SOUND 2,2,5,
6120 SOUND 42,568,40
6130 SOUND 28,676,40
6140 SOUND 1,169,20
6150 SOUND 49,190,20
6160 SOUND 2,2,5
6170 SOUND 42,568,40
6180 SOUND 28,1136,40
6190 SOUND 1,225,20
```

```
6200 SOUND 1,284,40
6210 RETURN
```

The subroutine starting at line 7000 and called from line 3540

This subroutine hangs the man, displays the correct number and plays 'Oh dear! What can the matter be?'. The music for this tune is:



A possible program for this subroutine could be:

```
7000 REM Failed to guess the answer
7010 REM Hangs the man
7020 REM Select black characters
7030 PEN 5
7040 Draws the rope
7050 LOCATE 11,7
7060 PRINT CHR$(209)
7070 LOCATE 11,8
7080 PRINT CHR$(209)
7090 LOCATE 11,9
7100 PRINT CHR$(209)
7110 LOCATE 11,10
7120 PRINT CHR$(209)
7130 REM Select yellow characters to draw the face
7140 PEN 9
7150 REM Draws the face
7160 LOCATE 10,11
7170 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
7180 LOCATE 10,12
7190 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
7200 LOCATE 11,13
7210 PRINT CHR$(139);CHR$(135)
7220 REM Select bright yellow characters to draw the body
7230 PEN 1
7240 REM Draws the body
```

```
7250 LOCATE 9,14
7260 PRINT CHR$(138);CHR$(143);CHR$(143);CHR$(143);
CHR$(143);CHR$(133)
7270 LOCATE 9,15
7280 PRINT CHR$(138);CHR$(209);CHR$(143);CHR$(143);
CHR$(211);CHR$(133)
7290 LOCATE 10,16
7300 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
7310 LOCATE 10,17
7320 PRINT CHR$(209);CHR$(143);CHR$(143);CHR$(211)
7330 REM Select bright green characters to draw the legs
7340 PEN 12
7350 REM Draws the legs
7360 LOCATE 10,18
7370 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
7380 LOCATE 10,19
7390 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
7400 LOCATE 10,20
7410 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
7420 REM Select bright red characters to draw the shoes
7430 PEN 3
7440 REM Draws the shoes
7450 LOCATE 10,21
7460 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
7470 LOCATE 10,22
7480 PRINT CHR$(209);CHR$(133);CHR$(138);CHR$(211)
7490 REM Transparent printing
7500 PRINT CHR$(22)+CHR$(1)
7510 REM Select yellow characters
7520 PEN 9
7530 REM Draws the top of the head
7540 LOCATE 11,10
7550 PRINT CHR$(140);CHR$(140)
7560 REM Draws the hands
7570 LOCATE 9,16
7580 PRINT CHR$(130);"    ";CHR$(129)
7590 REM Normal printing
7600 PRINT CHR$(22)+CHR$(0)
7610 REM Select bright yellow characters
7620 PEN 1
7630 REM Display correct answer
7640 LOCATE 1,24
7650 PRINT "NUMBER =";number
7660 REM Plays 'Oh dear! What can the matter be?'
```

```
7670 REM Bar 1
7680 SOUND 17,284,120
7690 SOUND 10,851,40
7700 SOUND 2,676,40
7710 SOUND 2,568,40
7720 REM Bar 2
7730 SOUND 1,2,5
7740 SOUND 17,284,120
7750 SOUND 10,851,40
7760 SOUND 2,676,40
7770 SOUND 2,568,40
7780 REM Bar 3
7790 SOUND 1,2,5
7800 SOUND 17,284,40
7810 SOUND 10,851,40
7820 SOUND 17,338,40
7830 SOUND 10,676,40
7840 SOUND 17,213,40
7850 SOUND 10,568,40
7860 REM Bar 4
7870 SOUND 17,284,40
7880 SOUND 10,851,40
7890 SOUND 17,338,40
7900 SOUND 10,676,40
7910 SOUND 17,426,40
7920 SOUND 10,568,40
7930 RETURN
```

Add the lines 5300 to 7930 to the program called TEST3. As the subroutines at lines 8000 and 9000 have not been written insert a line 105 that stops the program:

```
105 END
```

RUN the program. The game should play until the man is released or hanged after which the computer should display the usual Ready message. Remove line 105 and SAVE the program onto a disk calling it TEST4.

The subroutine that asks if you wish to play again

Let's assume that you wish this subroutine to display PLAY

AGAIN? superimposed on the middle of the previous screen. The characters should be in black on bright white. A possible program for this subroutine could be:

```
8000 REM PLAY AGAIN? display
8010 REM Select black characters and bright white
background
8020 PAPER 4
8030 PEN 5
8040 LOCATE 4,12
8050 INPUT "PLAY AGAIN? ",answer$
8060 IF answer$ = "NO" THEN play.again$ = "NO"
8070 IF answer$ = "no" THEN play.again$ = "NO"
8080 REM Return to normal colours
8090 PEN 1
8100 PAPER 0
8110 RETURN
```

The subroutine that displays the goodbye message

Let's assume that you wish to clear the screen and display goodbye. A possible program for this subroutine could be:

```
9000 REM Good-bye message
9010 REM Clear the screen
9020 CLS
9030 LOCATE 6,12
9040 PRINT "GOOD-BYE"
9050 RETURN
```

Add the lines 8000 to 9050 to the program TEST4 and RUN it. The program should now be complete.

Chapter Nineteen

What can I do now with my Amstrad CPC 664

Additional software for your Amstrad CPC 664

Throughout this book you have been using the language BASIC to instruct the computer to do simple calculations, draw pictures and play tunes. These programs are called the *software*. Computers have many uses in the modern world, e.g. in education, banking, robotics, artificial intelligence, etc. As a result of this diversification many computer languages have been developed each with instructions that are of use in a particular application. Some of these languages are:

BASIC	a general purpose language for beginners.
COBOL	a business language.
FORTH	used to control instruments such as telescopes.
LOGO	used to teach computing to children
PASCAL	a more advanced general purpose language.

Your Amstrad CPC 664 is capable of supporting languages other than BASIC making it useful in many different fields of computing.

Loading the language LOGO

The systems disk supplied with your Amstrad CPC 664 contains a version of the language LOGO. Switch on the computer and insert side two of your backup system disk into the floppy disk drive. Type:

```
|cpm
```

After a few seconds the computer displays:

```
Welcome to

Amstrad LOGO V1.1
Copyright (c) 1983,Digital Research
Pacific Grove, California

Dr. Logo is a trademark of
Digital Research

Product No. 6002-1232

Please Wait
```

Soon the screen clears and a question mark appears at the top left hand corner. This is the LOGO cursor which tells you that LOGO is waiting for you to type an instruction.

LOGO turtle graphics

Type (noticing the space between `fd` and `150`):

```
fd 150
```

Press an **ENTER** key. An arrow appears on the screen. The head of the arrow consists of a triangle which is called the *turtle*. LOGO graphics consist of creating pictures by moving this turtle round the screen. The screen has the same coordinates as in BASIC graphics and hence the graphics planner of Chapter 8 can be used to plan your designs. The turtle starts in the middle of the screen pointing upwards (often called north). The command:

```
fd 150
```

instructs the turtle to move forward 150. As it is pointing north it moves up the screen 150 points. In LOGO the command `fd` is called a *primitive*.

We shall now investigate some of the primitives available for moving the turtle. Type:

```
rt 40
```

Press an **ENTER** key. This instructs the turtle to turn in a clockwise direction by 40 degrees. Type:

```
rt -80
```

Press an **ENTER** key. This instructs the turtle to turn in an anti-clockwise direction by 80 degrees. Type:

```
bk 50
```

Press an **ENTER** key. This instructs the turtle to move backwards by 50 points. Observe that it still points in the same direction.

The above types of commands can be combined. Type:

```
rt -90 fd 200 rt 130
```

Press an **ENTER** key. You will see that the computer obeys the commands in the order `rt-90`, `fd 200` and then `rt 130`.

Sometimes you may wish to move the turtle without leaving a trace on the screen. The pen up (`pu`) primitive lifts the turtle's pen preventing it from leaving a trace. The pen can be lowered again with the pen down (`pd`) primitive. Type:

```
pu fd 200 rt 90 pd fd 300
```

Press an **ENTER** key. Again the turtle obeys the commands in order. It lifts its pen and moves 200 points forward. It then turns clockwise through 90 degrees, lowers its pen and moves forward 300 points leaving a trace.

As the screen display is mode 1, four colours are available. The colour of the trace can be changed by the primitive set pen colour (`setpc`). Type:

```
setpc 3
```

Press an **ENTER** key. The turtle has now changed its pen to ink 3 which is bright red. Type:

```
rt 90 fd 100
```

Press an **ENTER** key. The turtle's trace is now bright red.

The primitive clear screen (`cs`) can be used to return the turtle to its initial position. Type:

```
cs
```

Press an **ENTER** key. Notice that the `cs` primitive does not change the colour of the trace.

Using LOGO to draw pictures

A rectangle could be drawn on the screen by combining the primitives as follows:

```
fd 100 rt 90 fd 200 rt 90 fd 100 rt 90 fd 200 rt 90
```

This can be defined as a LOGO word called `rectangle` by typing (remembering to press an **ENTER** key at the end of each complete line):

```
to rectangle
fd 100 rt 90 fd 200 rt 90 fd 100 rt 90 fd 200 rt 90
end
```

LOGO now understands the word `rectangle` and can obey it just as if it were another primitive. Clear the screen using the `cs` primitive and then type:

```
rectangle
```

Press an **ENTER** key and a rectangle should be drawn on the screen.

Two rectangles could be drawn by typing:

```
cs rectangle pu bk 120 pd rectangle
```

Press an **ENTER** key. Two rectangles should be drawn on the screen.

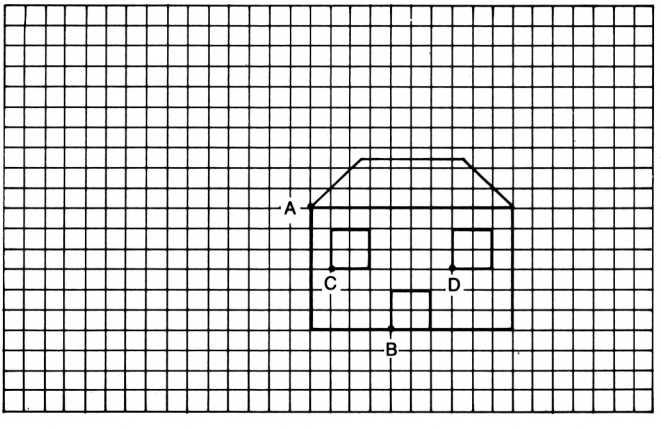


Figure 19.1 Outline of a house

Example 19a

Use LOGO to draw a house similar to that of Example 8d as shown in Fig. 19.1.

The house could be constructed by defining words for the walls, roof, doors and windows. The wall could be drawn by the word:

```
to wall
setpc 1 rt 90 fd 200 rt 90 fd 120 rt 90 fd 200 rt 90
fd 120
end
```

One advantage of LOGO is that the word `wall` can now be tested. Type:

```
cs wall
```

Press an **ENTER** key and the walls should be drawn on the screen.

The roof could now be drawn by the word:

```
to roof
setpc 3 rt 45 fd 70 rt 45 fd 100 rt 45 fd 70 rt 135 fd 200
rt 90
end
```

This definition can be tested by typing:

```
roof
```

Press an **ENTER** key and the roof should be drawn.

We shall now define the word `frame` to be the outline of the house. Type:

```
to frame
cs wall roof
end
```

Test `frame` by typing:

```
frame
```

Press an **ENTER** key. The outline of the house should now be drawn.

Before the door can be drawn the turtle must be moved to the point B. Define the word `moveAtoB` as:

```
to moveAtoB
pu bk 120 rt 90 fd 80 rt -90 pd
end
```

Test this word by typing:

```
moveAtoB
```

Press an **ENTER** key. The turtle should move to B and point north.

The door could be drawn by the word:

```
to door
setpc 3 fd 70 rt 90 fd 40 rt 90 fd 70 rt 90 fd 40
rt 90
end
```

Test this word by typing:

```
door
```

Press an **ENTER** key. The door should now be added to the house.

Before the windows can be drawn the turtle must be moved to the point C. Define the word `moveBtoC` as:

```
to moveBtoC
pu rt -90 fd 60 rt 90 fd 60 pd
end
```

Test this word by typing:

```
moveBtoC
```

Press an **ENTER** key. The turtle should move to C and point north.

The window could be drawn by the word:

```
to wind
setpc 2 fd 40 rt 90 fd 40 rt 90 fd 40 rt 90 fd 40 rt 90
end
```

Test this word by typing:

```
wind
```

Press an **ENTER** key. The left hand window should appear.

The turtle can now be moved to D by the word `moveCtoD`:

```
to moveCtoD
pu rt 90 fd 120 rt -90 pd
end
```

Test this word by typing:

```
moveCtoD
```

Press an **ENTER** key. The turtle should now move to D and point north. Type:

```
wind
```

Press an **ENTER** key and the other window should be drawn.

The turtle can be hidden by the primitive `ht`. Type:

```
ht
```

Press an **ENTER** key and the turtle should disappear.

The complete house could be drawn by the word `house` as follows:

```
to house
frame moveAtoB door moveBtoC wind moveCtoD wind ht
end
```

Test this by typing:

```
house
```

Press an **ENTER** key. The house should be drawn. The turtle can be displayed by typing:

```
st
```

Press an **ENTER** key.

Tutorial

19.1) Write a LOGO program to draw the arc of a circle A to B shown in Fig. 19.2.

19.2) Write a LOGO program to draw the flower shown in Fig. 19.2.

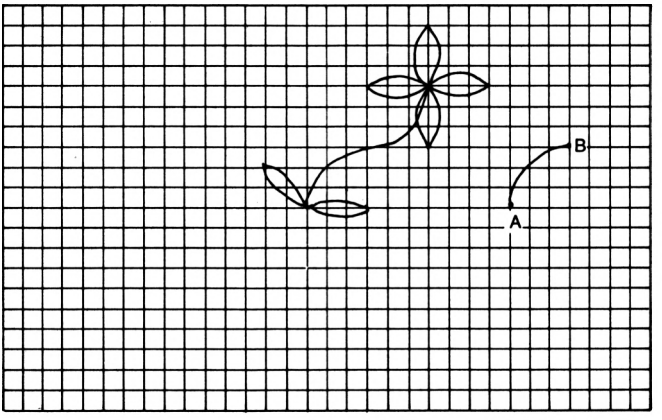


Figure 19.2 Picture of a flower

Comparison of LOGO and BASIC programs

In BASIC the computer knows how to obey a number of commands and it cannot be taught any new ones. The program is created by instructing the computer to obey the known commands in a given sequence. This type of program is difficult to test until it is complete. In LOGO the computer starts only knowing how to execute a limited number of primitives but can be taught others that are combinations of these. This forces the programmer to think more logically and has the advantage that each new word can be tested as it is created as shown in Example 19a. These features have made it very useful in the teaching of young children.

The above discussion of LOGO should only be regarded as an introduction to the language. It has not mentioned all the primitives associated with graphics nor any of those relating to text processing nor logic, etc. These other commands make LOGO useful in teaching the development of language and logical thinking.

Commercial software

It is not always necessary to write your own software as you can often buy a program that performs the same task. This is called a software package. In general if a suitable software

package is available then it is better to buy it than to write your own as this could save you hundreds of hours of work. CP/M, which stands for control program for microprocessors, is an operating system in which many business packages have been written. Side one of your system disk allows the Amstrad CPC 664 to run many of these business packages.

A wordprocessor is one of these commercial packages that allows you to manipulate words and sentences. Let's compare letter writing with and without a wordprocessor. Without a wordprocessor you would first write the letter on a piece of paper and then alter words, phrases, etc. until you felt that it conveyed the desired sentiments and message. Then you would type it but if you made a mistake you would either accept a messy page or type it again. Also if you required more than one top copy you would have to type it several times.

Now let's consider the same task with a wordprocessor program in your computer. You would type the first draft straight into the computer avoiding the need to write it on paper. Once typed you would edit it by deleting words, phrases, swapping paragraphs and correcting spelling. You could insert all or part of a previous document into your letter without retyping. You would then preview the text on the screen and adjust the layout until satisfactory. The letter could then be printed and saved onto tape or disk. If several people were to receive the letter then each could be sent his own personalised top copy rather than a photocopy with his name added later.

The computer's memory

The computer's memory consists of two parts:

a) ROM (Read Only Memory).
b) RAM (Random Access Memory).

The ROM stores the language BASIC. This type of memory cannot be altered by you and the information is not forgotten when the power is switched off.

The RAM is the honeycomb of memory cells introduced in Chapter 9. These memory cells are used to store both the program and the contents of your numeric and string vari-

ables. Hence if the program is long then fewer memory cells will be available to store your variables. Let's assume that you have loaded the wordprocessor program. As this is a very complicated program it will be quite long and so there will not be many memory cells left to store your variables which in this case will be your text. One solution is to store the program in ROM so that it does not use up any of the RAM leaving all the memory cells for your text. When a program is stored into ROM it is called *firmware*. Your Amstrad CPC 664 has been designed to accept extra firmware programs. This also has the advantage that you do not have to load them as they are permanently in the computer.

Additional hardware for your Amstrad CPC 664

The parts of the computer that you can touch and see, e.g. the keyboard, the floppy disk drive and the monitor, are collectively called *hardware*. Your Amstrad CPC 664 can be made more versatile by connecting additional hardware, e.g. a printer.

Printers

Many people regard a printer as the most useful hardware addition to their computer as it allows them to obtain a paper copy of their programs and other screen displays. There are two common types of printers:

a) Daisy wheel printer.
b) Dot matrix printer.

A daisy wheel printer can be thought of as a typewriter that is controlled by the computer. It has the advantage of producing very good quality print but the disadvantage of being limited to one character font.

A dot matrix printer has a printing head that typically consists of an 9 by 9 array of pins each of which can print a dot on the paper. Characters are printed by selecting the appropriate pins as shown in Fig. 19.3.

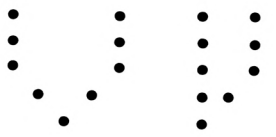


Figure 19.3 Dot matrix characters

By selecting different pins it is possible to change fonts. Hence with a dot matrix printer, italic, condensed or enlarged characters, etc. can be included in the text. It is also possible to print special symbols such as the Greek alphabet.

A paper copy of any screen display can also be printed as it can be created from patterns of dots. For example the hangman display in Chapter 18 could be transferred onto paper using a dot matrix printer.

The disadvantage of this printer is that the quality is not as good as for the daisy wheel printer since the characters are formed from a series of dots.

Solutions to the tutorials

19.1)
```
to pbend
fd 2 rt 2 fd 2 rt 2 fd 2 rt 2 fd 2 rt 2 fd 2 rt 2
end

to parc
pbend pbend pbend pbend pbend pbend pbend pbend pbend
end
```

Execute the word `parc`.

19.2)
```
to pbend
fd 2 rt 2 fd 2 rt 2 fd 2 rt 2 fd 2 rt 2 fd 2 rt 2
end

to parc
pbend pbend pbend pbend pbend pbend pbend pbend pbend
end

to nbend
fd 2 rt -2 fd 2 rt -2 fd 2 rt -2 fd 2 rt -2 fd 2 rt -2
end
```

```
to narc
nbend nbend nbend nbend nbend nbend nbend nbend nbend
end

to stem
parc narc
end

to leaf
rt -45 parc rt 90 parc
end

to whorl
rt -45 leaf rt -90 leaf rt 45
end

to head
leaf rt 45 leaf rt 45 leaf rt 45 leaf
end

to flower
cs whorl stem head ht
end
```

Execute the word `flower`.

Appendix One

Syntax errors

Some possible causes of syntax errors are suggested in this appendix. The errors relate only to the new statements introduced in the particular chapter.

Chapter 1

Syntax error: wrong spelling of **CLS**.

Chapter 2

Syntax error:
a) Wrong spelling of **PRINT**.
b) both the space after **PRINT** and the first quotation mark missing, e.g.

```
PRINTThis is the Amstrad computer."
```

Computer displays 0 instead of the phrase: the first quotation mark is missing, e.g.

```
PRINT This is the Amstrad computer."
```

Chapter 3

Syntax error:
a) Wrong spelling of **LIST**, **RENUM** or **EDIT**.
b) Space between **EDIT** and the line number missing, e.g.

```
EDIT30
```

Line does not exist: trying to **EDIT** a line that does not exist.

Chapter 4

Syntax error:

a) Remark with the REM missing.
b) Space after the REM missing.

Chapter 5

Syntax error:
a) Wrong spelling of SAVE, LOAD or CAT.
b) | before ERA or CPM missing.

Type mismatch during a SAVE or LOAD: first quotation mark round the program name is missing, e.g.

```
SAVE ROBOT"
```

Bad command: program name too long.

Chapter 6

Syntax error:
a) Wrong spelling of MODE, PEN or PAPER.
b) The space between these statements and the following number is missing.

Improper argument in ..:
a) Asking for a mode other than 0 or 1 at line .. .
b) Asking for a PEN or PAPER greater than 15 at line .. .

Chapter 7

Improper argument in ..: the LOCATE statement at line .. has placed the cursor outwith the screen.

Chapter 8

Improper argument in ..: trying to display a building block outwith the range 33 to 255.

Subscript out of range:
a) Space between PRINT and CHR$ missing.
b) Wrong spelling of CHR$.

Syntax error:
a) Brackets after CHR$ missing e.g. CHR$250.
b) Wrong spelling of MOVE, DRAW, FILL, TAG or TAGOFF.

c) Space after **MOVE** or **FILL** missing.
d) The comma between the co-ordinates in the **MOVE** and **DRAW** statements missing.

Improper argument in ...: asking for a **FILL** greater than 15 at line...

Chapter 9

Type mismatch in ..: this means that you have equated a string variable to a number or vice versa, e.g.

```
a="red"                    a$=2
```

Chapter 10

Syntax error:
a) = sign missing.
b) Using an invalid variable name.
c) Wrong spelling of **INT** or **RANDOMIZE**.
d) Space between **RANDOMIZE** and **TIME** missing.

Computer obtains the wrong answer to a calculation:
a) The label given to a memory cell is spelt in more than one way at different parts of the program, e.g.

```
30 random.number = INT(RND*6)+1

60 PRINT rndom.number
```

b) The wrong symbol is used in an arithmetic expression.
c) Brackets missing in the random number statement.

Chapter 11

Syntax error:
a) Space after **INPUT** missing.
b) The label for the memory cell missing after **INPUT**.
c) Missing comma in **INPUT** statement, e.g.

```
INPUT "answer"name$
```

Redo from start: you have entered a string at an **INPUT**

statement when the computer expected a number. The message is instructing you to enter a number and not to start the program again.

Chapter 12

Syntax error or program not obeying the IF statement correctly:
a) The space after IF or the space on both sides of the THEN or ELSE missing.
b) Wrong spelling of IF, THEN or ELSE.
c) Using the wrong symbols for less than, greater than, etc.
d) The THEN after an IF missing.

Chapter 13

WEND missing in ..: no WEND to match the WHILE at line .. .

Unexpected WEND in ..: no WHILE to match the WEND at line .. .
Operand missing in ..: no condition after the WHILE at line .. .

Chapter 16

Syntax error: space after SOUND missing.

Chapter 17

Syntax error: space after GOSUB missing.

Line does not exist in ..: the GOSUB at line .. points to a line that does not exist.

Unexpected RETURN at line ..:
a) There is no END statement at the end of the main program.
b) There is a RETURN in the main program.

BASIC statements

ABS, AFTER, AND, ASC, ATN, AUTO
BIN$, BORDER
CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG, CLOSEIN, CLOSEOUT, CLS, CONT, COPYCHR$, COS, CREAL, CURSOR
DATA, DEC$, DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE, DERR, DI, DIM, DRAW, DRAWR
EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR, ERROR, EVERY, EXP
FILL, FIX, FN, FOR, FRAME, FRE
GOSUB, GOTO, GRAPHICS
HEX$, HIMEM
IF, INK, INKEY, INKEY$, INP, INPUT, INSTR, INT
JOY
KEY
LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10, LOWER$
MASK, MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER
NEXT, NEW, NOT
ON, ON BREAK, ON ERROR GOTO, ON SQ, OPENIN, OPENOUT, OR, ORIGIN, OUT
PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT
RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM, RESTORE, RESUME, RETURN, RIGHT$, RND, ROUND, RUN
SAVE, SGN, SIN, SOUND, SPACE$, SPC, SPEED, SQ, SQR, STEP, STOP, STR$, STRING$, SWAP, SYMBOL
TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF, TRON
UNT, UPPER$, USING

VAL, VPOS
WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE
XOR, XPOS
YPOS
ZONE

Index

Computing is a skill and, like all other skills, can only be learned by practice. Don't just read this book, but try it out on your Amstrad.

A major feature of the 664 is that it includes a disk system. The built-in disk drive provides speed and convenience in loading programs and cataloguing is instant.

In the first few chapters the authors introduce you to your 664 and to BASIC with chapters on writing and storing programs, on the use of colour and drawing pictures. The book then covers use of the computer's memory, calculations, decision-making, loops, animation and creating computer music. The concepts are then used in a demonstration 'HANGMAN' program. A look is also taken at LOGO, commercial software and additional hardware. Appendices are provided on Syntax Error Messages and a list of BASIC statements.

The Authors
Jennifer Procter is a teacher of home computing at St. Martin's Community Education Resource Centre, Hamilton.
Cameron Procter is a Senior Lecturer in Physical and Life Sciences at Bell College of Technology, Hamilton.

ISBN 0-7447-0037-X

9 780744 700374

£9.50

INTRODUCING THE AMSTRAD CPC 664

Jennifer & Cameron Procter

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/