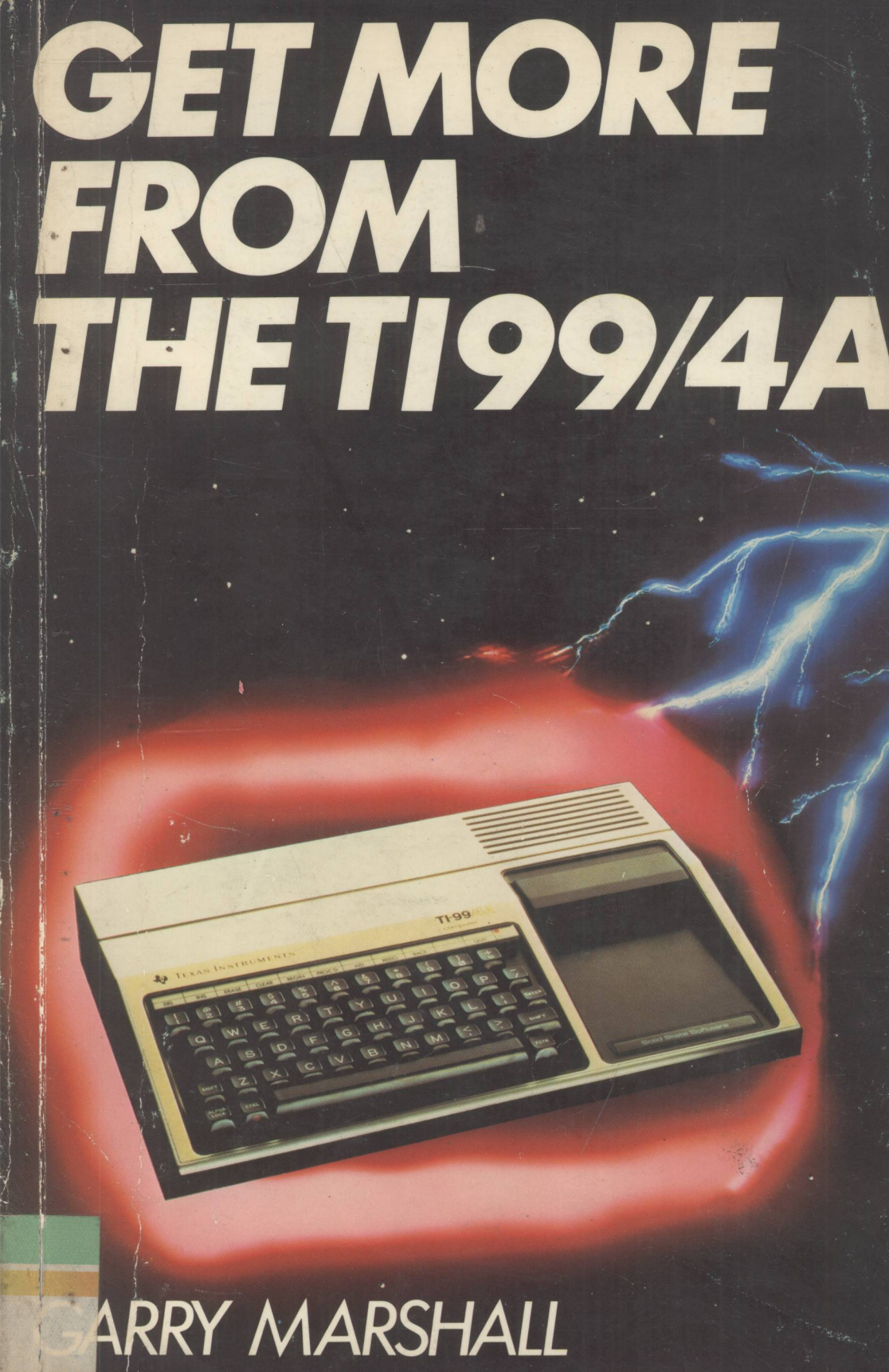
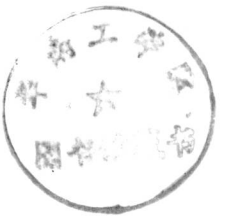
GET MORE FROM THE TI 99/4A



GARRY MARSHALL

Get More From The TI99/4A

Garry Marshall



GRANADA
London Toronto Sydney New York

Granada Technical Books
Granada Publishing Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Granada Publishing 1983
Reprinted 1983

British Library Cataloguing in Publication Data
Marshall, Garry
Get more from the TI99/4A.
1. Texas TI99/4A (Computer)
I. Title
001.64'04 QA76.8.T/

ISBN 0-246-12281-1

Typeset by V & M Graphics Ltd, Aylesbury, Bucks
Printed and bound in Great Britain

Preface

This book is about getting more from the Texas Instruments TI99/4A computer. The way of doing this that it advocates is to learn to program the TI99/4A effectively. Now, the User's Reference Guide that is supplied with the computer gives a rather good treatment of the features of TI BASIC and makes an excellent source of reference. It is not so good at explaining how to write and develop programs, however. This book aims to supplement the User's Reference Guide by showing how to develop interesting programs. By the same token, it avoids repeating material from the Guide, except for a little of the introductory material that must be included to make the book self-contained.

After a broad introduction to the features of TI BASIC in Chapter 1, the book moves on to developing short programs for colour graphics and sound effects in Chapter 2. In Chapter 3, it concentrates more on graphics and the production of effective screen displays. In the second and third chapters most of the features of TI BASIC that are needed in later chapters are introduced. At the same time utility routines are developed for performing tasks that cannot be achieved directly in TI BASIC. Chapter 4 provides a discussion on how to develop lengthy programs in a systematic way and on how to document them to make them readable. Then Chapters 5 to 8 are devoted to the development of some fairly long programs for interesting applications, including a Space Invader game and a simulation. All the programs presented in these chapters are written to take advantage of the particular strengths of the TI99/4A. The final chapter examines how the computer can be expanded, with particular emphasis on the programming languages that are available as alternatives to TI BASIC. In fact, the computer can be expanded with an impressive array of programs and peripherals.

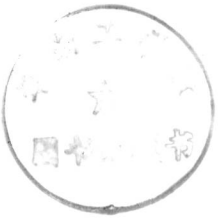
The book does not cover every aspect of TI BASIC. It does,

however, try to provide some motivation for each of the topics that it does cover. I hope that this approach will encourage the reader to use the TI99/4A with confidence, to experiment with its capabilities and, above all, to enjoy using it.

Finally, I should like to thank Richard Miles of Granada Publishing for his continual encouragement and for arranging for me to have the use of a TI99/4A with many programs and peripherals prior to and during the writing of this book. I would also like to thank Texas Instruments for making a TI99/4A available for me.

Garry Marshall

Contents



Chapter One
Introduction To TI BASIC

The version of BASIC that is built into the Texas Instruments TI99/4A is known as TI BASIC. It can always be accessed by pressing the 1 key when the computer's master selection list is displayed. Programming the computer in TI BASIC is one way to get it to do exactly what you want it to do. Whether you want it to display brilliant colour graphics, to play music, to play a game or to store and manipulate information, you can make it perform to your wishes by programming it.

The purpose of this introductory chapter is to introduce the features of TI BASIC so that we can see what capabilities it possesses and how they can be used as the building blocks for constructing programs. Later, in Chapters 2 and 3, we shall write some quite short programs and, in Chapters 6 to 8, we shall have progressed to the stage of developing substantial ones. Before proceeding to write programs of any length however, we shall pause in Chapter 4 to consider how they can be developed in a systematic way. Adopting a systematic approach to program development not only makes it possible for the reader to get the most from the programs themselves but also helps to ensure that the programs work properly!

The programs that are presented are intended to do more than demonstrate how to program the computer. They are all written with an eye to showing off the capabilities of the TI99/4A. Thus, its colour graphics and sound production capabilities are prominently featured. The programs also illustrate the kinds of uses to which computers can be put. In this way they should provide the reader with a fund of ideas for what to do with the computer either by enhancing the programs presented in the book or by using them as a launching pad for further, personal, developments.

The TI BASIC environment

TI BASIC provides a number of commands that make it easy to use. They are typed in directly and are obeyed as soon as the ENTER key is pressed. Tasks typical of those for which commands are provided are the entering, examining and altering of programs. The following Table 1.1 lists the more useful commands and gives summaries of their purposes.

Table 1.1. Some of the commands of TI BASIC.

Command	Purpose of command
BYE	To leave TI BASIC and return to the master title screen.
EDIT	To edit existing program lines. By typing EDIT followed by a line number, the program line with that number can be amended by replacing, inserting or deleting characters.
LIST	To list the program currently stored in the computer. A part of a program can also be listed by giving the line numbers of the first and last lines in the part as, for example, in LIST 200–300.
NEW	To erase the program currently stored in the computer and to prepare it generally for the entry of a new program.
NUMBER	To generate the line numbers for program lines automatically. When issued by itself the line numbers start at 100 and increase in steps of 10. However, the command can specify the starting number and the step as in NUMBER 200, 50 which gives an initial line number of 200 and then the numbers 250, 300 and so on.
OLD	To copy a program that is stored on cassette, or some other permanent storage medium, into the computer's memory. The command causes the instructions for operating the cassette recorder to be generated and displayed automatically.
RUN	To run the program that is currently stored in the computer.
SAVE	To save the program that is currently stored in the computer by copying it onto cassette or some other permanent storage medium. As with OLD, the instructions for operating the cassette recorder are generated and displayed automatically.

The features of TI BASIC

In essence, what most computer programs do is to accept and store information, to manipulate the information in some way, and to present the results of this to the person using the program. This pattern is evident in an arcade game program such as Space Invaders where the person playing the game provides the inputs by pressing keys as are appropriate for moving his missile base and for firing missiles. The inputs are manipulated by, first, determining which command they represent and then taking the appropriate action such as moving the missile launcher to the left. The result of this is presented to the user by an appropriate modification to the display screen. As an example from the use of computers in business, a stock control program gives a direct reflection of the pattern. Changes in stock are typically provided to the stock control program as inputs. This information is manipulated so that a correct representation of the current stock position is stored in the computer, and this can be displayed to show the state of the stock at any time. When a computer runs a program to enable it to control another item of electronic equipment, the program accepts as input signals from the equipment it is controlling. It then processes these signals to determine what action it needs to take, and then produces as output the control signals that will cause the necessary actions to be taken. Since most computer programs conform to this pattern, all computer languages, and TI BASIC in particular, must have features with which a programmer can direct the computer to accept information, store it, manipulate it, and display the results. Some of these actions can be achieved with commands, and we shall demonstrate this before moving on to show them being done by simple programs.

We can store a word such as 'Houston' in the memory of the computer by making an *assignment*. This is done by enclosing the word in quotation marks and assigning it to a variable. To do this we must give the name of the variable. A variable name should begin with a letter, it can be from one to fifteen characters long, and the other characters can be letters or numbers. (Actually, a few other characters can be placed at the beginning of or within a variable name, but we shall not do so in this book.) As far as possible, variables will be given names in this book which indicate the purpose to which they are being put. This helps to make programs more readable and easy to understand. Finally, if a word, rather than a number, is to be assigned to a variable, then the name of the variable

must end with a dollar sign. This is so the computer can tell when it is dealing with words and when it is handling numbers. Thus, one way to store our word is with the assignment command:

```
CITY$ = "HOUSTON"
```

When it is executed it causes the string of seven characters in the word to be stored in a part of the computer's memory which can be referred to as CITY$.

Numbers can be stored using similar assignments. For example, we can store the numbers five and six with the two assignments:

```
FIVE = 5
SIX = 6
```

Again, these numbers are stored in parts of the computer's memory that can be referenced by the names of the respective variables to which they are assigned.

With two numbers stored in the computer, we can write commands which cause these numbers to be processed and to store the results. For example, we can find and store the sum and the difference of the two numbers with

```
SUM = SIX + FIVE
```

and

```
DIFFERENCE = SIX - FIVE
```

When one of these commands is executed what happens is that the computer takes the part of the command to the right of the equals sign, which is written as it would be in ordinary arithmetic, and uses it to find a value. Thus, when finding the sum, it looks up the values assigned to the variables FIVE and SIX, and adds these values together. When a value has been found as a result of dealing with the right-hand side of an assignment, it is assigned to the variable whose name is given on the left-hand side. So, when the last two commands have been executed we have 11 stored under SUM and one stored under DIFFERENCE.

When these commands are obeyed there is no external evidence that anything has occurred, since they have caused events to happen only inside the computer, in its memory. However, to find out what has happened there we can use the PRINT command or, to equal effect, the DISPLAY command. Either of the commands

```
PRINT CITY$
```

and

```
DISPLAY CITY$
```

will cause whatever is stored under the variable name CITY$ to be displayed on the screen. In this case we shall see

```
HOUSTON
```

To see what is stored in the variables FIVE and SIX and the results that were stored in SUM and DIFFERENCE we can give the command

```
PRINT FIVE, SIX, SUM, DIFFERENCE
```

or

```
DISPLAY FIVE, SIX, SUM, DIFFERENCE
```

They will both cause the display

```
 5          6
 11         1
```

In this way, with simple assignment commands and commands involving PRINT or DISPLAY we can cause the computer to store and manipulate information, and to display the results. Each command is obeyed at once, and so to achieve a chain of actions we have to enter the successive commands one after another following the completion of the prior commands.

We turn now to writing programs to tell the computer what to do, rather than giving it commands to do one thing at a time. A *program* is a sequence of instructions that tells the computer how to perform a task when the sequence is obeyed. The computer must *store* the program first. When it is stored completely, it can be run using the RUN command. When a command is preceded by a number the computer recognises it as a program line and proceeds to store it as part of the current program. The number is usually referred to as a *line number* and the combination of number and command as a *program line* or *statement*. The computer uses the line numbers to order the program lines, constructing a program by placing the lines in increasing order of their line numbers.

A simple program to store two numbers, find their sum and difference, and display the result can be written based on previously given commands.

It is

```
100 FIVE = 5
110 SIX = 6
120 SUM = SIX + FIVE
130 DIFFERENCE = SIX - FIVE
140 DISPLAY FIVE, SIX, SUM, DIFFERENCE
150 END
```

Remember that just typing in the program as it is written automatically causes it to be stored. It can be listed by issuing the LIST command and run as often as you like by issuing the RUN command repeatedly.

This program is of strictly limited value since it always finds the sum and difference of the same two numbers. We can generalise it so that it can do the same for any two numbers that we might care to give the program when it is running by using the INPUT statement. When an INPUT statement is executed it causes the computer to display a question mark as a prompt and then to wait until an entry is typed and ENTER is pressed, when it assigns the entered value to the variable mentioned in the statement. Thus, execution of the statement

```
100 INPUT FIVE
```

will cause the entered value to be assigned to the variable named FIVE. The facility also exists for providing your own prompt rather than the question mark, and well-designed prompts make a program much easier to use. If we would like the prompt

```
FIRST NUMBER?
```

to appear when we should enter the first number, we can write the INPUT statement as

```
100 INPUT "FIRST NUMBER?" : FIVE
```

Note the use of the colon, which is compulsory. A program to accept any two numbers and find their sum and difference can now be written as:

```
100 INPUT "FIRST NUMBER?" : NUMBER1
110 INPUT "SECOND NUMBER?" : NUMBER2
120 SUM = NUMBER1 + NUMBER2
130 DIFFERENCE = NUMBER1 - NUMBER2
140 DISPLAY NUMBER1,NUMBER2, SUM, DIFFERENCE
150 END
```

A typical dialogue produced by running this program is:

```
FIRST NUMBER? 10
SECOND NUMBER? 2
10          2
12          8
```

Another way of providing data to a program is to use the READ and DATA statements. With these, the data (whether numbers or words) is given in the DATA statement or statements. The first READ statement that is executed in a program causes the first item of data to be read, the second READ statement reads the second item and so on. Clearly, since the data items must be explicitly listed in a DATA statement they must be known at the time the program is written. If this is not the case, then the use of an INPUT statement is probably a more appropriate way of providing data. The use of the READ and DATA statements is illustrated by the next program.

```
100 I = 1
110 READ WORD$
120 DISPLAY "WORD NUMBER "; I;" IS "; WORD$
130 I = I + 1
140 GOTO 110
150 DATA ABILENE, GALVESTON, LAREDO, AUSTIN
160 END
```

Note that since the data consists of words it is read into an appropriately named variable, WORD$. The GOTO statement is introduced in line 140. The effect of executing

```
GOTO 110
```

is to cause line 110 to be executed next. When this program is run it produces the display

```
WORD NUMBER 1 IS ABILENE
WORD NUMBER 2 IS GALVESTON
WORD NUMBER 3 IS LAREDO
WORD NUMBER 4 IS AUSTIN

*DATA ERROR IN 110
```

The program reads and displays the items of data, but it also gives an error. What has happened is that the GOTO statement in line 140 has created a loop that is executed for ever (unless an error occurs). Every time line 140 is reached it sends the computer back to line 110 again. However, the fifth time that the READ statement in line

110 is executed there is no data to read, for the DATA statement contains only four items. This is the cause of the error.

One way to amend the program is with the use of the *conditional* statement. This has the form:

IF condition THEN line number 1 ELSE line number 2

When executed, the condition is tested. If it is found to be true then the statement having its line number given by line number 1 is done next, otherwise that with the line number given by line number 2 is done next. The statement can be abbreviated to

IF condition THEN line number 1

In this form, the condition is tested, and if it is true the statement with the line number given by line number 1 is done next, otherwise the next line after the conditional statement is executed.

The last program can be corrected if we include a special item of data to indicate that it is the last item, and then use the conditional statement to detect it and cause the program to end properly. The resulting program is:

```
100 I = 1
110 READ WORD$
120 IF WORD$ = "END" THEN 170 ELSE 130
130 DISPLAY "WORD NUMBER "; I;" IS "; WORD$
140 I = I + 1
150 GOTO 110
160 DATA ABILENE, GALVESTON, LAREDO, AUSTIN,
    END
170 END
```

The program will display only the first four data items and not the last one which is only an end-marker and presumably not an item of data as such for the program.

Since the program that gave the error is about reading and displaying a word repeatedly, it could equally well be fixed using TI BASIC's facility for repetition. This involves the use of the FOR and NEXT statements. Using them, an alternative version of the fixed program is:

```
100 FOR I = 1 TO 4
110 READ WORD$
120 DISPLAY "WORD NUMBER "; I;" IS "; WORD$
130 NEXT I
```

```
140 DATA ABILENE, GALVESTON, LAREDO, AUSTIN
150 END
```

The repetitions are achieved by repeating the statements between the FOR and NEXT statements as often as directed by the FOR statement. In this case the first repetition is done with I = 1, the next with I increased by one to two, the next with I increased by one to three and finally with I = 4. The general form of the FOR statement is:

```
FOR variable = initial value TO final value STEP step
```

This gives the initial value to be assigned to the variable for the first repetition, the final value for the last repetition and the step by which the initial value is to be increased up to the final value for the repetitions in between. If STEP is omitted, as it is in the last program, it is taken to be one.

Finally, we will introduce TI BASIC's CALL KEY statement. It has the form:

```
CALL KEY (0, CODE, STATUS)
```

Its purpose is to allow interactive entry of data to a program. With its use a program can determine whether a key has been pressed and, if so, which one. However, the program does not halt, as it does with an INPUT statement, but proceeds immediately after executing CALL KEY to the next statement as it would with any other statement. When it has been executed, the value of STATUS indicates whether or not a key has been pressed, and the value of CODE gives the code for the character on the key that has been pressed. The codes are explained in full in the next chapter.

Summary

This chapter provided an introduction to the facilities of TI BASIC. First the commands it provides were described. Then the ways in which data entry, storage, manipulation and display can be achieved were explained. Input can be achieved with the INPUT, CALL KEY, and READ and DATA statements. Assignments permit both storage and manipulation. The PRINT and DISPLAY statements can both be used to display results.

Chapter Two
Graphics And Sound

The strong points of the Texas computer include its graphics and sound capabilities, and the ease with which graphics and sound effects can be programmed. In this chapter we shall examine the fundamental techniques for producing graphics displays and for generating sounds.

Graphics

A program or text appears on the screen when letters and numbers are positioned in the appropriate places. Letters and numbers are represented when stored in the computer by their codes. To give one example, the code for 'A' is 65. In fact, the computer uses the ASCII code to represent the standard characters that can be entered from the keyboard. The code is listed in Appendix 1. When the computer is switched on the codes from 32 to 127 are automatically assigned to the characters as shown there.

In the same way as positioning letters on the screen gives a paragraph of text, so a picture can be displayed by placing graphics characters on the screen. To illustrate this, the image shown in Figure 2.1 can be formed by combining the small number of graphics characters in Figure 2.2 in the way illustrated by Figure 2.3.

The computer itself does not provide any graphics characters. However, it does provide the user with the capability to define his own graphics characters. The codes from 128 to 159 have no characters assigned to them, and they are there, in essence, for the user to assign his own characters. Besides this, characters may be reassigned to any of the codes from 32 to 127 within a program if this should suit the user.

Every character that can be displayed on the screen occupies an area that consists of eight rows each with eight dots along it called a

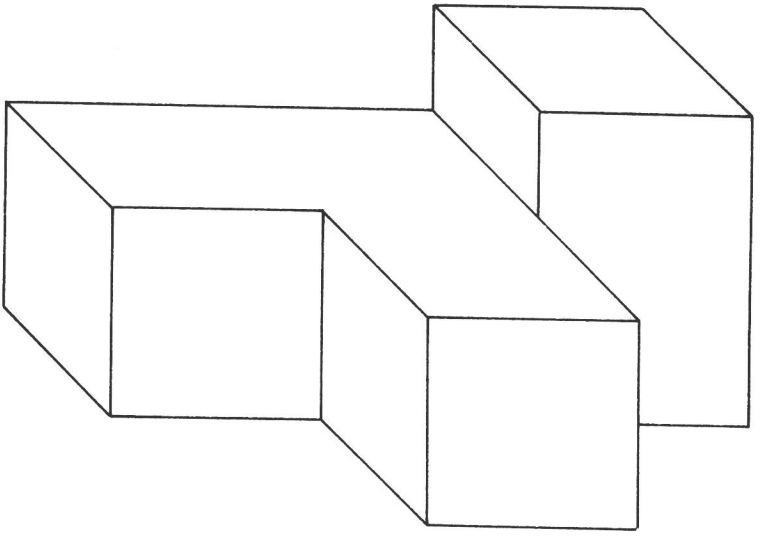


Fig. 2.1. An image.

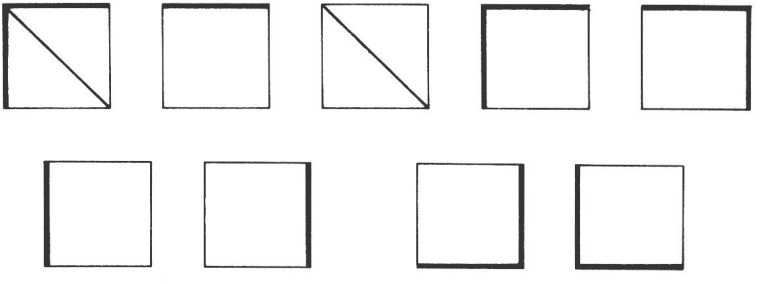


Fig. 2.2. Some graphics characters.

dot matrix. This is equally true whether the machine defines the character automatically or you define it yourself. A character is displayed in this area by (in monochrome terms) turning some of the dots on and leaving others off. Expressing this rather more appropriately in terms of colour, a character is displayed in colour by making some of the dots one colour and the rest another colour.