GETTING STARTED WITH THE TEXAS TI-99/4A

STEPHEN SHAW

Getting Started
with the

TEXAS TI 99/4A

Getting Started
with the

TEXAS TI 99/4A

Stephen Shaw

Phoenix Publishing Associates

First published in Great Britain by
PHOENIX PUBLISHING ASSOCIATES,
14 Vernon Road, Bushey, Herts. WD2 2JL

ISBN 0 9465 7604 1

Printed in Great Britain by
Billing & Sons Ltd.
Cover design by
Denis Gibney Graphics
Typesetting by
Prestige Press (UK) Ltd.

CONTENTS

Introduction

The Texas Instruments Home Computer, TI99/4A, represents an easy introduction to home computing. The machine is supplied with a language called TI BASIC. This simple implementation of BASIC includes useful and easily used subprograms to enable you to use the machine's graphics and sound capabilities.

TI BASIC is one of the simpler forms of the BASIC language, and as such you should find it easy to learn. Commands are typed in normally, you do not have to search for special keys. TI BASIC is not a fast language and you should not expect to write arcade speed games in this language. For really fast action games you must either purchase Modules, or the peripherals which are covered later in this book.

When you are ready to move on to more professional areas of computing, the TI99/4A is capable of expansion to an extremely powerful computer, and may even be used as a terminal for a large 'mainframe' computer.

The TI99/4A console on its own is only a start: it is not equivalent to a business minicomputer, and if you wish to use it now for any demanding task you should consult your dealer, who will advise you what extra devices you need to buy.

Texas Instruments follows a policy of continual product improvement, and the operating system of the TI99/4A may be subject to minor amendment.

Read this book, try a few short programs of your own, and as time progresses you will learn how to harness the power of your computer, but remember it will take time and practice.

1

Setting Up

How to prepare your console for operation

This information is included in the manual supplied with your console, but is supplied here, in a slightly different format, to provide you with a central source of information.

Unpack the console carefully. The power supply and television modulator are separate units packed with the console.

If your dealer has not fitted plugs onto the power supply cable, you should attach a mains plug fitted with a 3 Amp fuse.

The power supply may become quite warm, so place it where high temperatures can cause no damage. It will be out of the way if you place it on the floor, preferably not on a carpeted surface.

You may find that you will use the console for many hours at a time, so give careful consideration to its location : a table or desk with a firm top and room for you to spread your papers and books. You will need a chair which allows you to sit comfortably without backache.

If you wear glasses, the tv will not be located at the most

comfortable distance whether you are near or short sighted. You cannot damage your eyesight using the computer, but the use of glasses prescribed for the distance of your tv set when you compute, will alleviate problems of eye strain.

The tv modulator connects your tv to the console. Plug the 'DIN' audio plug into the computer and the coaxial plug into your tv. You will need to tune your tv to Channel 36, and it is essential that your tv is able to 'fine tune'. If you have problems, ask your dealer. Some fringing occurs on some colours: this is normal, but you should tune your set for the sharpest picture you can which does not introduce heavy diagonal lines over the entire picture.

When the console has been connected to the mains and to your tv, you may switch on and adjust your tv set for best reception. When properly tuned, the computer sound output should be heard on your television's loud speaker. It may be necessary to slightly adjust the tuning. To produce a sound output, press any key (except SHIFT,FCTN,CTRL, or ALPHA LOCK) so that the tv display changes to: PRESS:

1 to TI BASIC

Instead of pressing 1, press the space bar, and a brief tone will be produced. If you have difficulty adjusting the tuning to provide both sound and picture together, your television may be unsuitable, and you should consult your dealer.

Your computer is now ready for use. Press Key 1 and you are ready to key in a program in TI BASIC.

If you wish to use a TI Module, push it gently into the slot at the right hand side of the computer. The tv screen will

flicker and the 'test card' will reappear. Press any key for the 'menu' to select TI Basic, or the contents of the module.

You may need more than one attempt, as module contacts can become tarnished or grimy. If the computer ceases to respond, it is a 'lock out', something which happens with all computers. Do not panic — switch the console off and try again.

DO NOT touch the printed circuit board contacts in the module or you may destroy the electronic components within.

How to key programs into the computer

Your computer is capable of much, but requires a little help. It can do nothing unless it is first programmed either by inserting a preprogrammed module, or by keying in a program, written by yourself, or from a magazine.

BASIC (the name given to the 'language' most home computers are programmed in) was once quite standardised, but with each new computer adding new commands, and new features, there is now a great difference in the versions of Basic used in different computers.

Your TI99/4A uses TI BASIC, and your computer can only understand programs which are written in TI BASIC.

BASIC is very similar to English, using common words to instruct the computer, but the computer is very precise in it's requirements for the way those instructions are keyed in.

Each command word is separated from any other word

or number by a space. Some computers permit you to leave out spaces, but in order to allow you a wide choice of variable names, your 99/4A needs to see those spaces.If you put in a space where one is not required, the computer often removes it itself.

Take special care when entering the number 1 or the letter I, and the number 0 and the letter O. These can very often look similar in magazine listings. If you use the wrong one, the computer will usually be unable to RUN the program, and halt with an error message, such as BAD VALUE or STRING-NUMBER MISMATCH.

If the printed program contains DATA lines, check them thoroughly, as one comma too many or too few can cause a "program crash" when you try to RUN the program you have entered.

Learn to use the SHIFT key on the left: if you use the key on the right of the keyboard the inevitable accident will occur and instead of entering a + (SHIFT and =) you may press the FCTN key and the =. This will cause a system reset and you will lose your program.

Take note of the spaces in the printed program: if one is printed, you should enter it!

Listings will run if entered carefully, but occasionally a misprint occurs, or sometimes whole lines will be omitted.

If you do encounter problems, carefully check each line you have entered. Information on 'debugging' or correcting programs can be found in the section of 'How to use TI Basic'.

Normally you should key programs in with the ALPHA LOCK key in the DOWN position, unless the listing clearly

uses lower case letters, in which event release ALPHA LOCK when entering those letters.

2

TI BASIC

In this chapter we will look at the language in your console. There are a number of general books now available on the BASIC language, and one or two of these may help you if you experience difficulty in handling the language. Many evening classes in computing are also available.

For greater assistance we will follow as closely as possible the order of the Texas Instruments Manual.

Do not try to take in all the information in one reading, but go back and read it again a few times.

A computer works as a large number of switches, which are either on or off. Each 'switch' is described as a BIT. In order to pass information more quickly, the computer looks at more than one BIT at a time. The TI99/4A uses a 16 bit processor: it is able to look at 16 bits at a time. For most purposes however, the computer looks at 'words' composed of 8 bits. These words are called BYTES. A BYTE is a binary number composed of eight numbers, which may be 0 or 1. In digital representation the BYTE has a maximum value of 255 (Binary 11111111).

The computer stores its commands (reserved words) as one byte, rather than a collection of letters. It can only identify the command words if you follow the rules regarding the characters permitted in front of, and

following command words. In general, you may only use a space, arithmetic operator, or ENTER, but there are exceptions which you will see in the program listings in the manual and in the books of TI programs now available.

For discussion of the error tracing commands (Trace, Untrace, Break) see the section of How to use TI Basic.

LIST

The command LIST is used to list the contents of a program. Used on its own, it will list the program on the screen. You may use CLEAR (FCTN & 4) to halt the LIST. To start again at say line 400, you type in LIST 400- the hyphen indicating 'to the end', of LIST 400-600

LIST can also be used in many other ways. These are described in the chapter on modules and peripherals.

SAVE

Please refer to the section on cassette handling.

LET is optional, but uses up one byte of memory every time you use it. It is better to avoid it's use.

LET A=2 is the same as

A=2

END is also optional, but in this case it is good practice to use it. By adding END to your program, you may be certain when you list it in the future, that you have the complete program, and not a 'working copy'.

IF . . . THEN . . . ELSE

TI BASIC may appear to be slightly limited in its use of IF . . . THEN compared to some other computers. TI do however allow the ELSE alternative.

The problem arises because TI insist that you use the construction only to transfer to another line. You cannot add commands such as:

```
IF X=B THEN B=C
```

to do this you need Extended Basic.

However, TI BASIC does have 'relational operators' which will often help you out of this problem. These may be found described in the section on Advanced Programming.

FOR TO STEP

Note that in TI Basic you must always use the variable name after NEXT . NEXT on its own is in error. In some early computers you were not allowed to transfer to another line once a FOR NEXT loop had been established, but with the TI99/4A you need not worry. You may leave a FOR NEXT loop before the loop has been completed.

Sample use:

```
100 FOR FREQ=110 TO 200
110 CALL SOUND(100,FREQ,0)
120 NEXT FREQ
```

For . . to . . step may also be used to provide delays:

```
100 FOR DELAY=1 TO 300
110 NEXT DELAY
```

will take a little over a second to complete in TI Basic.

INPUT

Try to use a separate INPUT for each variable. It is possible to input more than one variable eg by using INPUT A,B but this requires the program user to input two numbers separated by a comma.

The TI form of input, INPUT "HOW MANY?":N uses a colon separator (:), most other Basics use a semi colon (;).

VARIABLES

When you wish to refer to a number, you may use that number, or a 'label' representing the number. For instance, if we tell the computer:

```
A=2
```

Then whenever the computer comes to 'A' (without other letters, that is, with spaces or brackets on either side), it will treat it as the number 2.

'A' is a VARIABLE, and can be allocated to any number. The TI99/4A may have variable names up to fifteen letters long: you may for instance use:

```
HIGHSCORE=12000
```

A variable representing a number is a NUMERIC VARIABLE and a variable representing a letter, a word, or a group of words is called a STRING VARIABLE. A string variable always ends with the dollar sign:

```
MESSAGE$="YOU WIN"
```

Strings (as they are called) are dealt with later in this book.

READ...DATA...RESTORE

TI Basic is slow at reading DATA lines, and if you need to use a number of READs, it is essential that you do not do it more often than absolutely necessary. It is a good idea to fill a variable array, and refer to that. (ARRAYs are dealt with at some length later in this chapter).

eg
```
FOR I=1 TO 5
READ A
IF A=1 THEN 200
NEXT I
DATA 2,3,1,0,6
```

if used often, could be replaced with:

```
FOR I=1 to 5
READ B(I)
NEXT I
```

then when a check is required

```
FOR I=1 TO 5
IF B(I)=1 THEN 200
NEXT I
DATA 2,3,1,0,6
```

It is worth mentioning that DATA causes more problems in debugging a program than any other command. There must be enough DATA to fill all the READs in the program, and they must be numbers if a numeric variable is READ.

Be careful how many commas you use in your DATA lines: too many or too few can cause many hours

searching for errors. The error messages you will receive may be some distance from a READ line, if you have loaded an incorrect value into a numeric variable due to missing out just one comma.

PRINT

TI Basic has a fairly slow screen scroll, but your information will appear more quickly if you use the print separators instead of a number of separate PRINT lines. You will also save memory.

eg
```
100 PRINT "PRESS"
110 PRINT "2. TO TERMINATE"
130 PRINT
140 PRINT "H FOR HELP"
```

Will appear more quickly if you use:

```
100 PRINT "PRESS":"1. TO START":"2. TO TERMINATE"::"H FOR HELP"
```

TI Basic allows you to key in a program line up to 4 screen lines long, so use this facility. Notice that instead of a single PRINT to scroll one line, an extra colon has been used in our single line amendment. Each colon causes the screen to scroll once.

COLOUR AND SOUND

The TI99/4A allows you to set the screen, and the foreground and background colours of the characters, to any of fifteen colours, plus transparent. The transparent colour allows the screen colour to show behind a character.

CALL SCREEN is used to set the screen colour, and

CALL COLOR is used to set the character colours.
Note that COLOR is spelt the American way.

To Try:

```
100 FOR N=1 to 8
110 CALL HCHAR(N,1,24+N*8,32)
120 NEXT N
130 FOR N=1 TO 16
140 CALL SCREEN(N)
150 FOR DELAY=1 TO 300
160 NEXT DELAY
170 CALL COLOR(N,N,1)
180 NEXT N
190 END
```

In BASIC, characters are referred to by standard codes known as ASCII CODES. The ASCII code for the capital letter A is 65 for instance. These codes may simply be referred to as CHARACTER CODES.

TI BASIC allows you to define characters with the ASCII codes 32 to 159, and for colour purposes these are divided into sixteen sets. You may define different colours for each set, but all of the characters in that set must be the same colour.

Characters are defined using CALL CHAR (CODE,STRING$), where STRING$ is a string, or string variable, made up of HEXADECIMAL characters (Ø to 9 plus A to F).

As each character occupies a grid of 8 x 8 dots, it can be defined by splitting it down the middle to form 16 rows of 4 dots.

Each possible combination of ON and OFF dots in a row of four can be defined in terms of one of the sixteen hexadecimal characters.

A character with one dot ON in the top right corner is defined as: "0100000000000000".

The definition is by row, first the left side then the right. Each row of 4 dots can be considered a row of binary switches. The right switch is 1, the next 2, the next 4 and the leftmost switch 8. When a dot is ON, add its value to the others which are ON in the same row. You will obtain a unique number, from 0 to 15.

From this decimal number you change to a single hexadecimal digit (hexadecimal numbers have a 'base' of 16). The number 15 for instance is hexadecimal F.

If you wish to place a single character on the screen, use the CALL HCHAR command, it is slightly faster than CALL VCHAR.

TI BASIC programs will run faster in EXTENDED BASIC, but it must be noted that this language has only 145 character sets. Therefore, it you use characters coded 144 to 159 in your TI BASIC programs, these programs will not run in Extended Basic.

There are several versions of the TI console around, and there have been slight changes in the relative shades of the colours used. If you purchase a program, and the colours look odd, it is because it was probably written on a 99/4, or an NTSC 99/4A but you should be able to change the colours to something more suitable by amending the line numbers.

Sound is produced with CALL SOUND TIME,F1,V1,F2,V2,F3,V3,N,V4) where TIME is in

milliseconds, F1,F2 and F3 are FREQUENCY in Hertz (cycles per second) and V,V2 and V3 are volume (0 loudest, 30 quietest). N is a noise generator which provides some sound effects.

A CALL SOUND may use only one frequency if you wish: the second and third frequencies are optional and may be omitted. The Noise is also optional.

A CALL SOUND will occupy the computer for about 50 milliseconds, and then, even if the sound is still continuing, it will proceed with the next instruction. If it comes to another CALL SOUND, the computer will wait until the first has finished, unless the second CALL SOUND has a negative time, in which case the first CALL SOUND will immediately be terminated and the second CALL SOUND begin.

TI state that you can only have tones down to 110 Hz on your computer, but that is not quite the case:

Try:

```
100 INPUT A
110 IF<37 THEN 100
120 CALL SOUND(2000,200,30,200,30,A*3,30,-4,0)
130 CALL SOUND(500,200,30)
140 GOTO 100
```

It would appear that the console can at least appear to go well below 110Hz. Try an input of say 50 or 60. If you find the sound interesting, try changing the -4 to -8.

Keep in mind that the computer takes about 40 milliseconds to process a CALL SOUND command. It is not possible to use this command to change the TYPE of sound produced.

CALL KEY

Call Key is used to sense the use of a key on the keyboard and permits data to be entered without scrolling the screen (the INPUT command causes the screen to scroll). An ACCEPT AT routine using CALL KEY can be found later in this book.

If you wish the computer to assume the ALPHA LOCK key is down, while a program is running, you can instruct it using CALL KEY, and avoid having to request the program user to ensure the key is down.

Use CALL KEY(3,K,S) in your program, and as soon as the program passes over it (no key has to be pressed) the computer will consider the alphalock key to be down, whether it is or is not. You resume normal operation with CALL KEY(5,K,S). These switching calls can use any variables you wish, and may be dummy calls or you may actually use them to obtain a key response.

In the appendix section you will find a list of the key codes which are available from the keyboard using the CTRL and FCTN keys. The normal keyboard, with and without using SHIFT will allow you access to characters 32 to 127 (a few of these codes do require the FCTN key).

IMPORTANT: if you use the split keyboard, CALL KEY(1 . . and (2 . . the value returned for keys X and M is only approximately zero, and although it prints on screen as zero it will not equate with it. Instead of using IF KEY =0 THEN, you are forced to use

```
IF KEY+1=1 THEN...
```

CALL JOYST

A program illustrating the use of this command is

developed in a later section of this book.

Please note that when using the joysticks, the alphalock key must be in the up position. If alphalock is down, the computer will not be able to sense when the joystick is pushed upwards.

Using Call Key(3. . .) does not affect joystick operation, but you should not use CALL JOYST(3. . .) as this may prevent correct operation of the joystick.

ATN

ATN is a trigonometrical function which you may not need to use often, but in TI BASIC it may be used to obtain an accurate value for the mathematical constant PI:

```
PI=4*ATN(1)
```

This and the other trigonometric functions provided work in radians. You may convert radians to degrees by using ATN:

```
DEGREES=RADIANS*180/4*ATN(1),    or
more simply
DEGREES=RADIANS*45*ATN(1)
```

INT

INTEGER is a numeric function you will use quite often. It removes the fraction from a number, so that 2.3 for instance becomes 2. It is frequently used with the RND function, and can also be used to round decimal numbers.

For example, if B is a decimal number to 13 places, and you wish to print only the first two places, you could use:

```
PRINT INT(B*100)/100
```

If you wanted the last decimal to be 'rounded', the alternative is

```
PRINT INT(B*100+.5)/100
```

RANDOM NUMBERS

Random numbers are useful in any program where you need to follow an unpredictable path. A program to display dice would be an example.

The 99/4A generates pseudo random numbers. If you have a short program:

```
100 FOR I=1 TO 10
110 PRINT RND
120 NEXT RND
```

Every time you run the program the SAME 'random' numbers will be printed. This can sometimes be of value if you wish to be certain of the effect but still have the appearance of randomness.

You may instruct the computer to start the list of 'random' numbers somewhere else, by adding the line:

```
90 RANDOMIZE N
```

where N is a numeric variable. The value of N determines where the random numbers begin.

Merely adding 90 RANDOMIZE, without 'seeding' the function with a value, will cause the computer to start at a truly random number, and if you run the program several times, different values will be printed each time.

RND takes a value between 0 and 1, and is usually used in the format: NUMBER=INT (RND*MAXIMUM)+1

The variable NUMBER will then take any integer value from 1 to MAXIMUM, including maximum. The maximum value of RND is .99999' so INT(RND*100) gives a maximum of 99. You have to add the odd 1 to enable you to actually reach the maximum you want.

You may prefer to have at the start of your program:

```
DEF RAN(X)=INT(RND*X)+1
```

Then when you want a random number up to say 12, you may enter in your program: NUMBER=RAN(12). You have created your own random function. Also see DEF.

SQR

SQR is used to obtain the SQuaRe root of a number:

```
A=SQR(4)
```

Many computers will not equate SQR(4)=2, or fail on some other comparison, due to internal rounding of numbers. Your 99/4A will equate all ten squares up to 100. Try it on a friend's computer. You may not need to use this very often, but it is an indication of the numeracy of the 99/4A. Try:

```
100 FOR I=1 TO 100
110 A=SQR(1)
120 IF I=A*A THEN 130 ELSE 140
130 PRINT I;"PASSED THE TEST
    CORRECTLY"
140 PRINT "NEXT VALUE OF I"
150 NEXT I
```

NOTE that 1,2,4,9,16,25,36,49,64,81 and 100 pass the test OK. The failures are due to internal rounding, which still exists, but it is not quite so marked on the 99/4A as on other computers. If your program needs to make a comparison such as this, use the INT function to remove any (unprinted) fraction.

In the program above for instance, there is some improvement by using INT(A*A) in line 120.

There is a bigger improvement using

```
110 A=INT(SQR(1))
```

which will remove from A any invisible fraction.

STRING EXPRESSIONS

A STRING is a non-numeric value or variable. A letter of the alphabet or a word or group of words may form a string.

NB: A NUMBER may also be a string:

2 is a number, "2" is a string.

A number (no quotation marks) must be used for mathematical operations. A string expression is therefore identified by quotation marks, or if represented by a variable, by a dollar sign after the variable name:

```
MESSAGE$="I WIN"
```

POS

The POS function is rarely used on other computers, but enables you to program very concisely on your 99/4a.

In the following example, CALL KEY is used to detect whether keys A B C or D are pressed, and control is passed accordingly. First, without using POS:

```
100 CALL KEY(0,K,S)
110 IF K=65 THEN 200
120 IF K=66 THEN 250
130 IF K=67 THEN 300
140 IF K=68 THEN 350
150 GOTO 400
```

In this case, the keys have adjacent ASCII codes, and it would be possible to use:

```
110 IF S=0 THEN 400
120 IF (K<65)+(K>68) then 400
130 ON K-64 goto 200,250,300,350
```

omitting 140 & 150.

However, in many games you may wish to test for keys which are well spread, such as AKESDXQPP. The POS function can then offer the solution. Still using ABCD:

```
100 CALL KEY(0,K,S)
110 IF S=0 THEN 400
120 ON POS("ABCD",CHR$(K),1)+1
GOTO 400, 200, 250, 300, 350
omit 130-150
```

If the key pressed is not in the string used in POS, then the expression has a value of zero, so one is added to enable us to use ON . . GOTO, and the first transfer occurs if an unwanted key is pressed. In this case 'only' three lines have been saved, but if you wish to use more valid keys, you still only need to use three lines. This can be very useful in a program.

Although you may use a string up to 255 characters long, the POS function is unreliable for strings longer than 127 characters.

SEG$

SEG$ is used when you wish to print a SEGment of a string, or remove a part of a string.

TI BASIC uses only one command to segment strings, SEG$. Other computers use LEFT$, RIGHT$, and MID$, but you only really need the one.

It is used for instance in this DISPLAY AT routine taken from THE TEXAS PROGRAM BOOK.

```
100 REM
110 REM PRINT AT X,Y,M$
120 REM ROUTINE
130 REM
140 FOR J=1 TO LEN(P$)
150 IF Y<32 THEN 180
160 Y=3
170 X=X+1
180 IF X<24 THN 200
190 X=1
200 CH=ASC(SEG$(P$,J,1))
210 CALL HCHAR(X,Y,CH)
220 Y=Y+1
230 NEXT J
```

(Set X and Y to the start position of your word, placed in M$. Then GOSUB this routine, and remember to add RETURN at the end to go back to the place in your program you left).

VAL

VAL is intended to make a number contained in a string available as a number, for use in mathematical operations. It changes "2" into 2, and may be used NO=VAL("2"). It is the opposite of STR$, used to change a number into a string:

```
A$=STR$(2)
```

The TI VAL function will change a string such as "2" to the numeric variable 2.

For instance: A=VAL("123")

This is of great importance when memory space is short, as a string variable representing "2" uses less memory than a numeric variable representing 2. This is explained in the section on Advanced Programming.

Please note that the TI VAL will only work if the string contains numbers only. It will not function for numeric expressions such as "2*3+8" nor if alphabetical letters are used such as "12 APPLES".

DEF

DEF is used to DEFine your own functions. TI Basic gives you great freedom in using this function — you are not restricted for instance to using a definition commencing FN as on some computers. DEF may also be used to create string functions.

Below are some examples of DEFs you may wish to use:

To print a random number from 1 to X.

At the beginning of your program type:

```
DEF  RAN(X)=INT(RND*X+1)
```

Then in your program, when you wish to use such a number, you type (for example)

```
A=RAN(7) or
A=RAN(LEVEL)
```

where LEVEL is a numeric variable.

Note that although X is used in the defining statement, you may use any number or variable when you use the new function. Although the function has here been called RAN, you may use any name you wish so long as it is not a reserved word (see TI manual for list).

The DEF statement is best used at the beginning of your program.

```
DEF  PI=4*ATN(1)
```

Then when you wish to use PI in your program, just use the variable PI. NB: This has the same effect if you omit the DEF function. There is no advantage in using DEF instead of a simple LET.

Further uses of DEF may be found in the Advanced Programming Section.

ARRAYS

Arrays in TI Basic may have 1 2 or 3 dimensions.

A one dimensioned array may be thought of as a tower of building blocks. Each block has written on it a value or

message. To find out what the message on block 3 is, we count upwards to the third block...

In TI Basic, normally the 'bottom' block is considered to be Block 0 (zero).

A tower of blocks may have blocks up to Block Number 10 without having to tell the computer how many blocks there are, but for a larger array, the computer has to be advised to reserve memory by using the DIM statement.

If you want to use an array of 16 values, before you use any of the values you must have the line DIM NAME(16) where NAME is the numeric variable for the tower of blocks.

We can place values in the array elements as follows:

```
100  DIM NAME(16)
110  FOR N=0TO 16
120  NAME (N)=N*2
130  NEXT N
```

Now the variable NAME(3) has a value of 6. The variable is referred to in this way, with the 'level' after the variable name, and in brackets.

TI Basic differs from some other basics in the way it handles arrays in some important ways:

You must use a number in the DIM statement. A variable is not accepted.

Once an array has been DIMensioned, you may not alter the size of the array.

If an array variable occurs in your program before a DIM statement, the array is automatically dimensioned as 10. You cannot then use the DIM statement for that array.

Both numeric and string variables may be arrayed.

TI Basic does NOT permit you to use the same name for an arrayed variable and a simple variable. You cannot use both NAME as a variable and NAME(3) say.

Arrays reserve sections of memory, and you should not use a larger array than is needed. A numeric array will use 8 bytes for each element, regardless of the value in the element.

A string array initially occupies 2 bytes per element, but, as you place strings in the elements, the space for each element will depend on the string you place in it (memory used = number of characters plus 1 byte).

If you can use string arrays instead of numeric arrays, you will usually save a great deal of memory.

A string array can be converted to a number by using the VAL function: eg A=VAL(SCORE$(2))

IF YOU DO THIS, ensure that you place the character "0" (zero) in any empty sectors to prevent possible program crashes.

As stated above, the TI 99/4A will assume the bottom sector is called Number Zero. Often you will find that having the base called Number One is quite adequate. If you do not use the 'zero' element, it is a waste of memory to have that element reserved, and TI allow you to instruct the computer to use a Base of 1.

The instruction to do this is OPTION BASE 1 (Note: The number is NOT in brackets!).

You can use the base 1 for all your arrays or none of them. Once set, you cannot reset the base, and as it is automatically set by any reference to an array before the

computer finds the OPTION BASE statement, it MUST occur before any such reference.

In TI Basic therefore, FIRST: If you wish to set the base to 1, use OPTION BASE 1. Then, if your array is to have more than 10 elements, use DIM ARRAYNAME(NUMBER). Then you may use the array!

You may become more familiar with arrays by studying printed programs. (NB: Some other computers use the DIM statement to reserve memory for simple strings. This does not apply to the TI99/A.).

Two dimensional arrays are similar. It may help if you consider the first element name as a hotel floor, and the second as a hotel room. For instance: DIM NAME(FLOOR,ROOM).

The use of multidimensional arrays should be undertaken cautiously, and careful consideration should be given to the use of OPTION BASE 1.

For example, a numeric array having dimensions (20,20) totals 21x21, or 441 elements, at 8 bytes each, that is 3.5k of memory used with just one DIM line! Use OPTION BASE 1 and the number of elements drops to 20x20=400. Times 8=3.2k, saving 300 bytes.

SUBROUTINES: GO SUB-RETURN

Whenever you use the same routine several times in a program (for instance the PRINT AT routine in THE TEXAS PROGRAM BOOK, reprinted in this chapter under SEG$), you may use GOTO and ON FLAG GOTO, but it is easier to use GOSUB to enter the routine, and when you have finished, RETURN will send you back to the program line immediately following the GOSUB with which you entered the subroutine.

Note: The computer stores the line number from which you GOSUB. This memory is only freed when you RETURN. If you GOSUB, ensure that you RETURN or you will quickly find a MEMORY FULL message appearing.

It IS possible to jump into a subroutine with GOTO provided you jump out with another GOTO, but that type of programming can lead to errors if you are not very careful, and should only be resorted to when you have absolutely no option. This should be rare.

An ideal example of this can be found in a short routine to find out how much free memory there is after you have typed a program in. Add on to the end:

```
10000  A=A+B
10010  GOSUB 10000
```

Now type in RUN 10000. This little program will now run, and in due course the MEMORY FULL message will appear. Now type PRINT A, and press ENTER. The value of A is approximately equal to the memory space remaining.

(Your program may still not run even if there is a lot of space remaining: memory is still required to handle the values of the variables, the return addresses of GOSUBs and so on).

You MAY use GOSUB to enter a subroutine at any stage, you do not have to GOSUB to the beginning of a routine.

Within the limitations of memory, you MAY GOSUB from a subroutine, but keep an eye on the number of RETURNS!

The ON . . . GOSUB command is similar to the ON . . . GOTO command (which see), except that instead of a

simple line transfer, the computer remembers where it has jumped from, and a RETURN will send it back to the line immediately following the ON . . . GOSUB.

REMEMBER to watch your RETURNs.

PECULIARITIES

'PAUSES'

When a program is RUNning, from time to time your computer will appear to stop operating for a very short period. This pause is especially noticeable when using the PRINT AT routine to be found later in this book, or when using Sprites with Extended Basic.

The reason for the pause is called 'garbage collection'.

When you amend a program line by re-entering it, or by using the Edit mode, there is a pause before the cursor reappears. This pause becomes more extended in a long program.

During this pause the computer is deleting the previous version of the line, moving all the following lines up in memory and adding the new version of the line to the bottom of the program memory: in short, doing a great deal of work.

When a program is running, and variables are defined, the values of the variables are stored in memory. When a new value is allocated to a variable, to avoid frequent delays to your program, the computer retains the old value in its memory, even though it will use the new value.

As time progresses, the memory will become full of these old variable values. When memory becomes full, the computer discards the redundant values : this is called garbage collection. It is more efficient to only do this when memory becomes full than every time a variable is redefined, but a very small pause is caused.

These pauses will be more frequent if your program is a long one, as there will be less memory to fill up with dead variable values.

REDUNDANT CHARACTER DEFINITIONS

When you switch the console on, some characters are undefined. If you define these characters in one program, the computer will retain that definition even if you use NEW and load a new program. Only by using BYE or QUIT will the definition be erased.

Therefore never assume a character is undefined : you may have already run a program which has defined that character!

Example:

```
Type in:
100 CALL CLEAR
110 A$="FF8181818181FF"
120 B$="0000FF0000FF0000"
130 CALL HCHAR(12,1,140,128)
140 GOSUB 210
150 CALL CHAR(140,A$)
160 GOSUB 200
170 CALL VCHAR(1,12,140,120)
180 CALL CHAR(140,B$)
190 GOSUB 210
200 STOP
210 FOR T=1 TO 1000
220 NEXT T
230 RETURN
240 END
```

RUN this program. When it ends, type in:

PRINT CHR$(140) (ENTER)

Notice that the definition is still there.

If you wish, RUN the program again and note the difference at the beginning, as the character is no longer undefined.

Now type in:

NEW (ENTER)

and repeat:

PRINT CHR$(140) (ENTER)

The definition is still there. If a new program is loaded, which uses this character and assumes the character is undefined, the character will not be printed as a blank but as the character we have defined with the above short program.

This TI Basic program simulates two puzzles

The screen is used as a memory device, with CALL GCHAR used to find out what is in a particular position, and then the information is manipulated and new characters displayed.

The program can be speeded up by using a 6 x 6 array to hold the information and using that instead of GCHAR. This program will work: can you make it work better?

NOTE: Many variables in this program are the letter I, or a letter I with a number following.

Be careful to distinguish between the letter I and the number 1.

```
100 REM  SQUARES   S SHAW 1981       4X4 IS BEST
110 RANDOMIZE
120 DEF RAN(X)=INT(X*RND)+1
130 GOSUB 2200
140 GOSUB 2100
150 GOSUB 1340
160 GOSUB 980
170 FOR I=8 TO 6+2*V STEP 2
180 FOR I2=5 TO 2*H+3 STEP 2
190 CALL GCHAR(I2,I,I4)
200 POSR=I2
210 POSV=I
220 IF I4=32 THEN 260
230 NEXT I2
240 NEXT I
250 REM
260 IF SCR=0 THEN 300
270 SCR=SCR+1
280 IF SCR=2 THEN 360
290 RETURN
300 CALL KEY(0,A,B)
310 IF A=ASC("P")THEN 130
320 CALL HCHAR(15,22,63)
330 CALL HCHAR(15,22,32)
340 IF B<1 THEN 260
350 KEY=POS("12345678900WERP",CHR$(A),1)+1
360 IF (H=4)*(KEY>5)THEN 260
370 ON KEY GOTO 260,390,450,520,690,860,890,920,950,2300,2330,
    2360,2390,2420,2450,130,130
380 SCR=SCR+1
390 REM  ^
400 IF POSR=5 THEN 260
410 CALL GCHAR(POSR-2,POSV,I3)
420 CALL HCHAR(POSR-2,POSV,32)
430 CALL HCHAR(POSR,POSV,I3)
440 GOTO 170
450 REM  DOWN
460 IF POSR=15 THEN 260
470 IF (H=4)*(POSR=11)THEN 260
480 CALL GCHAR(POSR+2,POSV,I3)
490 CALL HCHAR(POSR+2,POSV,32)
500 CALL HCHAR(POSR,POSV,I3)
510 GOTO 170
520 IF H=6 THEN 580
530 IF POSV=8 THEN 260
540 CALL GCHAR(POSR,POSV-2,I3)
550 CALL HCHAR(POSR,POSV-2,32)
560 CALL HCHAR(POSR,POSV,I3)
570 GOTO 170
580 M$=""
590 I5=5
600 FOR I=8 TO 18
```

```
610 CALL GCHAR(I5,I,I2)
620 M$=M$&CHR$(I2)
630 NEXT I
640 M$=SEG$(M$,3,9)&SEG$(M$,2,1)&SEG$(M$,1,1)
650 FOR I=8 TO 18
660 CALL HCHAR(I5,I,ASC(SEG$(M$,I-7,1)))
670 NEXT I
680 GOTO 170
690 IF H=6 THEN 750
700 IF POSV=14 THEN 260
710 CALL GCHAR(POSR,POSV+2,I3)
720 CALL HCHAR(POSR,POSV+2,32)
730 CALL HCHAR(POSR,POSV,I3)
740 GOTO 170
750 M$=""
760 I5=5
770 FOR I=8 TO 18
780 CALL GCHAR(I5,I,I2)
790 M$=M$&CHR$(I2)
800 NEXT I
810 M$=SEG$(M$,11,1)&SEG$(M$,2,1)&SEG$(M$,1,9)
820 FOR I=8 TO 18
830 CALL HCHAR(I5,I,ASC(SEG$(M$,I-7,1)))
840 NEXT I
850 GOTO 170
860 I5=7
870 M$=""
880 GOTO 600
890 I5=7
900 M$=""
910 GOTO 770
920 I5=9
930 M$=""
940 GOTO 600
950 I5=9
960 M$=""
970 GOTO 770
980 T$="PRESS APPROPRIATE KEY TO"
990 R=17
1000 VR=3
1010 GOSUB 2260
1020 T$="MOVE BLANK SQUARE"
1030 R=18
1040 VR=3
1050 GOSUB 2260
1060 IF V<5 THEN 1110
1070 T$="OR SLIDE ROWS
1080 R=19
1090 VR=3
1100 GOSUB 2260
1110 T$="1. UP      2. DOWN"
1120 R=20
1130 VR=3
```

```
1140 GOSUB 2260
1150 IF V>5 THEN 1210
1160 T$="3. LEFT   4. RIGHT"
1170 R=21
1180 VR=3
1190 GOSUB 2260
1200 RETURN
1210 T$="3.TOP<  4.TOP>  5.2<  6.2>"
1220 R=21
1230 VR=3
1240 GOSUB 2260
1250 T$="7.3<    8.3>    9.4<  0.4>"
1260 R=22
1270 VR=3
1280 GOSUB 2260
1290 T$="Q.5<    W.5>    E.6<  R.6>"
1300 R=23
1310 VR=3
1320 GOSUB 2260
1330 RETURN
1340 CALL CLEAR
1350 REM  DRAW
1360 FOR I=7 TO 7+2*V STEP 2
1370 FOR I2=4 TO 4+2*H STEP 2
1380 CALL HCHAR(I2,I,98)
1390 NEXT I2
1400 NEXT I
1410 FOR I=8 TO 6+2*V STEP 2
1420 FOR I2=4 TO 4+2*H STEP 2
1430 CALL HCHAR(I2,I,96)
1440 NEXT I2
1450 NEXT I
1460 FOR I=7 TO 7+2*V STEP 2
1470 FOR I2=5 TO 3+2*H STEP 2
1480 CALL HCHAR(I2,I,97)
1490 NEXT I2
1500 NEXT I
1510 IF V<5 THEN 1740
1520 FOR I=8 TO 18 STEP 2
1530 FOR I2=5 TO 15 STEP 2
1540 CALL HCHAR(I2,I,45+I/2)
1550 NEXT I2
1560 NEXT I
1570 CALL HCHAR(11,14,32)
1580 RANDOMIZE
1590 T$="   WAIT-RANDOMISING"
1600 R=17
1610 VR=3
1620 GOSUB 2260
1630 FOR Y=1 TO 32
1640 SCR=1
1650 KEY=RAN(14)+1
1660 GOSUB 170
1670 SCR=1
1680 KEY=RAN(2)+1
```

```
1690 GOSUB 170
1700 GOTO 1710
1710 NEXT Y
1720 SCR=0
1730 RETURN
1740 REM
1750 I3=65
1760 FOR I=5 TO 11 STEP 2
1770 FOR I2=8 TO 14 STEP 2
1780 CALL HCHAR(I,I2,I3,1)
1790 I3=I3+1
1800 NEXT I2
1810 NEXT I
1820 CALL HCHAR(11,14,32)
1830 REM
1840 T$=" WAIT-RANDOMIZING"
1850 R=17
1860 VR=3
1870 GOSUB 2260
1880 FOR Y=1 TO 55
1890 SCR=1
1900 KEY=RAN(4)+1
1910 GOSUB 170
1920 NEXT Y
1930 SCR=0
1940 RETURN
1950 PRINT "SQUARES":::"STEPHEN SHAW 1981":"PRESS '1' OR '2' FOR:"
1960 PRINT "1. 6X6 PROBLEM":"2. 4X4 PROBLEM"
1970 PRINT "FOLLOW DIRECTIONS AT BOTTOM   OF SCREEN AFTER
     DIAGRAM":"HAS BEEN DRAWN"
1980 CALL KEY(0,A,B)
1990 IF A=49 THEN 2050
2000 IF A=50 THEN 2010 ELSE 1980
2010 H=4
2020 V=4
2030 CALL CLEAR
2040 RETURN
2050 H=6
2060 V=6
2070 CALL CLEAR
2080 RETURN
2090 CALL CLEAR
2100 PRINT "SQUARES    1981":::"STEPHEN SHAW STOCKPORT":"THE
     OBJECT IS TO RESTORE "
2110 PRINT "THE ORIGINAL PATTERN OF ":"SQUARES,WHICH THE CO
     MPUTER":"HAS SCRAMBLED"
2120 PRINT "USING THE COMMANDS AVAILABLE.":"IN BOTH PUZZLES
     THE BLANK":"MOVES UP OR DOWN"
2130 PRINT "IN THE 6X6 PUZZLE":"IT CANNOT MOVE > OR < BUT":
     "THE WHOLE ROW SLIDES "
2140 PRINT "WATCH AS THE COMPUTER":"SCRAMBLES THE ORIGINAL"
     :"PATTERN TO SEE HOW IT":"WORKS!"
2150 PRINT "(BEING RANDOM YOU MAY END":"UP BACK AT THE START!)"
2160 PRINT "PRESS KEY P TO PLAY AGAIN":"WHEN YOU HAVE COMPL
```

```
    ETED";"YOUR PUZZLE"
2170 INPUT "PRESS ENTER":T$
2180 CALL CLEAR
2190 GOTO 1950
2200 CALL CLEAR
2210 CALL CHAR(96,"000000FF")
2220 CALL CHAR(97,"1010101010101010")
2230 CALL CHAR(98,"101010FF10101010")
2240 CALL SCREEN(12)
2250 RETURN
2260 FOR G=1 TO LEN(T$)
2270 CALL HCHAR(R,VR+G,ASC(SEG$(T$,G,1)))
2280 NEXT G
2290 RETURN
2300 I5=11
2310 M$=""
2320 GOTO 600
2330 I5=11
2340 M$=""
2350 GOTO 770
2360 I5=13
2370 M$=""
2380 GOTO 600
2390 I5=13
2400 M$=""
2410 GOTO 770
2420 I5=15
2430 M$=""
2440 GOTO 600
2450 I5=15
2460 M$=""
2470 GOTO 770
2480 END
```

3

How to use TI Basic

PRACTICAL PROGRAM WRITING

Before you switch your console on, to write a program, or even gather a large pile of paper to work on, sit and think about your proposed program.

Work out what your program is to do, and try to split it into small blocks of tasks to be accomplished. Then you can write the coding for each block, and check it to ensure you have not made any mistakes, before moving to the next block.

It is much easier to write a small program that works well than to write in one sitting a 15k program: it is improbable that it will work first time, and you will be faced with a lot of checking!

Some experienced programmers can just sit at their console and input a new program but, certainly at first, you should write your proposed program down on paper. Check the flow of the program before you input it: can any variable or input reach a value which would cause the computer problems? If so, is an 'error trap' required, or does the program need rethinking?

The Basic language is in some ways similar to English: the same task can be accomplished in several ways, but some ways are more efficient than others. A method which works well in one program may be inappropirate in another, therefore it is not possible to give any more than the most general guidelines.

A digital stopwatch can be an advantage when you are trying to find the quickest way of doing something. Usually a single process is too fast to time, but place it in a loop.

(A loop is a part of the program which is repeated several times, until a particular condition is met. In the following FOR . . . NEXT loop the condition is met when the variable I reaches a value of 1000).

```
FOR     I=1 TO 1000
CALL    HCHAR(3,4,45)
NEXT    I
```

and it is possible to obtain a reasonably accurate time to compare with other ways of doing the same thing — for instance, to place a single character at one screen position you may also use:

```
CALL    VCHAR(3,4,45)
```

— try substitution in the above loop and see the difference in execution time.

In general the 99/4A is slow at reading DATA and at scrolling the screen (PRINT), but there are occasions when these are the best commands to use.

It is faster for instance to PRINT 24 lines than to use CALL HCHAR 768 times (the number of characters on the screen)!

Your program idea may need to be amended to meet with the demands of the computer, and you should never be afraid of completely scrapping your work and starting again: often new inspiration can lead to a far more efficient program.

Before developing some useful routines, a word about editing and debugging: no matter how good a typist you are, even entering a short program will need the use of these facilities.

Editing is what you do when you change a program line —perhaps only one character in the line.

A number of computers use 'screen editing', where you move the cursor around the screen until it is placed where you want to make your alteration.

The TI99/4A uses a 'line editor' for programs (some modules also use a screen editor, eg TI-WRITER).To use a line editor, you first select the line number you wish to amend, place it on the bottom of the screen, and move the cursor along the LINE until you hit the place to be amended.

To bring the line you want on screen, key in the line number, then hold FCTN down and press key E or X. Your line will appear with the cursor at the beginning of the line.

Use FCTN and keys S and D to move the cursor over the line without deleting or altering anything.

If you wish merely to alter the line, typing over it may be sufficient, but the 99/4A also allows you to delete and insert characters.

To DELETE a character, place the cursor over it (FCTN plus S or D) and then press FCTN and key 1. This will delete the character the cursor is on, and everything following will move one space to the left.

To INSERT text, place the cursor after the last character you wish to leave untouched, using FCTN and S & D, then press FCTN and key 2. Now anything you type will force everything after the insert point one character to the right, and your inserted text will appear. The cursor position also moves to the right as you type. You leave INSERT mode by pressing FCTN and S or D, or by pressing ENTER.

When your line is correct, press ENTER to enter the new line into memory, and leave EDIT mode. However, if you also wish to amend the line before or after the line just finished, you may move directly to that by pressing FCTN and E or X.

The maximum line length in TI Basic is four screen lines, but the computer is often capable of taking a longer line, as the absolute restriction is on the length of the line in BYTES of memory used, not the number of character on the screen.

You may use the edit function to insert 'overlong' lines as follows:

Type in the last part of the required line first. Press ENTER.

With the line on screen, press FCTN and key 2, then key in the first part of your line, from the beginning. The part you first entered is pushed to the right.

Now return the line to the screen using the edit mode: type in the line number and then press FCTN and key X.

It is possible to overfill a line this way, and you will receive an error message if you do. However you will usually be able to go to an extra half screen line, and in some cases you may be able to squeeze in two extra lines!

The advantage of putting as much as possible in a line is that by using less program lines you save a little memory. In general, lines with a lot of numbers in will be difficult to expand in this fashion, but lines with a lot of text or commands can usually be considerably extended.

When the line is fully entered, press ENTER. The LIST the line just to make sure it is all right.

You do not have to INSERT at the beginning of the line, but you may find it easier to do so. Give the computer time to move all the characters to the right when using INSERT. Keep your eyes on the screen.

A brief word of warning: in TI BASIC, your program uses the same area of memory as the values of variables, separated by a 'marker' in memory. As a program runs, the free memory is continually filling. Unwanted values are only purged when the memory is full, resulting in short pauses in program operation.

If the variable area of memory is almost full when you stop to edit a program, inserting extra material MAY result in the permanent loss of the marker which marks the limit of the actual program. This will cause irreversible damage to your program and may prevent the LIST function from operating correctly, or cause a system lockout when you try to run the program.

Therefore try to avoid running the program and then editing it! Before you edit, take a copy of the program to tape — this has the added benefit of apparently clearing

the garbage, and not only gives you a security copy but actually prevents the problem occuring!

This problem will not be apparent if you use extended basic plus the 32k ram expansion, as variables then occupy a different sector of memory.

DEBUGGING

Having entered your program, you type in RUN, and instead of the program running, you receive an error message. DEBUGGING is required.

A BUG is quite simply an error in the program, either a mistype or an error in your use of Basic.

The computer does check the lines you enter, but will only spot such things as using only a single bracket (or quotation mark ".

When you RUN your program, the computer first goes through your program and sets aside memory for each variable and sub program that you have used. During this 'prescan' further errors may be spotted and an error message printed on the screen.

Typical errors spotted at this time are incorrect use of arrays (trying to use DIM after the variable has been used) or a mismatched number of FORs and NEXTs.

Most errors will only produce an error message as the computer finds them when your program is actually running — for instance trying to GOTO a non existent line number, or trying to RETURN when there is no outstanding GOSUB.

The error messages generated by the TI99/4A are well

described in the manual, and will usually indicate a line number in which the computer has met something it cannot cope with.

Unfortunately, the actual error may not be in the line stated in the error message.

For instance, BAD VALUE IN 100 may refer to:

```
100 CALL HCHAR(ROW,COL,42)
```

The program line is correct. The error message has appeared because one or both of the variables ROW and COL have a value which is out of range.

(Using CALL HCHAR the ROW and COLUMN values passed to the sub program must be in the ranges 1 to 24 and 1 to 32 respectively. If you go outside this range the computer will halt with an error message.)

To see what the values are, when the error message appears, just type in PRINT ROW;COL then press ENTER.

The two values on screen will be the current values of ROW and COL, which have caused the problem.

The job is then to review the program to see how the variables obtained that value, and see what changes need to be made.

This ability to check variable values after an error message is very valuable. Note that once you amend a program line all variables will be reset to zero.

Frequent causes of problem bugs are DATA statements, with a comma too many or too few. The ability to check variable values is useful here:

An array is being filled from a DATA statement, eg

```
FOR I=1 TO 5
READ A
VAR(I)=A
NEXT I
DATA 23,54,8,A,5
```

An error message will be generated by the above, as 'A' is not a number. When the error message appears it is possible to enter:

```
PRINT I;VAR(I)
```

This will provide the clues needed to lead to the erroneous DATA line — which may be several hundred program lines away!

In such cases you need to have a good idea of what values should be found in connection with each variable, and you may need to spend some time working through the program.

If you cannot fathom why a program is not working as it should merely by reading the LIST, the TI99/4A also has a TRACE option.

Key in TRACE and then RUN. The line numbers will be listed on screen as the program progresses, and you can watch for an unexpected line transfer as the computer moves from one line to the next. Using TRACE will disrupt any screen display. Switch TRACE off by keying in UNTRACE.

To find out what the variable values are at a particular point in the program, you can insert PRINT statements in the program, or instruct the program to BREAK by

adding a line in the appropriate place with the instruction: BREAK.

Then when the program stops you may enter PRINT VAR etc.

To continue the program enter CON (and press ENTER). Remember to remove the BREAK line when you no longer need it.

As your programs become longer, so it becomes more difficult to spot the errors, but finding and removing errors is a very good (if time consuming) way of learning how to use your TI99/4A. With time and experience you will learn to quickly spot the easy bugs and to tackle the harder ones in a logical fashion. Read your manual as often as necessary, especially the section on ERROR MESSAGES. Often the answer to a difficult bug is there just waiting for you to read it.

SYSTEM LOCK OUTS

A lock out has occurred if your console no longer responds to the keyboard (especially QUIT) and ceases to function normally. Unusual sound and graphic effects may occur.

It is quite normal for all computers (and word processors) to lock out from time to time. The cause is an error in the instructions passed to the processor, which it cannot deal with. There are a number of causes:

A STATIC discharge is a frequent cause of problems. Although computers no longer need the carefully controlled environment of the mainframe, they remain sensitive to static. The problem is most acute in warm dry weather, or if you wear clothing made of artificial fibres

(acrylics are particularly bad). Nylon carpeting can also be a problem. The TI99/4A can handle static quite well, but on occasion you may meet the problem.

Use of cotton clothing, a humidifier, and an anti static spray on the carpet may be called for in especially hostile environments. A conductive carpet is also sold by some computer suppliers.

Poor communication with modules or peripherals may also be a problem: the contacts are essentially self cleaning, but it may be necessary to disconnect/connect a few times to make good contact. Contacts are silver plated and are subject to tarnishing, and may require this treatment if a module or peripheral is not used for some time. Tar can be deposited on the contacts if there are smokers in the room. In extremely severe cases of pollution, an isopropyl alcohol solvent may be used but great care is required to prevent damage.

Some modules contain insufficient error traps and permit you to pass confusing instructions to the processor. Extended Basic in particular allows a number of lock out producing errors. If your syntax is correct, this will not happen.

Loss of the stack/program marker can cause problems in TI BASIC. This occurs when you run a program (filling the stack) and then add to the program. In some cases the computer will add the stack to the program with sometimes colourful results, but permanently destroying the program (if a lock out does not occur). After running a program, it is wise to save it before editing: this appears to clear the stack.

When the computer ceases to function, and possibly makes a piercing sound, do not panic! The only way out of a lock out is to switch off and (after a few seconds)

restart. Even the MOST expensive systems sometimes lock out.

Just as with English, your ability to use BASIC increases with use. Examine as many 99/4A programs as you can and look to see

i) WHAT each part of the program does
ii) HOW it does it
iii) WHY it does it.

Then try to improve the program!

This simple graphics demonstration program uses several of the features of TI BASIC described in the preceding section. The descriptive text which follows will help you to follow the program.

To enter this program, select TI BASIC, then ENTER the word NUM. This will automatically provide the line numbers and you will only have to type in the remainder of the lines.

```
100 REM DEMO OF COLOUR
110 REM IN TI BASIC
120 REM
130 RANDOMIZE
140 PRINT " ":::::"ONE MOMENT...":::::::
150 DIM F(100)
160 DEF RAN(A)=INT(A*RND)+1
170 M$="0"
180 FOR CHARR=32 TO 152 STEP 8
190 CALL CHAR(CHARR,M$)
200 NEXT CHARR
210 FOR SET=1 TO 16
220 CALL COLOR(SET,16,SET)
230 CALL VCHAR(1,2*SET-1,24+SET*8,48)
240 NEXT SET
250 FOR SET=1 TO 9
260 CALL HCHAR(22,2*SET,48+SET)
270 NEXT SET
280 CALL HCHAR(22,20,48)
290 FOR SET=11 TO 16
300 CALL HCHAR(22,2*SET,38+SET)
310 NEXT SET
320 FOR SET=10 TO 16
330 CALL HCHAR(22,2*SET-1,49)
```

```
340 NEXT SET
350 FOR X=1 TO 1000
360 NEXT X
370 FOR SET=3 TO 14
380 CALL HCHAR(2*SET-5,1,24+SET*8,32)
390 NEXT SET
400 FOR X=1 TO 4000
410 NEXT X
420 INPUT " ENTER TO CONTINUE":A$
430 CALL CLEAR
440 CALL SCREEN(12)
450 FOR X=1 TO 150
460 R=RAN(24)
470 VR=RAN(32)
480 SET=RAN(16)
490 CALL HCHAR(R,VR,24+SET*8)
500 NEXT X
510 FOR X=1 TO 50
520 R=RAN(24)
530 VR=RAN(32)
540 SET=24+8*RAN(16)
550 NO=RAN(32-VR)
560 CALL HCHAR(R,VR,SET,NO)
570 CALL HCHAR(25-R,33-VR,24+8*RAN(14),NO)
580 CALL VCHAR(1+VR/2,R/1.5+1,SET,3/4*NO)
590 NEXT X
600 FOR COUNT=32 TO 152 STEP 8
610 CALL CHAR(COUNT,"88CCEEFF88CCEEFF")
620 CALL CHAR(COUNT+1,"80C0E0F0F8FCFEFF")
630 CALL CHAR(COUNT+2,"7F3F1F0F070301")
640 CALL CHAR(COUNT+3,"FF3C18181818 3CFF")
650 CALL CHAR(COUNT+4,"00C3E7E7E7E7C3")
660 CALL CHAR(COUNT+5,"FFFEFCF8F0E0C080")
670 CALL CHAR(COUNT+6,"00010307 0F1F3F7F")
680 CALL CHAR(COUNT+7,"183C7EFFFF7E3C18")
690 NEXT COUNT
700 FOR COUNT=1 TO 100
710 CALL HCHAR(RAN(24),RAN(31),RAN(129)+31)
720 NEXT COUNT
730 X=110
740 K=2^(1/50)
750 FOR COUNT=1 TO 100
760 F(COUNT)=X*K^COUNT
770 NEXT COUNT
780 FOR COUNT=1 TO 100
790 CALL SOUND(100,F(RAN(100)),1)
800 CALL COLOR(RAN(16),RAN(16),RAN(16))
810 IF RAN(10)>3 THEN 830
820 CALL SCREEN(RAN(16))
830 NEXT COUNT
840 FOR Z=1 TO 5
850 FOR COUNT=3 TO 30
860 CALL VCHAR(8,COUNT,RAN(129)+31,2)
870 CALL VCHAR(5,COUNT,RAN(129)+31,2)
880 CALL SOUND(-100,F(RAN(100)),1)
890 NEXT COUNT
```

```
 900 FOR COUNT=30 TO 3 STEP -1
 910 CALL COLOR(RAN(16),RAN(16),RAN(16))
 920 CALL VCHAR(12,COUNT,RAN(127)+32,2)
 930 CALL VCHAR(15,COUNT,RAN(127)+32,2)
 940 CALL SOUND(-110,F(RAN(100)),1)
 950 NEXT COUNT
 960 NEXT Z
 970 FOR Z=1 TO 26
 980 FOR X=1 TO 28
 990 M$=CHR$(RAN(127)+32)&M$
1000 NEXT X
1010 PRINT M$
1020 M$=""
1030 CALL SOUND(-400,F(RAN(100)),1)
1040 NEXT Z
1050 FOR X=1 TO 100
1060 CALL COLOR(RAN(16),RAN(16),RAN(16))
1070 NEXT X
1080 M$="1234567809ABCDEF"
1090 X=0
1100 FOR Z=1 TO 8
1110 FOR CT2=1 TO 16
1120 P$=P$&SEG$(M$,RAN(16),1)
1130 CALL SOUND(-100,RAN(200)+110,RAN(10))
1140 NEXT CT2
1150 FOR CT=1 TO 8
1160 CALL CHAR(8*CT+24+X,P$)
1170 NEXT CT
1180 X=X+1
1190 P$=""
1200 NEXT Z
1220 CALL SCREEN(RAN(16))
1230 FOR X=1 TO 100
1240 CALL HCHAR(RAN(24),RAN(32),RAN(127)+32)
1250 CALL SOUND(-50,F(RAN(100)),RAN(20))
1260 CALL COLOR(RAN(16),RAN(16),RAN(16))
1270 NEXT X
1280 CALL SCREEN(RAN(16))
1290 FOR X=1 TO 100
1300 CALL COLOR(RAN(16),RAN(16),RAN(16))
1310 CALL SOUND(-20,110+20*RAN(40),3)
1320 NEXT X
1330 CALL SCREEN(RAN(16))
1340 M$=""
1350 FOR Z=1 TO 26
1360 FOR CT=1 TO 28
1370 M$=CHR$(RAN(127)+32)&M$
1380 NEXT CT
1390 PRINT M$
1400 M$=""
1410 CALL SCREEN(RAN(16))
1420 CALL COLOR(RAN(16),RAN(16),RAN(16))
1430 NEXT Z
1440 FOR CT2=1 TO 100
1450 CALL COLOR(RAN(16),RAN(16),RAN(16))
1460 NEXT CT2
```

```
1470 CALL SCREEN(RAN(16))
1480 STOP
```

The use of brackets after RAN in line 160:

DEF RAN(X)=....

does NOT indicate that RAN is a variable array. They inform the computer that when the newly created function RAN is used in a program, a variable will be passed to the definition by means of brackets.

e.g. 460 R=RAN(24)

The computer replaces RAN(24) with the defined statement, using the value 24 in place of X.

The use of brackets does not always imply the use of an array.

COLOUR DEMONSTRATION PROGRAM

This program has been provided to give a practical demonstration of some of the features of TI BASIC described earlier.

The program has been written in small blocks, and each block will be described separately.

The first section, lines 100 to 420 form the start of the program. The first block is intended to display the colours available, and by making them cross over each other, show the relative contrasts.

Because some random patterns are created later, RANDOMIZE has been used to provide different patterns

each time the program is run. The array F contains frequency values for use with CALL SOUND later on. As it will contain 100 values, DIM is used to instruct the computer to allocate memory to hold the values.

The DEF function is used to create a new random function, which will provide integer (eg no fraction) numbers from 1 to the figure used with the new function in the program: watch out for the new function RAN(X) in the program.

M$ has been set to "0" (a string with a zero in it) for use in defining all the characters as blanks (eg spaces) in the following loop in lines 180 to 200. The string "0" could have been placed in the definition function in line 190, instead of the string variable.

NB: Although the TI99/4A distinguishes the number Ø from the letter O on screen by squaring the O, in listings a slashed Ø usually represents the number.

Each group of eight characters is in a separate character set, and each set may be a different colour. Lines 210 to 240 change the colour of each set: TI Basic has 16 sets, and 16 colours. The foreground colour of every set has been set to WHITE (code 16).

Line 230 places a number of vertical stripes on the screen, each a different colour. As the screen is 32 columns wide, the stripes have been set to 2 columns wide each, and therefore each CALL VCHAR uses 48 characters (2 x the 24 rows).

So that we know which colours are which, they are labelled by the routine in lines 250 to 340.

Lines 350-360 give a small delay.

It is not possible to provide 16 horizontal bands with a space between each, as the screen only has 24 rows, but lines 370 to 390 cross the screen with as many rows as can fit.

In lines 430 to 500, random characters are placed on a blank screen, and in lines 510 to 590, random stripes and columns are placed on screen.

Lines 600 to 690 redefine the characters for the purposes of the following sections of the program. Remember that the strings used with CALL CHAR can only contain the numbers 0 to 9 and the letters A to F.

700 to 720 again place random characters on screen.

Lines 730 to 770 fill the F array with values to be used with subsequent CALL SOUNDs. Line 730 sets the lowest possible frequency, and line 740 sets the basis for the tones: the formula used creates what is known as 'microtonal' music, with very little difference between adjacent tones.

Then, accompanied with random tones, the character colours are varied at random in lines 780 to 830.

Lines 840 to 960 provide sound, colour changes, and small bars of random characters.

Lines 970 to 1040 use the PRINT routine to place random characters on screen, and the colour is varied in 1050 to 1070.

In lines 1080 to 1200, characters are given a random definition and in 1220 to 1260, the colours are varied, random characters placed and tones generated.

1270 to 1310 again varies the colours, and uses a

different random tone generation method (line 1300).

The remainder of the program again uses PRINT to provide a random display and the colours are varied.

Note in particular the usefulness of the DEF statement in this program.

There are many 'loops', and some loops contain other loops: see for example lines 840 to 960. These loops are 'nested', with the COUNT loop inside the Z loop.

An ARRAY is used to store frequencies.

The loop counters (eg CHARR, SET and so on) also function as numeric variables in the loops. Their value increases by one for each cycle of the loop until the maximum value (set by the TO X in FOR TO NEXT) has been reached.

In line 1120, SEG$ is used with the string variable M$ (set in 1080) to create a random definition of a character. A letter is chosen at random from the string variable M$, and used to create the definition in the variable P$.

Note that P$ is reset to a 'nul' (empty) string after each character has been defined. Then it is reused with defferent letters to form a new definition.

TI BASIC GLOSSARY

A list of all the commands and functions available in TI BASIC, with brief descriptions.

ABS(X) Used to obtain an ABSolute value — eg it ignores that + or - sign.

APPEND Used to extend files when using the disk system or mini memory module.

ATN(X) Provides the arctangent: this is a trigonometrical function. ATN (X) provides the angle in radians whose tangent is X.

BREAK Used in a program to temporarily halt execution. A message is placed on screen. The program is continued by typing in CONTINUE

BYE Used to return to the master screen. Preferable to the use of the QUIT key, as only BYE will properly close any open files, and ensure no data is lost.

CALL CHAR(N,STRING$)
Used to define character N by means of a hexadecimal string STRING$.

CALL CLEAR Used to clear the screen.

CALL COLOR(SET, FG,BG)
Used to define the foreground (FG) and background (BG) colours of each of 16 sets of characters. Sixteen colours, including transparent are available.

CALL GCHAR(ROW,COL,CH)
Places in the variable CH the ASCII code of the character on the screen in ROW, COLumn.

CALL HCHAR(ROW,COL,CH,NO)
Used to display a character CH in ROW, COLumn. When a number or variable in the position NO is used, the character is repeated horizontally NO times.

CALL JOYST (NO,X,Y)
Returns X and Y values for Joystick NO.

CALL KEY (NO,CODE, STATUS)
Interrogates the keyboard. If a key is depressed the ASCII code is placed in variable CODE. STATUS indicates if a key is pressed, and if it is the same key when two adjacent CALL KEYs detect a key pressed. NO indicates the key unit, and is used to split the keyboard and to switch the codes returned by the Function and Control keys.

CALL SCREEN(NUMB)
Used to change the colour of the screen.

CALL SOUND(TIME,F1,V1,F2,V2,F3,V3,N,V4)
Used to generate sound. Up to three tones may be used with an optional noise channel. TIME is in milliseconds and F1,F2 and F3 are the frequencies in cycles per second.

CALL VCHAR(ROW,COL,CH,NO)
As CALL HCHAR, but the character CH is repeated vertically NO times.

CHR$(CODE) Used to make a character CODE available as a string.

CLOSE Used to close a data file.

CONTINUE — Used to resume a program when execution has been halted with the CLEAR key or a BREAK command.

COS(X) — Provides the cosine of angle X, where X is in radians.

DATA — Used as a heading on lines containing values to be READ.

DEF — Used to DEFine a function of your own.

DELETE — Used with the disk system to DELETE a file.

DIM — Used to DIMension an Array.

DISPLAY
i. Same effect as PRINT
ii. One of two file storage formats. DISPLAY uses the same codes and format as the computer uses for screen displays.

EDIT — One method of entering EDIT mode.

END — An optional marker for the end of your program.

EOF — 'End Of File' used with disk files.

EXP(NO) — The inverse of the natural logarithm function LOG. Thus X=EXP(LOG(X))

FIXED — Used to define data files. The alternative, not available with cassettes, is VARIABLE.

FOR...TO..(STEP)
Used to establish loops which execute until the counter reaches the value following TO.

GOSUB

Used for a line transfer when the program is to RETURN to the line following the GOSUB line after the section transferred to is completed.

GOTO — Used to make a simple line transfer.

IF...THEN...ELSE

Used to make conditional line transfers, with an optional alternative transfer if ELSE is used.

INPUT

i. Used to fill a variable from the keyboard, or other device if a data file is used.
ii. Used to specify a file is to be used for INPUT only.

INT(NO) — Used to provide the INTeger of a number NO. eg any fraction is removed.

INTERNAL — Used to specify the format of a data file. INTERNAL specifies the codes used by the processor internally. The alternative is DISPLAY.

LEN(STRING$)

Used to provide the LENgth of the string STRING$.

LET — Optional. LET A=2 and A=2 are both accepted.

LIST — Lists a program on screen or other device.

LOG(NO) — Provides the natural logarithm of number NO.

NEW — Used to clear the console memory in preparation for a new program.

NUM / NUMBER
Provides line numbers automatically when a program is to be keyed in. Starting number and increment may be defined. Default is to start at 100 and increase each line number by 10.

OLD — Used to load a program from cassette or other storage device.

ON . . . GOSUB
Used for GOSUB transfers when the value following ON determines which of the line numbers following GOSUB are to be used.

ON . . . GOTO — Similar to the above, but for simple line transfers when it is not wished to RETURN to the line following the transfer.

OPEN — Used to OPEN a file to a device. The format of the file is specified after the OPEN command.

OPTION BASE
Used to set the minimum value of an array to zero or one.

OUTPUT — Used to specify a data file is to be used for output only.

POS(STRING$,X$,NO)
Used to obtain the first occurrence in STRING$ of X$.

PRINT — i. Used to display characters on the

screen and to scroll the screen upwards.

ii. Used to send data to an external storage device when used with files.

RND — Used to obtain a RaNDom number. The same sequence of numbers is generated every time a program is RUN.

RANDOMIZE — Used to set the initial number used by RND to be different each time a program is RUN.

READ — Used to place the values in DATA statements into variables.

REC — Used with the disk system to specify a specific RECord in the file.

RELATIVE — Used with the disk system for random access to data on a file by using RECord numbers.

REM — Used to add REMarks to programs.

RES/RESEQUENCE — Used to resequence a program, that is, change the line numbers. Default is to start the program at line 100 and use increments of 10, but start number and increment may be specified.

RESTORE — Used to reset the DATA pointer, either to the first data item, or to data on a particular line.

RETURN — Any line transfer by GOSUB must be terminated with RETURN. The program

then transfers to the line following the GOSUB.

RUN — Instructs the computer to RUN the program in its memory.

SAVE — Used to SAVE a PROGRAM to a storage device.

SEQUENTIAL — Used to specify that file data is recorded in the order it is output. The alternative is RELATIVE, which is not available with cassette files.

SGN(NO) — Used to obtain the SiGN of the number NO. Indicates if NO is zero, negative or positive.

SIN(X) — Provides the SINe of the angle X where X is in radians.

SQR(X) — Provides the SQuare Root of the number X.

STOP — Used to STOP program execution. A STOPped program may only be resumed by re-RUNning it.

STR$(NO) — Used to change a number or numeric variable into a string. The converse is VAL.

TAB — Used when PRINTing to print at from a specified column.

TAN(X) — Provides the TANgent of angle X where X is in radians.

TRACE	Causes the computer to list on screen the number of each line as a program is executed. Switched off with UNTRACE.
UNBREAK	Used in command mode to remove BREAK commands which have been placed in command mode.
UNTRACE	Switches TRACE off.
UPDATE	Used with disk file processing to enable a file to be read or written to.
VAL	Used to obtain the numeric equivalent to a string. The string must be composed only of valid numbers.
VARIABLE	Used with disk files to indicate the file is to be as long as the item to be stored, which may vary.

4

Cassette Handling

USING CASSETTE RECORDERS WITH YOUR TI99/4A

Standard audio recorders provide an inexpensive method to store your programs and data.

Due to modifications in the design of audio tape recorders, an increasing number of them can be difficult to use with computers, and TI have therefore produced a TI Recorder, designed for use with the computer.

You may be able to use other recorders, but before purchase you should try them with program tapes recorded by someone else.

For ease of operation, the tape recorder should have a tape counter, and if you buy one with a tone control you may find it easier to use.

The TI99/4A is designed for use with recorders having 3.5mm jack sockets. Recorders with 5 pin DIN sockets may be unsuitable due to the different input and output levels of these machines.

Always use a mains power supply for your cassette player, to ensure the tape runs at the correct speed.

The magnetic particles on your tapes will pass on some

of their magnetism to your tape recorder heads : this in turn will slowly wipe off your program. Regular use of a tape head cleaner and demagnetiser is strongly recommended, usually a monthly clean and demagnetisation are sufficient.

It is better to demagnetise before you notice any problems. Similarly a build up of oxides will prevent your recorder from picking up a proper signal, and may cause it to digest your tape.

Connecting your Recorder

You will usually be using only one tape recorder, and this is connected using the lead with two 3.5mm jack plugs and one 2.5mm jack plug. Plug the smaller plug into your recorder's remote control, the plug with the red wire into the microphone socket, and the plug with the white wire into the earpiece socket. The other end of the cable is a nine pin plug. This is connected to the socket on the rear of the console : NOT the socket on the left, which is for the joystick.

When the leads have been connected, press your cassette PLAY button and check to see if the motor is running (the computer console must be switched on). If the motor is silent, you will need to use the polarity reversal adaptor supplied with the cassette lead. This should ensure that your recorder now works. If you experience difficulty, refer to your dealer.

You are now ready to record a program, or to load a commercial tape.

Loading a cassette tape:

When your console has been connected to your tv and tape recorder, and switched on, you may load a

prerecorded program from tape.

Press a key to obtain the menu selection and then press key 1 to obtain TI BASIC READY.

Now make sure the ALPHA LOCK key is DOWN, and key in:

OLD CSI and press ENTER.

The computer will now give you instructions on what to do: First rewind the tape and then press ENTER.

This is a good point at which to set your volume and, if you have them, tone controls. Start with the volume control at the mid point, and if available, set the tone control at maximum treble.

Now press cassette PLAY and then press ENTER. The tape should start running (do you need the polarity reverser? see above). The first part of the tape is a pilot tone, which is followed by a fluctuating signal. You may obtain an error message at this stage, which will indicate that the volume is not set correctly. NO DATA FOUND means that the volume is far too high, or far too low (or the cassette recorder isn't connected).

ERROR IN DATA means the volume is not quite right. Make a note of the tape counter setting when the tape stopped, then press R (to Read) and follow the screen instructions. When the tape is rewound, before you press PLAY again, make a small change in the volume setting, then try again.

If you are again unlucky, see if the tape counter is now reading a higher or lower figure. If it is higher, you need to move the volume a little more in the same direction. If lower, move it in the opposite direction — then try again.

Tape recorders vary in their success at matching the requirements of the computer: on some it hardly seems to matter how you set the volume, but on others the tape will only load at one setting.

Tapes recorded by someone else will usually need to be loaded at a higher volume than your own, and may have a narrower range of acceptable volumes. The reason for this difficulty is that tape recorders do not all have their heads at exactly the same level, and with a tape recorded on another machine, some of the signal will not be picked up on your recorder. So you need a higher volume setting to make up for this loss.

With a very few recorders, you may find that your tape is running at a speed which varies from that of the machine the tape was recorded on, preventing the program loading. This is a very rare problem but you may come across it.

When the tape is loaded, the computer will instruct you to press cassette STOP then press ENTER. You will receive a DATA OK message and after a short pause a flashing square will appear : this is the cursor, and indicates that the computer is ready for your instructions.

You will want to RUN the program, so type in RUN and press ENTER. The computer will take a few seconds to sort itself out but in due course the program will be available for you to play.

When you only have the console, you may only run programs in TI Basic. Trying to load or run programs in any other language (such as Extended Basic) will produce error messages.

Saving programs to tape

When you have keyed in a program you may wish to save it on tape, for future use. Otherwise you will have to key it all in again. Use cassette tapes of maximum length C60. C90 tapes and longer are thinner and liable to stretch slightly in use, distorting the data the computer needs. Many large newsagents now sell computer tapes of C10, C15 or C20 length. A C20 tape will hold three average length programs on each side.

If you are keying in a long program, it is a good idea to save your work from time to time, even though it is not finished, then if some disaster occurs (such as baby brother unplugging your console) you don't have to start all over again : you may load your work and carry on from there.

When you are ready to save a program, type in SAVE CS1 (with the alpha lock key down) and follow the directions which will appear on the tv screen. When the computer has finished the recording, you will be asked if you wish the tape to be checked : if your program is worth recording, it is worth verifying the recording — always respond Y (yes) you wish the tape to be checked. Then follow screen directions.

This part of the procedure is the same as loading a tape, but the computer will be checking the tape data against what it has in memory. See the section on loading programs for further assistance.

The reason you should always check your recording is that even the best tapes have small section with little or no oxide coating, causing what are termed 'drop outs'. Very short duration drop outs will not affect music, but you will lose some data. If you find that your recording is unloadable for any reason, the program is still in the

computer, and you may use SAVE CS1 again to try a different tape.

Please remember that tapes are relatively fragile so do not keep them near to any strong electromagnetic field. In particular, it is unwise to place a tape near to the console, the tv modulator, or your tv set, for any length of time.

As tapes can be damaged so easily, always make two copies of your program on separate tapes. If one is damaged you can then make a further backup from the second tape.

A magnetic tape has a limited life and you should rerecord onto another tape if a tape begins to become difficult to load.

Further notes on use of save:

The keys R C & E are used when an error has been found, to R(ead) or R(ecord), C(heck), or E(xit). These keys are also active whenever the current instruction is 'THEN PRESS ENTER'. Be careful not to press them unless you wish to Exit the routine.

When you have finished recording a program, and the display asks you to 'PRESS CASSETTE STOP THEN PRESS ENTER', instead of pressing ENTER and moving on to the verify routine, you may press R to immediately record the program again, say on another tape.

If you wish to save a program, and find on looking at the screen, the message 'PRESS CASSETTE PLAY THEN PRESS ENTER', you will realise that instead of SAVE CS1, you have typed OLD CS1, and you are well on your way to destroying all your hard work. You can escape from this problem by pressing E. An error message may appear, but your program should be safe.

5

File Processing

PROGRAMS are saved to tape using SAVE CS1, and loaded back into the computer with OLD CS1 (if you are using a tape recorder).

The values of the program variables are not saved with the program, and will initially have a value of zero when the program is loaded.

If you wish to save a high score, or other data, you must use file processing operations as described below.

Tape files can take a few minutes to load, but form an inexpensive introduction for the novice programmer to the field of data management. Fortunately, TI have provided a fairly simple operating system.

Before you can save or load data, you must 'open a file'. This operation describes to the computer the format of the data on your tape, and allows you to refer to the tape by a simple label, called the file number.

When the computer comes to an OPEN statement, a section of tape is wound on, with no signal. This is to clear any leader on the tape. You will find it easier if you always start your data files at the beginning of the tape, then you can be sure the computer will find the data where it expects it to be.

A cassette file may be to load data or to save it, and this is defined in the opening statement. The usual cassette operation instructions are given when the OPEN statement is operated on. Please keep this in mind and ensure that you do not open a file after a complex screen display has been created!

The OPEN statement is in the form :

OPEN #N:"CS1",INTERNAL or
DISPLAY,INPUT or OUTPUT, FIXED NUM

The number N must be a number and not a variable, and lie between 1 and 255. It is possible to have several files open at once (eg to CS2, a printer, the speech synthesiser if using Terminal Emulator 2 module etc), and the file number is used to instruct the computer which peripheral it is to use.

The choice between INTERNAL and DISPLAY is one of coding : in DISPLAY format, the computer data is recorded to printable ASCI code. DISPLAY format can sometimes be of advantage in DISK processing when greater speed is sometimes possible, but for TAPE files, it uses more tape and also requires more complex program writing.

Using TAPE files, use the INTERNAL format.

File organisation with tape files is always SEQUENTIAL and this word is not required in the opening statement. This means that the second item of data read from the tape is the second item of data which has been recorded on the tape. It is not possible to 'skip' to say the 6th item of data without reading everything in between.

As mentioned, you MUST indicate if the file is for INPUT or OUTPUT. You use INPUT to read data from a tape, and OUTPUT to place data onto the tape.

You should only have one file open to CS1 at a time, otherwise the computer will lose track of the data on the tape.

FIXED means that every time the computer writes data to the tape, the same amount of tape is used no matter how long the data. Any unused tape is filled with nuls. If no number follows the word FIXED, the data field is 64 bytes long. A number always occupies 9 bytes, and a string one byte more than the string has characters.

To save data once the file has been opened, you use the PRINT command, but add the file identification: PRINT #1:

There is ALWAYS a colon (:) after the file number.

To save a single variable value, you would use: PRINT #1:A

To make the most of the 64 byte field length, you may save 7 variables at once, each separated in the PRINT statement by a semi colon:

```
PRINT #1:A;B;C;D;E;F;G
```

Strings are saved in a similar way:

```
PRINT #1:A$;B$
```

or

```
PRINT #1:"THIS STRING IS TO BE SAVED"
```

If the standard field length of 64 bytes is too short, you may specify one of two alternative field lengths, 128 or 192. You do this by placing these numbers after the word FIXED (with a space between).

When you have finished saving data to the tape, remember to close the file with CLOSE #1. This will generate the message "PRESS CASSETTE STOP & PRESS ENTER", so as before ensure that there is no screen display to be disrupted.

When the computer is saving data, it does so one field at a time, and it is essential the computer has control of the cassette motor. If your remote control does not function, fit the polarity reverser (provided with the TI Tape Cable) between the remote socket on your tape recorder (the small one: 2.5mm) and the small jack plug on the lead from the computer. If you have purchased a third party lead and require the polarity reversing, you should ask your supplier to do so. It is possible to swop the wires yourself, but with some systems there may be problems as the screening may be broken.

To load your saved data, you must reopen the file, this time specified as an INPUT file. If you used long field lengths when saving the data, you MUST specify the same length in the OPEN statement to load the data. Similarly if a file is saved in INTERNAL format, you MUST read it in the same format.

If you have placed several values in one field, you must read them in the same way — if you have used:

```
PRINT #1:A;B;C;D
```

then you MUST read four NUMERIC variables, although the names may differ:

```
INPUT #1:Z;X;R;T
```

If a numeric variable has been saved, you must read a numeric variable. A field containing string data must be input to a string variable.

Your tape files contain the values of the variables printed to them (or direct numbers or strings). The variable name is not saved, and the variable is not affected by saving its value.

Have a look at the file management program in Vince Apps book of Programs for the 99/4A for an example of using tape files.

Remember to CLOSE the file when you have finished with it.

Tape files can occupy large amounts of tape, and if you are saving a large number of items, you will need at least a C15 or C20 tape.

If you graduate on to DISK BASED filing, the 99/4A offers many more options, to give you greater control and access. This includes variable length files for better use of disk space, update mode which allows you to read and write to a single open file, and relative files which allow you to read or write to a specific record within the file. The disk system also allows the use of named files, and of course greater speed.

You may use the MINI MEMORY MODULE as a file storage device. It is treated in a similar way to disk files except that only one file can be saved to the module, called "MINIMEM". For minimem or disk, if you use FIXED with no number, the file is padded to 80 bytes length.

For Minimem operation, your opening line may be as short as:

```
OPEN #1:"MINIMEM"
```

The file is assumed by the computer to be

FIXED 80,SEQUENTIAL,DISPLAY (care!),UPDATE.

Any of these assumptions (or defaults) can be altered by adding the definition you want to the OPEN statement. Minimem will retain its data files provided you switch the console off before inserting and removing the module and do not use it for anything else. Minimem also permits the memory expansion to be used for data storage (see Mini Memory Manual) but this data is lost when you disconnect the power.

You may find that you need to slightly alter the volume level on your tape recorder to load data files correctly. It IS possible to load corrupted data if the volume is slightly incorrect, and this may result in your program 'crashing' with an error message such as "BAD VALUE' which is not immediately caused by an incorrect volume setting.

Some TI Modules save data to tape in 'program format', using the normal SAVE CS1 and OLD CS1 routines. The advantage of this is that the data does not occupy so much tape, takes less time to load, and it is possible to use the CHECK option to verify the data is saved correctly.

Using tape data files using OPEN & PRINT, the only way to verify your data is to read it back yourself, with an INPUT file.

At the time this text was written, it was possible for 99/4a owners to save and read their own data in this format if the Personal record Keeping or Statistics modules were in the module slot.

These two modules add several new subprograms when TI BASIC is selected, but as this is not advertised it may be amended in future. Their use is fairly technical, as memory has to be reserved by the programmer. For fuller details write to Texas Instruments or the User Group.

NOTE: EOF, "END OF FILE", is not available for tape files.

Typical routines to SAVE and LOAD high scores for a game:

TO SAVE a high score, which is in a variable called HISCORE

```
100  REM DO NOT OVERRECORD YOUR PROGRAM
110  REM ONLY RECORD DATA ON BLANK TAPE
120  OPEN #1:"CS1",INTERNAL, OUTPUT, FIXED
130  PRINT #1:HISCORE
140  CLOSE #1
```

TO LOAD this data back into the program:

```
200  REM TAPE MUST BE IN SAME  POSITION
210  REM AS AT START OF ABOVE SAVE  ROUTINE
220  OPEN #1:"CS1", INTERNAL,INPUT,  FIXED
230  INPUT #1:HISCORE
240  CLOSE #1
```

6

Advanced Programming

In the following chapter we will consider, briefly, how the computer works, with a view to making better use of its facilities.

The TI99/4A contains a 16 bit microprocessor, the 9900. This was one of the first 16 bit processors to be made. However, apart from a very small section of memory, the processor communicates with the rest of the console using an 8 bit data line. As a user you do not therefore see the high speeds theoretically possible with a 16 bit micro. Nonetheless, the 16 bits are used to make speech synthesis possible, and can also be used by an experienced programmer to speed up some chain code programs.

Most computers place user programs in RAM (random access memory) which is addressed (eg spoken to) by the CPU (central processing unit). This RAM is therefore CPU RAM: it is used by the CPU.

The 99/4A however only has a tiny amount of CPU RAM (16 bit addressed) and when only the console is used, the users program does not reside in CPU RAM.

To provide sprite action, TI have provided a second processor the VDP (visual display processor) and this has its own RAM, referred to as VDP RAM. This 16k of memory cannot be directly addressed by the CPU.

The VDP RAM is used for screen display, variables, and your program . . . . which the main processor cannot directly address. Your program is passed to the CPU two bytes at a time, which does not form a particularly fast means of communication, and may be responsible for some of the slow speed of the 4A.

Because there is no RAM addressable by the CPU, you cannot enter or run programs in machine code, unless you add CPU ram in the form of either the 32k expansion card or the 4k mini memory module.

The CALL PEEK and CALL LOAD of Extended Basic operate only on CPU ram, and if you only have the console, you will not be able to find your program in memory.

Only the mini memory module allows you to PEEK and POKE the VDP RAM, using CALL PEEKV and CALL POKEV.

If you have the mini memory or the 32k ram and extended basic, you may look at how the computer stores your programs. With Extended Basic and the 32k ram, you may look for your program from memory location -25 to -24576. The first line entered of your program will 'end' at -25, then as each new line is entered (or edited), regardless of the line number, it is placed on top. If a line is edited, the old line is removed, all subsequent lines change position, and the new line goes to the top.

With Mini Memory, provided you do not have a disk controller attached, your program starts at VDP address 16383 and works its way towards 1536 (or the bottom of the stack if earlier). With a disk controller attached, the program ends at the bottom of the stack and is pushed towards 16383, which makes it harder to look at the program (the locations keep changing).

With this information, you may not only look at your program and see how the program is stored, you may alter the program lines. Using CALL LOAD (or CALL POKEV) it is possible for one program to write over itself and create a completely new program.

SAMPLES:

Type in:

```
100 REM TEX
110 A=B+2
120 C$=D$&"E"
```

Now, if you have extended basic and the 32k ram, add:

```
200 FOR I=-25 TO -64 STEP -1
210 CALL PEEK(I,A)
220 PRINT I;A;CHR$(A)
230 NEXT I
```

For mini memory owners:

```
200 FOR I=16383 TO 16343 STEP -1
210 CALL PEEKV(I,A)
220 PRINT I;A;CHR$(A)
230 NEXT I
```

Do not edit any of these lines! If you make a mistake start again!

These are the results with Extended Basic

```
MEM:VALUE:     MEANING:
-25    0       END OF LINE
-26   88       ASCII code for X
-27   69       ASCII code for E
-28   84       ASCII code for T
-29   32       ASCII code for SPACE
-30  154       CONTROL code for REM
-31    6       LENGTH OF LINE
-32    0       END OF LINE
-33   50       ASCII for 2
-34    1       "1 digit follows"
-35  200       "Number follows"
-36  193       CONTROL CODE for +
-37   66       ASCII for B
-38  190       CONTROL CODE for =
-39   65       ASCII for B
-40    8       LENGTH OF LINE

-41    0       END OF LINE
-42   69       ASCII for E
-43    1       "1 letter follows"
-44  199       "String follows"
-45  184       CONTROL CODE FOR & (concatenation)
-46   36       ASCII for $
-47   68       ASCII for D
-48  190       CONTROL CODE for =
-49   36       ASCII for $
-50   67       ASCII for C
-51   10       LENGTH OF LINE
```

A lot can be learned of the machine's operations in this manner: it is possible to see how the program is stored, and also possible to learn the control codes for the BASIC commands.

NOTE: Although you key in REM, only one byte has been used.

If you key in GOTO only one byte is used, but GO TO(with a space in between) uses 2 bytes because two command words are used.

Note that A$ takes up two bytes but that "E" takes up three: one to indicate 'string', one to indicate how many letters, and then one for 'E' itself.

Note that '2' similarly occupies 3 bytes but that 'B' only uses one byte.

If program memory is scarce, replacing often used numbers with single letter variables can save a little memory. You will appreciate that the number '12345' will occupy 7 bytes but a variable set to this value only occupies 1 byte!

(NB: Each numeric variable used also occupies stack space, and will always use 8 bytes of stack space regardless of the number).

Although not shown here, the CALL routines occupy 1 byte for the word CALL, but for example COLOR takes up 7 bytes, as it is treated as an 'unquoted string' (command code 200). Because command code 200 is used instead of 199, you cannot CALL A$. (A$ IS A QUOTED STRING).

You may use this procedure to thoroughly investigate the way your computer stores its programs.

The memory locations -52 onwards in this example hold the LINE INDEX. As program lines do not appear in memory in line number order, but rather in the order keyed in, the computer makes use of an index, which IS in line number order.

Each line used occupies 4 bytes for the index:

```
-52  226
-53  255
-54  100
-55    0
```

Locations -54 and -55 give the line number (100) and locations -52 and -53 give the location of that line, slightly coded!

The memory location is 255x256 + 226 (which is 65506).

Although the computer can address 64k, it does so by splitting it into + and - 32k. To convert this large result to a memory location we can use, we subtract 65535:

Location = 65506 - 65535 = -31, which is where the line commences.

As each program line takes up a minimum of 7 bytes:

Index=4, Line end=1, Line length=1,
Single command=1.

It makes sense when memory is tight to use as few lines as possible: this is where you can make appropriate use of the facilities of Extended Basic.

Here is a short program to show how you can force a program to write itself.

```
EXTENDED BASIC with 32k RAM:

Key in in THIS order!:

100 GOTO 140

110 PRINT "!!!!!!!!!!!!!"

120 END

130 STOP

140 CALL INIT

150 CALL LOAD(-47,156,199,13,84,69,83,84,32,
    67,79,77,80,76,69,84,69)

160 GOTO 110

170 END
```

After you have keyed this in, LIST it, then RUN it and LIST it again. Note that CALL LOAD has been used to enter several values into memory at one time: the first value goes into -47, the second value to -46, the third to -45 and so on.

Although you can overwrite a program line in this manner, the new line must be as long (in internal storage) as the old one, unless you wish to rewrite the line index.

(Line 110 above contains 13 exclamation marks).

Using the MINI MEMORY, you have access to the VDP ram, and you can use this facility to change the definition and colour of the cursor, or even to have a sprite or two running in TI BASIC.

Cursor Definition:

The cursor definition is held in VDP RAM 1008 to 1015 (eg 8 bytes). You are used to defining characters with sixteen hexadecimal characters, but your code is translated by the console to 8 bytes:

Each row of pixels can form a binary number of 8 bits, with the left most pixel having a value of 128. Thus a solid block of pixels in one row can be thought of as the binary number 11111111, which in decimal form is: 128+64+32 +16+8+4+2+1 =255

For a block cursor, try:

```
100  CALL POKEV(1008,255,129,129,129,
129,129,129,255)
110  INPUT A$
```

The cursor will retain its new definition until the console is reset by using NEW or by loading a new program.

Cursor colour is defined in VDP RAM 783. The foreground colour and background colour are contained in a single byte. To separate them you need to divide the byte into two nybbles.

First form the two required colours into binary numbers (binary 0 has a colour value of 1):

WHITE=16 =binary 15 = 1111
TRANSPARENT=1 = binary 0 = 0000

Thus a white cursor on a transparent background would be:

11110000 in binary, which is decimal 240.

To see if this works, try: CALL POKEV(783,240).

Although TI Basic does not recognise sprites, a small part of the memory used by sprites is free, and by placing values there we can place three sprites on screen (stationary): The sprite definitions must be placed in VDP RAM 768, and are made up of 4 values. The sprite definition must terminate with 208.

The first value is the pixel row (up to 192)

The second value is the pixel column (up to 255)

The third value is the character code (NB: ASCII CODE +96)

The fourth value is the colour code (-1)

Thus

```
CALL POKEV(768,98,128,161,161,1,208)

OR FOR THREE SPRITES:
CALL POKEV(768,98,128,163,1,20,40,164,1
130,170,165,1,208)
```

The memory area dealing with sprite velocity is also clear, temporarily, but is used by BASIC for the value stack: in normal operation the computer will remove your velocity data as it moves stack data around.

It IS possible to fool the computer though: the top of the stack can be pushed down out of the way be redefining some of the lower case characters (they are usually derived rather than defined, thus saving memory).

First use:

```
100  A$="F111"
110  FOR I=96 TO 120
120  CALL CHAR(I,A$)
130  NEXT I
```

Now you can move your sprite(s):

Velocity is to be placed in VDP RAM 1920-, and each sprite requires 4 bytes. The first two bytes are for row and column velocity (max 255) and the other two are for vdp use.

Having placed velocities in the correct VDP RAM, the computer must be instructed to move them! This is done by loading the number of sprites to be moved into CPU RAM -31878.

After the above memory relocation try:

```
200  CALL CLEAR
210  CALL POKEV(768,98,128,161,1,208)
220  CALL POKEV(1920,50,50)
230  CALL LOAD(-31878,1)
240  GOTOO 240
```

Being able to use one or two sprites can permit some advanced graphic work in your TI Basic programs. Each sprite can be positioned to within one pixel on the screen, and can be moved one pixel at a time. (A standard character is 8x8 pixels).

Graphics Modes

The VDP chip permits the use of 4 graphics modes, but only three are available with BASIC programs, and only one if you only have the console.

The standard mode is 32x24 characters. This is all that is available to you if you only have the console.

Some utility programs are available giving pseudo hi resolution graphics, which work by redefining characters, but they tend to be a little slow.

TEXT mode allows 40 x 24 characters. It requires a machine code program to allow you to use it (various utilities are commercially available).

MULTICOLOUR MODE divides each character into four blocks, and each block can be any colour.

To see multicolour mode, try the following:

(Requires Mini Memory or Extended Basic + 32k ram):

```
100 CALL INIT
110 CALL LOAD(-31788,204)
120 CALL KEY(0,A,B)
130 IF B<1 THEN 120
140 CALL HCHAR(1,1,45,200)
150 FOR Z=1 TO 57
160 FOR X=1 TO 14
170 PRINT CHR$(Z+30);
180 NEXT X
190 NEXT Z
200 CALL LOAD(-31788,224)
210 CALL KEY(0,A,B)
220 IF B<1 THEN 210
230 END
```

Enter RUN, then press any key to start the action. At program end, press another key to return to normal. If necessary switch off to return to normal!

HI RESOLUTION MODE allows pixel plotting, but with nearly 49000 pixel positions (plus colour information) there is no room to operate both this mode and the BASIC operating system. It is only possible in machine code programs — such as PARSEC.

IF . . . THEN . . . ELSE

TI BASIC may appear to be slightly limited in its use of IF . . . THEN compared to some other computers. TI do however allow the ELSE alternative.

The problem arises because TI insist that you use the construction only to transfer to another line. You cannot add commands such as:

```
IF X=B THEN B=C
```

to do this you need Extended Basic.

However, TI BASIC does have 'relational operators' which will often help you out of this problem.

The instruction IF X=1 THEN 100 is acted upon by the computer only if the expression (X=1) is TRUE.

A TRUE expression is treated by the computer as having a value of -1, while a FALSE expression is treated as having a value of 0.

The IF . . THEN structure does not require an expression to evaluate to 0 or -1 however, and you may use a variable on its own to perform a line transfer :

IF X<>0 THEN 100 will transfer to line 100 if the variable X has any value.

IF X THEN 100 will have exactly the same effect but use less memory.

It is possible in TI BASIC to build up a set of expressions in an IF . . . THEN line, which will simulate OR and AND, and, if you are careful, you may go well beyond OR and AND.

Each expression to be evaluated MUST appear in brackets.

For example:

IF (A=1)+(B=10) THEN 100

If both A= and B=10, then the sum of the two expressions is -2 (-1 plus -1), and as the result is not zero, the transfer to line 100 will take place.

If only A=1, but B=5, then the sum will be (-1+0) or -1, and the transfer will still take place.

If A=3 and B=5, then the sum is (0+0) or 0, and the transfer to line 100 will not occur.

What we have then is a way of saying

IF A =1 OR B=10 THEN 100 (don't type this line in).

Using a different mathematical operation, the multiply or *

IF (A=3)*(B=2) THEN 100

When A=3 and B=2, the calculation is (-1 * -1) or +1.

The result is none zero and the line transfer takes place.

When A=3 and B=1, the calculation is (-1 * 0) or 0

The result is zero so no transfer will occur.

What we now have is a way of saying, in TI Basic:

IF A=3 AND B=2 THEN 100.

Provided you are careful to always know the possible results of the various expressions and mathematical operations, you may build up some very powerful IF . . . THEN commands, which will save you many lines of tedious programming.

This however is not all you can do with relational expressions. How about trying to program IF A=5 THEN B=6 ELSE B=0 in TI Basic?

B=-6*(A=5) has exactly this result. If (A=5) then the calculation is B=-6*-1, or B=6. If A does not equal five, the calculation is B=-6*0, or B=0.

This is fairly advanced programming, but if you need this sort of power, it is there for you to use. All you need is to keep track of the possible values of the variables you use.

To whet your appetite: IF (X=1)+(Y=1)+(Z=1)=-2
THEN 100

This interesting group of expressions will transfer to line 100 if any two of the three bracketed expressions is true. What this fairly short line says to the computer is:

If any two of X Y & Z are equal to one then . . .

Instead of the = sign, you can use any of the relational

operators, = < and >. The same structures can be used with string variables.

The JOYSTICK program following this section makes use of thee relational expressions.

DEF

Here are some more advanced uses of DEF

DEF ACS(C)=1.5708-2*ATN(C/(1+SQR(1-C*C)))

All that does is supply you with an ARCCOS function. The trigonometrical functions which TI do not provide can all be made up in a similar manner.

Now in your program when you want to compute the ARCCOS (the angle *IN RADIANS* of a right triangle formed by the hypotenuse H and one of the sides X. Angle=ACS(X/H)) you use this defined function. eg ANGLE=ACS(SIDE/HYP)

If you don't like radians, you can amend the defining line, or add a second, after this one:

DEF DEG=RAD*57.29578

Then when you want to convert a radian result, set the variable RAD to the radian angle and whenever you use DEG the computer will use the degree measure equivalent to the radian angle which RAD is set to.

This represents another use of DEF: No 'argument' is passed. The defined variable DEG will, whenever used, take a value which depends on the *current* value of the variable RAD.

To avoid using INT frequently for the same variable, you may use the DEF function:

```
DEF  A=INT(A)
```

Now whenever the variable A is used, any fraction will be dropped each time.

The DEF function does not allow you to have more than one argument to be passed, and must occupy just one program line. It is possible however to use one defined function in another, so long as they occur in sequence in your program.

eg

```
DEF  RAN(X =INT(RND*X+1)

DEF  SCORE=RAN(LEVEL)+SCORE
```

A routine to use the joystick, which we will develop into a rather complex multi-purpose input routine:

The CALL JOYST format is not ideally written for the TI99/4A. The row and column variables are reversed compared to the graphics commands, and the subprogram seems to assume the screen origin is at bottom left (it is actually at top left).

The return variables are placed in the command as follows:

```
CALL  JOYST  (NUMBER,  COLRETURN,
ROWRETURN)
```

whereas the graphics commands are in the form:

```
CALL  HCHAR(ROW,COL,CODE)
```

To move a character to the screen left, we need to decrease the value of the column. If the joystick is moved to the left, the column return variable is indeed negative, while movement to the right gives a positive return.

However, the top of the screen is Row 1, so to move down the screen the row must be increased : move the joystick down and the row return variable is NEGATIVE. This can cause confusion very easily! The sign has to be changed to make this work!

Remember: the alphalock MUST be up for the joystick to work.

If we wish to amend ROW and COL variables using the joystick, it is necessary to use:

```
100  CALL JOYST(1,COLRET,ROWRET)
110  ROW=ROW-ROWRET/4
120  COL=COL+COLRET/4
```

Notice the different signs used to amend the row and column variables. We have to divide by four because the CALL JOYST will only return 4, 0, or -4.

To use the above in a program leaves one problem: you need to know which is joystick number one! This can be marked, but you can also scan both joysticks:

```
100  CALL  JOYST(1,CR,RR)
110  CALL  JOYST(2,CR2,RR2)
120  ROW=ROW-RR/4-RR2/4
130  COL=COL+CR/4+CR2/4
```

This will take a little longer to process and you will have to see the effect in your program before you decide to use it.

When using CALL KEY there is a status return we can check to see if NO key has been pressed. With joysticks there is no status return, only the return variables. A status return can however be created.

Instead of the CALL KEY "IF ST=0 THEN" it is possible to use "IF CR+2*RR=0 THEN"

Why multiply the second return by 2? Check through all the possible returns from the joystick and you will see that a simple addition, subtraction or multiplication will not return a unique answer to equate with "joystick central".

So far we are amending the variables ROW and COL without checking to see if they are valid. To use HCHAR etc they must be from 1 to 24 or 32 respectively. Anything else will produce an error message and halt the program.

It is possible to use lots of lines of coding in TI Basic:

```
200  IF ROW<1 THEN 210 ELSE 220
210  ROW=1
220  IF ROW>24 THEN 230 ELSE 240
230  ROW=24.....

     and so on.
```

It is easier however to add to the ROW incremental line a value check which will reverse the increment if it places the value outside the limits.

To do this we need to use the relational expressions discussed under "IF THEN" in the previous section.

If a relational expression is TRUE it has a value of -1

If a relational expression is FALSE it has a value of 0

Thus PRINT (2=3) will appear as Ø, but
PRINT (2=2) will appear as -1.

Dealing with the ROW first, if the variable ROW starts with a value of 1, and the joystick is pushed up, we must reverse the reduction of ROW.

There are two expressions which must be true: if both ROW=1 and RR=4 then after we have added 1 to ROW we must deduct it, to leave it set to 1:

```
ROW=ROW-RR/4+(ROW=1)*(RR=4)
```

Now it is impossible for ROW to become less than 1.

This has been developed further in the sample program which you will find printed separately.

A typical use of the joystick is to move a character around the screen and this is what the sample program will do. To give greater flexibility, this program checks both joysticks, and also checks the keyboard (keys WERSDZXC).

First the screen is cleared and the row and column variables are set to initial values. Our character is placed on screen and the joy sticks and keyboard are scanned.

The next line checks to see if an input has been made: if neither joystick nor the keyboard has been used, the program will go back and look at the joysticks/keyboard again. The plus sign between the relational expressions serves as an 'OR'.

If the status of one keyboard unit is NOT zero, the program continues.

Now the program is divided into two. If the keyboard has

been used, the variable ST will have a non-zero value which causes the program to branch to the keyboard section. Otherwise it continues with the joystick section.

The joystick section is a slight development of what has been discussed above. We are checking for both limits to the row variable.

The keyboard section uses similar principles, but the limit checks are a little different: If the ROW variable has a value of 1, it cannot be decreased as (RW<>1) takes a value 0 (false) and no change is made.

In the sample program a character is moved around the screen, but if you wish to leave a line of characters, just delete the line which places a blank (32) in the old position.

That was quite a complex program to develop, so check it over thoroughly. The use of relational expressions can become quite complex, but they can both speed up execution time and save memory usage.

ACCEPT AT

The next program is an ACCEPT AT routine in TI BASIC. A PRINT AT routine can be found in VINCE APPS book of TI99/4A programs.

The INPUT command causes the screen to scroll, and in the middle of a game with a complex screen display that can be disruptive!

The required routine will allow you to fill a variable string and place the input onto any desired part of the screen. The initial screen location is held in the variables R(row) and VR(column).

The required input is to be placed in a string variable IN$. To ensure the variable is 'empty' it is cleared at the start of the routine.

The string is filled by means of a series of CALL KEYs, terminated with the ENTER key (which gives a code of 13).

For user confidence the cursor (character 30) is flashed at the position the input will appear at. This comes immediately after the call key. If required a CALL SOUND could be inserted just before the CALL KEY to provide the usual 'input' tone.

If no key is pressed, the cursor will just flash.

When a key is pressed, a check is made to see if it is the ENTER key, to terminate input (Key code 13). If ENTER has not been pressed, you can check to ensure the key falls within a required range.

In the sample given, only the number keys 0 to 9 are accepted as inputs (ASCI codes 48 to 57). If the choice of keys is not so neatly in sequence, you may use:

```
IF POS("ESDX",CHR$(S),1)<1 THEN 110
```

If the key pressed is not E, S, D or X in this sample, the key is not accepted and the program returns to the CALL KEY.

If the key IS accepted, the letter is placed on the screen in the appropriate location, and the value of the column is increased by one.

A check is then made to see if the input has gone past the end of the screen : if no check was made, a BAD VALUE would be possible.

In this example, if the column exceeds a value of 33, it is reset to 32 and an automatic ENTER is inserted to terminte the input. You may prefer to substitute GOTO 110 instead of ending the input : this alternative places the cursor back on the last position (eg at screen right).

If the key is accepted, the character the key represents is added to the input string and the program returns tothe CALL KEY.

Once ENTER has been pressed, a check is made to see if aninput HAS been made (a nul input string could cause your program to crash). If no entry has been made the program returns to the CALL KEY.

If all is well, you are allowed to RETURN to the place in your program you left with a GOSUB. The input now lies in variable string IN$ for you to manipulate as you wish. As the example program has limited the input to digits, a numeric variable can be set by using N=VAL(IN$) in your program.

Additional programs may be found elsewhere in this book. Try to see how they work — can you improve them!

A quick routine for right justification (useful when using columns of numbers) — the variable to be printed is in the string variable A$ (use STR$(4) etc to convert numbers to strings) and the right column in variable RC:

```
X=RC-LEN(A$)
FOR C=1 TO X
A$=" "&A$ (this adds one space in front of
A$)
NEXT C
PRINT A$
```

SIMPLE TIPS

A=B^2 takes longer to process than A=B*B.

A=20000 uses more program memory than A=2E4

Data Compression

When memory is limited, any technique which allows you to fit a quart into a pint pot is useful.

A number of data compression techniques have been evolved, some of them quite complex, but we shall deal with the subject only briefly here.

First remember that the TI99/4A stores:

NUMBERS in the number of digits in the number plus 2 bytes.

STRINGS in the number of characters in the string plus two.

NUMERIC VARIABLES in the number of characters in the variable name, plus 8 bytes of the stack.

STRING VARIABLES in the number of characters in the variable name including the dollar sign, plus stack memory the size of the string plus one.

To store a screen row, column and character value in three numbers takes 12 bytes, in three numeric variables of one letter each, 3 bytes plus stack space of 24 bytes.

Using numeric variables, the stack space is only occupied once for each variable, so if the numbers are used frequently, use of variables may save memory.

Where a large number of such data is to be stored, memory may be saved by compacting each set into a single number:

say Row 12, Column 23, Character 32
can become a single number 122332.

To break the number up if it is in variable A, we can use:

```
ROW=VAL(SEG$(STR$(A),1,2))
COL=VAL(SEG$(STR$(A),3,2))
CH=VAL(SEG$(STR$(A),5,2))
```

Instead of using a number, it is also possible to place the data straight into a string, say A$, where A$=STR$(A).

Using a string has an advantage in that it may be up to 255 bytes long, and one string can thus contain a great deal of information.

In an adventure program for instance, 255 bytes can easily contain screen information, pointers to standard text, strength and agility counters and so on.

The only drawback is that the greater the degree of compaction, the slower your program will run.

However if your program is impossible without compaction, it is well worth looking at ways in which data can be placed into the computer using as little memory as possible.

An understanding of how the computer uses its memory is vital for this to be effective.

Quick sort

Sorting data is a frequent task for many programs, and it seems reasonable to use the fastest sorting method available.

This sorting routine is very fast.

The variables used are:

A,B,C,D,E, F()
A$(), B$

To use the sort, your program must contain the initialisation lines at the beginning. You should not use the above variables in the main program, but if necessary you can change the variable names in this routine to avoid conflict.

The initialisation is here for 200 items. If you wish to sort a different number of items, set C to the number of items to be sorted (line 130) and DIMension A$ in line 100 to the number of items plus one. Then in line 150 set A$ (Number of items plus one) to : (FCTN and A).

Your program may enter the routine with GOTO (EXIT with GOTO) or with GOSUB (EXIT with RETURN).

The items to be sorted are to be placed in the array A$(), and when the routine is finished, the items will still be in the array, but in ascending order, depending on the ASCII codes of their letters:

eg AA after A, B after AZZZ and so on. A$(0) is NOT used for the items to be sorted, it is a flag.

This routine will sort up to 1000 items.

After that, you will need to DIMension the S array — to S(11) for 2000 items, S(12) for 4000 items and so on.

```
Initialisation:
 100 DIM A$(201)
 110 A=1
 120 B=1
 130 C=200
 140 A$(0)=" "
 150 A$(201)="¦"

    { program }


2000 IF C-B<10 THEN 2320
2010 D=B
2020 E=C
2030 B$=A$(B)
2040 IF B$>=A$(E)THEN 2070
2050 E=E-1
2060 GOTO 2040
2070 IF E>D THEN 2100
2080 A$(D)=B$
2090 GOTO 2190
2100 A$(D)=A$(E)
2110 D=D+1
2120 IF A$(D)<B$ THEN 2110
2130 IF E>D THEN 2170
2140 A$(E)=B$
2150 D=E
2160 GOTO 2190
2170 A$(E)=A$(D)
2180 GOTO 2050
2190 IF C-D<D-B THEN 2260
2200 F(A)=C
2210 A=A+1
2220 F(A)=D+1
2230 A=A+1
2240 C=D-1
2250 GOTO 2000
```

```
2260 F(A)=D-1
2270 A=A+1
2280 F(A)=B
2290 A=A+1
2300 B=D+1
2310 GOTO 2000
2320 E=B
2330 E=E+1
2340 IF E>C THEN 2430
2350 B$=A$(E)
2360 D=E-1
2370 IF A$(D)<=B$ THEN 2410
2380 A$(D+1)=A$(D)
2390 D=D-1
2400 GOTO 2370
2410 A$(D+1)=B$
2420 GOTO 2330
2430 IF A=1 THEN 2490
2440 A=A-1
2450 B=F(A)
2460 A=A-1
2470 C=F(A)
2480 GOTO 2000
2490 RETURN
```

7

Extended Basic

The Extended Basic module is not inexpensive, and information on what it can do is not widely available. Therefore this section has been included to help you to decide if you need the module, and to give some short hints on its use.

Extended Basic exists in two distinct versions, Vn 100 and Vn 110. Only a very small handful of the earlier VN 100 have been sold in the UK, and this book refers to VN 110. A principal difference is speed: 110 is much faster. There have also been changes in the operating systems which sometimes cause incompatability between the two versions (with special reference to the sprite routines and user sub programs.

EXTENDED BASIC is a VERY much larger language than TI Basic. The increase in operation speed is not shown by magazine 'bench tests' which use very short specific programs. In a typical program you will find the program runs in about 30% less time. Line transfers and screen handling are particularly faster than in TI Basic.

In itself, this is a great attraction, but Extended Basic also adds very many new commands and functions, enabling better use to be made of limited memory, and also permitting friendlier programs to be written.

Many TI BASIC programs can be loaded in Extended Basic, and will then run faster.

Exceptions are:

TI Basic programs over 12k cannot be loaded due to lack of memory.

Some TI Basic programs will load but cannot RUN due to lack of memory.

TI Basic has two extra character sets: if these are used, they will produce a BAD VALUE error in Extended Basic. Extended Basic uses the memory saved by dropping these sets (15 & 16) to produce the Sprites.

What is different about Extended Basic? The following is only a short list:

ACCEPT AT and **DISPLAY AT** permit you to 'display' text anywhere on the screen, or to 'accept' text anywhere. There are numerous variations to these commands:

Optional 'beep', the ability to input data already on the screen or type over it, validation of input, and, for the display of data, the ability to 'image' numbers. This allows simple justification of numbers.

Typical use: ACCEPT AT(6,12)BEEP VALIDATE("YN") SIZE(-1):A$

If there is a "Y" at row 6, column 12, just pressing ENTER will place Y into A$. Press any key but Y or N and the input will not be accepted, the computer returns to the line itself, and waits for a valid input.

CALL: In addition to the subprograms provided (eg COLOR, SOUND and so on) you may write your own sub

programs, which you activate with CALL — for example CALL MYSUBPROG.

As with the TI subprograms, you may pass values or variables to your subprogram, and variables used in the subprogram are separate to variables in your main program. This is potentially a powerful programming capability.

Unfortunately the usual system error traps work badly with user written sub programs, and if an error message is generated it will usually be the wrong message for the error the computer has discovered. You must be careful how you write your subprogram!

A number of the built in CALLs have been extended, to allow you to define 4 characters at once, or amend all the colors at once — eg CALL COLOR(1,2,2,2,3,3,8,15,16) and so on.

This not only saves memory but also processes more quickly.

SPRITES you have probably heard of: they are smoothly moving graphics characters, which move under the control of the Video Display Processor, while your program carries on with other things.

TI allows you 28 sprites, each of one to four characters. They may also be double size (eg a 4 character sprite occupying the screen area of 16 characters).

There are subprograms built in to determine if sprites are in a particular position or overlapping, and you may quickly reposition them, change velocity, change colour or change the character of a sprite.

There are two restrictions which mean you need programming skill to use them to full effect:

The processor can only handle 4 sprites at a time in line. Each row of pixels is restricted to 4 sprites — any extra are made invisible. (A PIXEL is one dot in a character grid. A TI99/4A character is made up of dots in an 8 x 8 grid).

The **CALL COINC** coincidence checker only checks at the instant that command is used: you need to use it fairly often if a coincidence is not to be missed.

Nevertheless, in the hands of skilled (and patient) programmers, sprites can produce some VERY clever programs (in a form of the BASIC language too).

IF . . . THEN . . . ELSE has been greatly improved. You are no longer limited to line transfers, but may use:

IF A=4 AND B=6 THEN R=10 ELSE PRINT "OOPS"

Using IF . . THEN . . ELSE with commands enables you to use the memory available in a much better way.

LET

You do not actually use LET with the 99/4A, but is has to be listed somewhere . . .

In Extended Basic, instead of using: A=0 B=0 and so on, you may assign one value to several variables in a neat and memory saving manner:

A,B,C,D,E,F,G=0 will reset all those variables to zero!

LINPUT: Ordinary INPUT removes leading spaces and causes problems if you wish to input a string with a comma in it.
LINPUT avoids these problems. It stands for LINE INPUT.

LIST . . . when you LIST you may make the computer PAUSE in the list by pressing any key, then press another key to make the LIST continue.

ON BREAK NEXT disables the CLEAR key except when the computer has halted for an INPUT or ACCEPT AT. By avoiding these input commands you can make your program unbreakable.

ON ERROR is a VERY useful command. Normally an error causes your program to break, but you may use this command to transfer program execution to an error routine of your own. Your error routine may end by instructing the computer to try the problem line again, to go on to the next line, or to go to any other line in the program. You may also print your own error messages, such as 'PRINTER NOT CONNECTED". This command can be used to make your programs totally user friendly. NB: Do not insert until your program is completely debugged!

ON WARNING is similar but with fewer options: you may halt the program or continue.

PROGRAM LINES: May now contain more than one command, and can be entered up to 5 screen lines long (but limited to 128 bytes long internally).

You may use IN COMMAND MODE for instance:

FOR A=110 to 220 : : CALL SOUND (200,A,0) : : NEXT A

The double colon is a statement separator. In TI Basic you could enter PRINT A::B::C.

In Extended Basic you must leave a space between the colons:

PRINT A: :B: :C

(A program in TI Basic is converted automatically by the machine to the new format, but you must take care when typing in a program. Due to an omission in the error handling system, typing too many colons together in Extended Basic can cause the processor to 'lock out')

When this is linked to the new capabilities of the IF . . . THEN command, it is possible to put together some very powerful program lines:

IF A=B THEN C=5 : : PRINT A : : ELSE IF A=8 AND B=C THEN GOTO 3400 ELSE CALL SOUND (100,110,0) : : GOTO 200

As the lines become longer and more complex, you do need to take greater care, but the language gives you a very powerful tool.

In addition to using REM after double colons, you may use a 'tail remark', which is a '!' as follows:

SCORE=0 ! RESET SCORE

RUN It is possible to RUN one program from another:

RUN "CSI" or RUN "DSK1.PROGTWO"

SAVE Programs may be saved in PROTECTED format, which prevents listing, editing or saving, and may be saved to DISK ONLY in Merge format, which allows program segments to be spliced together.

SPEECH From the speech editor module comes CALL SAY and CALL SPGET, which enable your Extended Basic program to use the speech synthesiser. Although TI provide a vocabulary list with the Extended Basic module, full instructions are not provided.

CALL SAY allows you to SAY a word from the vocabulary. Words NOT in the list will be spelt. Some 'words' are really phrases, but if you use CALL SAY("READY TO START"), the computer will SPELL the words! This is because the space is treated as a word separator. For the computer to recognise that these three words are one unit in the vocabulary, you need to enclose them in hash marks: CALL SAY("#READY TO START#")

The standard punctuation marks are also word separators and provide differing degrees of pause between words.

In order of length of pause, the separators are:

+ (space) - , ; : .

The + is zero pause and the full stop is a one second pause. The separators may be repeated to build up any pause, eg:

CALL SAY("I---KNOW") or CALL SAY("I-,,KNOW")

CALL SPGET is used to fill a string variable with the data use by the speech synthesiser. This can increase the speed of execution if you fill some string variables at the start of your program, and then use them when you wish to speak. It takes a little while to fill the variables, as each string is 255 characters long (many of them are nul or zero value).

eg	CALL	SPGET("WORKING",WK$)
	then	
	CALL	SAY(,WK$) will say the word.

Note the comma in front of WK$. If two strings are used,

in addition to the leading comma, they must be separated by TWO commas: CALL SAY (,A$,,B$)

In addition, you may use SEG$ to curtail the string you have returned. You can then separate the initial sounds of each word, and use these to create your own vocabulary. You are NOT limited to the preprogrammed list: you just have to work a little to expand it.

For example, having loaded WK$ as above, try:

```
WK$=SEG$(WK$,1,60)
CALL SAY(,WK$)
```

Notice any change? Try using different lengths in the SEG$ command. This is an area for experimentation.

SIZE returns the amount of free memory.

EXTENDED BASIC uses some of the system RAM, and you do not have quite as much memory available for your programs. In addition, the cassette loader cannot handle programs over 12k.

The good news is that with Extended Basic you may access the memory expansion unit, which permits you to load (from DISK) a program up to 24k, and still have some 14k available for variables and so on.

The new function key **REDO** will repeat your last entry, and if the last entry was a program line (either just entered, or recalled using FCTN X) the line reappears on the screen with the cursor at the beginning of the line NUMBER, allowing you to change the line number if you wish. This function is useful if your program contains a lot of lines either the same or with only small differences.

Lockouts have been found to occur in the present

Extended Basic by using CALL PEEK at one particular section of memory (the addresses vary from console to console), and by using a number of print separators without spaces:

> PRINT ::::::: (Correct in TI BASIC, but Extended Basic requires a space between each colon).

An added attraction of the module is that it permits you to load and run Assembly language programs, provided you have the extra peripherals required.

In summary, EXTENDED BASIC requires a little more care in use but gives you considerably more programming power.

The following program has been included to show how SPRITES are used in EXTENDED BASIC. The program was developed in a highly experimental manner, as various routines and values were tried. To obtain the best from SPRITES it is usually necessary to work in this manner.

```
100 REM SPEEDRACE
110 REM A SAMPLE PROGRAM IN
120 REM TI EXTENDED BASIC
130 REM USING SPRITES
140 REM
150 REM ====================
160 REM
170 CALL CLEAR
180 PRINT "SPEEDRACE":"COPYRIGHT 1981":"BY STEPHEN SHAW"
190 PRINT "USE S & D TO MOVE ":"LEFT & RIGHT":" ":"USE KEYS 1,
    2,3,&4 TO":"SELECT GEAR"
200 PRINT "DISTANCE & TIME ARE ":"DISPLAYED.":"DISTANCE
    SUFFERS IF YOU":"CRASH"
210 PRINT "PRESS ANY KEY TO CONTINUE"
220 CALL KEY(3,V,M)
230 IF M<1 THEN 220
240 CALL SCREEN(2)
250 FOR X=1 TO 100 :: NEXT X
260 CALL CLEAR
```

```
270 CALL MAGNIFY(3)
280 M=1
290 X$=RPT$("0",40)
300 CALL CHAR(100,"96FEBA3838BAFEBA"&X$)
310 CALL CHAR(108,"5A5A5A5A5A5A5A5A5A5A"&X$)
320 CALL CHAR(104,"FF1111FF0000FF11FF"&X$)
330 CALL SCREEN(4)
340 CALL SPRITE(#6,108,13,80,9,90,0)
350 CALL SPRITE(#7,104,13,75,25,90,0)
360 CALL SPRITE(#8,104,13,70,38,90,0)
370 CALL SPRITE(#9,108,13,65,9,90,0)
380 CALL SPRITE(#10,104,13,60,25,90,0)
390 CALL SPRITE(#11,104,13,55,38,90,0)
400 CALL SPRITE(#12,104,13,50,9,90,0)
410 CALL SPRITE(#13,104,13,45,25,90,0)
420 CALL SPRITE(#14,104,13,40,38,90,0)
430 CALL SPRITE(#15,104,13,85,139,90,0)
440 CALL SPRITE(#16,104,13,80,150,90,0)
450 CALL SPRITE(#17,108,13,75,170,90,0)
460 CALL SPRITE(#18,104,13,70,139,90,0)
470 CALL SPRITE(#19,104,13,65,150,90,0)
480 CALL SPRITE(#20,104,13,60,170,90,0)
490 CALL SPRITE(#21,104,13,55,139,90,0)
500 CALL SPRITE(#22,104,13,50,150,90,0)
510 CALL COLOR(8,3,4)
520 CALL SPRITE(#23,108,13,45,170,90,0)
530 CALL VCHAR(1,8,140,216)
540 CALL COLOR(14,12,12)
550 CALL VCHAR(1,7,95,24):: CALL VCHAR(1,17,95,24):: CALL
    CHAR(95,"5555555555555555")
560 FOR CT=1 TO 4
570 CALL SPRITE(#CT,100,CT+6,CT*47-45,93-CT*8,0,0)
580 NEXT CT
590 CALL SPRITE(#5,100,16,160,74,0,0)
600 REM **
610 REM   ***
620 CALL SOUND(-1000,-2,30-7*SPEED)
630 CALL COINC(ALL,D):: IF D<0 THEN GOSUB 780
640 CALL KEY(0,A,B):: IF A=ASC("S")THEN CALL MOTION(#5,0,-10)
650 IF A=ASC("D")THEN CALL MOTION(#5,0,10)
660 IF A<30 THEN CALL MOTION(#5,0,0)
670 CALL COINC(ALL,D):: IF D<0 THEN GOSUB 780
680 IF A>48 AND A<53 THEN SPEED=(A-48)/3
690 CALL COINC(ALL,D):: IF D<0 THEN GOTO 760
700 T=T+1 :: S=S+6*SPEED :: DISPLAY AT(10,18)SIZE(10):STR$
    (S)&" "&STR$(T)
710 CALL COINC(ALL,D):: IF D<0 THEN GOTO 760
720 IF T/5=INT(T/5)THEN M=-M
730 CALL MOTION(#1,SPEED*40,M*5,#2,SPEED*40,M*5,#3,
    SPEED*40,M*5,#4, SPEED*40,M*5)
740 CALL COINC(ALL,D):: IF D<0 THEN GOSUB 780
750 GOTO 620
```

```
760 GOSUB 780
770 GOTO 620
780 CALL SOUND(-900,-6,0)
790 CALL MOTION(#1,0,0,#2,0,0,#3,0,0,#4,0,0,#5,0,0)
800 SPEED=1/3
810 S=S-50
820 IF S<0 THEN S=0
830 CRASH=CRASH+1
840 IF CRASH=15 OR T>200 THEN GOTO 920
850 M=+1
860 T=T-(5*(T/5-INT(T/5)))
870 FOR CT=1 TO 4
880 CALL SPRITE(#CT,100,CT+6,CT*47-45,93-CT*8,0,0)
890 NEXT CT
900 SPEED=0
910 RETURN
920 CALL CLEAR
930 PRINT "YOU HAVE TRAVELLED   ";"A DISTANCE OF ";S
940 PRINT "AND HAD ";CRASH;" CRASHES!"
950 IF S>500 THEN PRINT "YOU ARE NOT A BAD DRIVER"
960 IF S<100 THEN PRINT "YOU SHOULD NOT BE ON THE";"ROAD"
970 PRINT "TO TRY AGAIN,ENTER 'RUN'"
980 END
```

8

Modules

TI produce a number of powerful and useful modules in addition to the range of games you may be familiar with. This section is intended to help you obtain good value from the modules.

Some of the games modules contain a TEST MODE, which was inserted by the programmer to permit program debugging. The programs concerned are in machine code.

Titles spotted so far are: TI Invaders, Munch Man, Alpiner and Chisholm Trail.

This is not an advertised function and may be removed. But it is worth trying with any game module you have.

Insert the module and select the game.

When the first game title page appears, quickly hold down SHIFT and press 8, 3, and 8.

A new screen should appear with various prompts. Enter responses as quickly as you can. The prompts allow you to enter the game at any level, and in the case of TI Invaders for instance, to select a slow speed.

PERSONAL RECORD KEEPING

The Personal Record Keeping module enables you to store up to about 10k of data, and provides various handling and output facilities to help you manipulate your records.

A printer is useful but not essential. If you have a printer, greater flexibility of display, as well as additional functions, are provided with the Personal Report Generator module, which requires data prepared with the Personal Record Keeping module.

The PRK sorting routines are slow. Data is saved in memory image ('program') format, and thus uses less tape (or disk) space, is faster to save and load, and the verify option is available for tape files.

You will not be able to catalogue a collection of six thousand records, but small collections can be catalogued with the module. The number of items depends on how many characters you wish to use to describe each item.

It is possible to use the PRK module as a simple diary system, or a very simple spreadsheet, as it is possible to perform mathematical operations on the data you place in your module.

The Personal Record Keeping and the Statistics modules both extend the range of commands available in TI BASIC. Again this is not advertised, and may be amended.

With either module inserted, select TI BASIC. You may now use the following commands:

DISPLAY AT

CALL D(R,C,L,V)
Where R and C are the row and column the word is to start at.

L is the length of screen to be blanked from position R, C and also sets the maximum length of the display.

V is a value or string or variable to be displayed.

If V is longer than L, the display will be curtailed. R,C and L may be numbers or numeric variables.

Try:

CALL D(10,4,5,1/3)

ACCEPT AT

CALL A(R,C,L,F,A,MN,MX)
R,C and L are as with CALL D, but in CALL A, L sets the maximum length of the input.

F **MUST** be a NUMERIC VARIABLE. It takes a value of 1 if ENTER is pressed, and other values if some control keys are used eg BEGIN:6 REDO:4 AID:3 BACK:7 CLEAR:2

A is the numeric or string VARIABLE to be filled with the input.

MN and MX are optional when using a numeric variable, and set the minimum and maximum acceptable values: any input outside these values is rejected.

NB: The CLEAR key is used to clear the input field. It WILL NOT break into the program! Use CALL A with a little care if you think you may need to BREAK the program!

Other commands are also added to TI Basic with these two modules, eg CALL P (partitions memory), CALL L and CALL S which save and load data in program format to the partitioned area, and CALL G which handles the data in the partitioned section and CALL H which defines the format of the data.

A booklet on these commands has been published by TI, and you may be able to purchase a copy from the main UK User group. Sample programs may be found in 'TIDINGS' Vol 2, No 4, from the main user group.

These extra commands are the only way in which a user can save data in PROGRAM format — used by most TI Modules. Program format permits tape verification, and uses a lot less space on your tape or disk.

TI-WRITER

TI-WRITER is TI's word processing module. It comes in the form of a module, a disk, and a large manual. The 32k memory expansion is required in addition to the disk drive and a disk controller, plus the Expansion Box, RS232 card and a printer.

TI-Writer is a very powerful word processor and can carry out most of the tasks a purpose built word processor is capable of.

A word processor is a great deal more than an electric typewriter. Numerous editing facilities are provided to enable the text to be manipulated.

Typical facilities (found on TI-Writer) are:

Full screen editing —

The cursor can be moved in a number of ways, using

preset tabs, word tab, block movement (24 lines at a time) and the usual cursor control keys.

Text can be deleted or inserted.

Movement of paragraphs to different places in the text.

Ability to merge files, and save parts of files.

Ability to change one word in the text to another, wherever it may occur (for instance, replacing 99/4a with 99/4A).

Mailing list option: now you can write personalised circular letters (the sort which begins: Dear Mr Smith, You have been selected from the people in Acacia Avenue.....)

The module is compatible with programs which have been LISTed to disk (eg LIST "DSK1.PROGNAME") which allows programs to be inserted into text, and also manipulated with the various editing options allowed.

TI Writer can also be used to create or edit any disk file using 'DISPLAY VARIABLE 80' format, such as the Editor/ Assembler uses for machine code. You may use the TI Writer to edit data created and used in your own programs.

The screen is changed to 40 columns, but the TI Writer page is 80 columns long. These 80 columns are displayed in three windows, covering columns 1 to 40, 20 to 60 and 40 to 80.

Although the page is 80 columns wide, by using the commands provided with the TEXT FORMATTER (one of the TI Writer programs), it is possible to print up to the maximum length your printer will allow.

The module allows the use of any printer connected to the RS232 Card, using either the serial or parallel interfaces (see next chapter).

The command codes used by your printer can be inserted into your text, to allow you to switch say from normal print to italic print.

Centering of text is possible using TI Writer, as is 'right justification', where all lines finish in the same column at the right (the normal style of a book). TI-Writer does this by inserting extra spaces between words. The result is quite effective.

If you have the peripherals and you write fairly often, a word processor may be of use to you. This module is effective, and considering the large number of different commands a word processor must be able to handle, it is fairly easy to use.

MINI MEMORY MODULE

The mini memory module carries out a number of functions, but only one at a time:

You may use it for ONE of:

FILE HANDLING:
The module itself can be used in a TI Basic program as though it was a single disk file called "MINIMEM", and all the file handling commands available with disk drives will work with the module. It has a battery backup, and the information you store in the module will therefore remain after you switch your console off.

The module permits you to use the 32k Expansion Memory as a second 'solid state disk drive' called "EXPMEM2", which may store up to 24k of data. This data is lost when the 32k expansion is switched off.

Using either the module or the 32k expansion as data files, the information is retrieved even more quickly than with a disk drive. The computer does not have to waste time in moving a disk drive head over the disk.

It is possible to store data in the module or expansion memory with one program, and then to access the data with a second program, provided you do not reset the system by using QUIT or removing the module or power supply. This may help you to run a long adventure program for instance, by first placing the text into the memory and then loading your control program.

PROGRAM STORAGE:
A small program (up to 4k) may be stored in the module using SAVE MINIMEM and recovered using OLD MINIMEM. The program is loaded almost instantly.

ASSEMBLY LANGUAGE ACCESS:
With the module a cassette is supplied with a 'line by line assembler' which provides a primitive and difficult to use method of writing your own machine code programs.

You will need to purchase the Editor/Assembler manual for information on the 99/4A Assembly language, and should be aware that the manual is not written for the novice.

The LBLA itself occupies the module, and the maximum machine code program you may write with it is therefore about 750 bytes.

A few machine code games are now appearing on cassette which can be loaded into the module.

The mini memory provides a low cost entry into the field of machine code programming, but at present no book suitable for the novice is available.

Machine code is a 'low level' language, which is not as easy to use as BASIC. Because the computer does not have to translate the commands, a machine code program may be as much as 1600 times faster than a TI Basic program.

EXTENSIONS TO BASIC:
The mini memory adds some commands for use in your TI Basic programs, allowing you to PEEK and POKE both CPU and VDP memory, and to obtain the hexadecimal string defining any character:

PEEK and POKE are used in many computers to look at and change the contents of one single memory location in the computer. The 99/4A console has 16k of user memory (RAM) known as VDP RAM, which is not directly addressable by the CPU (Central Processing Unit). The Mini Memory is the ONLY module available which allows you access to the VDP ram.

CALL PEEKV and CALL POKEV are used, and samples may be seen in preceeding chapter on advanced programming. They may be used to look at your PROGRAM, or to manipulate the SCREEN DISPLAY.

CALL LOAD and CALL PEEK are used to access the CPU RAM, which comprises of the 4k mini memory, the 32k expansion memory, and the 255 bytes of CPU ram in the console. CALL PEEK can also be used to examine the contents of CPU ROM (READ ONLY MEMORY).

CALL CHARPAT is used to obtain the defining string for a character, which you may then manipulate with SEG$

```
CALL HCHAR(1,1,94,760)
CALL CHARPAT(94,A$)
A$=SEG$(A$,1,14)&"FF"
```

CALL LINK permits a TI BASIC program to use a machine code utility or program stored in the Mini Memory with CALL LOAD.

CARE The mini memory contains a battery with a state life of two years, and will retain any data you load into it, even after the console is switched off and the module removed.

However, data is destroyed if you:

Insert or remove the module when the console is switched on.

Use CALL INIT or the INITIALISE option.

Use the module for something else.

Data in the module is also subject to corruption by static electricity, and you should not rely on it as a sole copy of your program or data. Always keeep a tape backup.

If you use the module as a data file, the contents can be saved to tape: thus you may store adventure text into the module with a BASIC program, and then copy the data onto tape easily using the 'S' option from the 'Easybug' selection from the main menu. Data is reloaded with 'L' option.

EDITOR ASSEMBLER:
The editor assembler package comprises a module, two disks and a large manual. One disk contains the 'source code' for one of TI's module games, to help you to understand the language.

Although large, the manual is not suitable for novices, and some information is difficult to find.

The 32k expansion memory, disk drive, disk controller and peripheral expansion box are REQUIRED for this package.

The EDITOR allows you to enter source code, and uses a good screen editor. When you are satisfied, the ASSEMBLER will turn your source code into MACHINE CODE in one of three formats chosen by you: Standard, required if you wish to run the program with the Extended Basic module. Condensed, which uses less disk space. Program format: which uses even less space but cannot be loaded by a BASIC program. The Editor Assembler and Mini Memory have special commands.

Programs you write in assembly language may run with the extended basic, mini memory or editor assembler modules, but you may need to use different coding for each:

As example, Extended Basic uses different internal memory mapping, and therefore you have to use different memory locations for example to print to the screen.

With Editor Assembler, you may run the disk versions of TI's games modules. Although providing much greater speed, assembly language is not for everyone.

The following is a PART of the source code of a program to DISPLAY AT:

```
DS      MOV R11,R10
        CLR R0
        LI R1,1
        BL @GN
        BL @LC
        DATA 1
        DATA 23
```

```
MOV @FC,R$
DEC R4
SLA R4,5
```

. . . which is adequate to demonstrate the difference to BASIC.

TERMINAL EMULATOR 2

The Terminal Emulator 2 module is designed for telecommunications, but as no suitable modem is presently available to connect your 99/4a to the Post Office network, you will not be able to use that facility.

Some major users of the computer use the TE2 module to link their 99/4A to 'main frame' computers, using the TI computer as an 'intelligent terminal', which has its own programs and passes data to and from the larger computer.

Of interest to domestic purchasers however, is the much improved speech facilities of the TE2 module. With a speech synthesiser connected, your program in TI Basic can say anything that you wish it to. You also have control over pitch and emphasis.

The method used is to open a file:

```
OPEN #1:"SPEECH",OUTPUT
```

and then when you wish your program to say something, you PRINT to this file:

```
PRINT #1:"I CAN SAY ANYTHING YOU WANT ME TO"
```

Speech is much faster with TE2, and because there are no limitations on the string printed to the file, you may adjust pronunciation by changing your spelling.

Would you like your computer to read your program to you? This can be of help when checking a listing to your program, looking for a missing line for instance.

Use:

```
LIST "SPEECH"
```

LOGO

LOGO is a language module, and requires the 32k memory expansion. Disk drive and controller are advisable.

LOGO is a 'build it yourself' language, in which you build up your own commands from a small set of 'primitives'. It is not therefore a program to exchange programs in, but a language to learn with, and is extensively used in a few primary and junior schools.

A redrafted version of LOGO to be known as LOGO 2 has been announced from TI, with greater user memory and added features, but at the time of writing was not available.

LOGO is of great interest to schools, and you may find it useful if you have young children, or an interest in creating your own language, or learning to express yourself in a clear and logical manner.

The following is an extract from a TI LOGO procedure, and informs the computer how to carry out the command:BLINK:

```
TO BLINK
TELL :ALL
SC :RED
TELL TILE 32
```

```
SC[4 15]
WAIT 40
TELL :ALL
SC :WHITE
TELL TILE 32
SC[15 4]
WAIT 40
BLINK
END
```

MULTI PLAN

Multi Plan is a 'spread sheet' program module, and requires the 32k memory expansion and a disk system. A printer (with RS232 card) are advised but optional.

The program was written by Microsoft (whose versions of BASIC are widely used in American computers), and is similar to a popular program called VISICALC (not available for the 99/4a). The idea of a spread sheet is to set up data in rows and columns and tell the computer of the relationships between certain figures. You may then investigate 'what if . . .' situations by changing certain data, and allowing the computer to change the remainder in accordance with the relationships between the data.

The use of spread sheets is complex, but in making business decisions helpful information can be quickly calculated and obtained.

Other Languages

The following are announced by TI. In addition independent sources may be able to supply other implementations of these languages:

FORTH is announced on DISK requiring Editor Assembler module and 32k memory, plus disk system. Not yet released.

PILOT is announced on disk for P Code card and 32k memory expansion. Not yet released.

PASCAL is available on three disks, requiring the P Code card and 32k memory expansion.

9

Peripherals

Systems:

The original expansion system comprised separate peripherals, each with their own power supply, which plugged into the right hand side of the computer and each other.

As the number of peripherals increased, this resulted in a number of electric supply cables, and the need for a very long desk.

Hence TI produced the PERIPHERAL EXPANSION BOX, which plugs into the right hand side of the console by means of a cable. It has its own power supply, but this is used for all the peripherals placed in it.

The TI BOX (as we shall call it) is supplied with a single interface 'card' which merely allows it to be connected to the 99/4a.

The 'cards' used in the Box are far more than the usual printed circuit board usually associated with expansion systems: each card is inside a strong metal sub chassis. The box itself is also very strong (and heavy) metal.

The box has space for a single disk drive, although it may be possible to use two low power 'half size' drives

mounted side by side. TI do not provide half size drives and you will need to have a well informed and helpful dealer if you wish to fit them.

The standard TI Disk Drive is a single sided single density drive which uses soft sectored disks with 40 tracks. Each disk can store about 90k of information, and each disk can contain up to 127 named files.

To operate the disk drive you need the DISK CONTROLLER card, which is supplied wilth a DISK MANAGER module. The controller card can operate up to three disk drives: the second and third must have their own power supply and case, and are used outside the Box.

The Disk Manager allows you to test your disk, initialise them, change file and disk names, and provide a catalogue of disk contents. You can use double density disks, but the computer will only use them as single density.

It is possible to modify the controller so that a double sided disk drive can be used, with the seond side treated as DISK 2.

A Disk Manager 2 module and double sided drive have also been announced from TI, but are not available at the time of writing.

A disk system allows you to load and save programs or data much faster than from cassette. Also, because a disk does not have to be read in the same order as it was saved, random access files are possible, for faster and more powerful data handling.

A program may be LISTed to disk as well as SAVEd to disk. A LISTed program is placed on disk in DISPLAY

VARIABLE 80 format, and may be used with the TI WRITER module.

Extended Basic permits a program to be saved in MERGE format, to allow programs to be merged into each other, and also allows you to manipulate the program : you may create utilities which remove REM lines, or shorten variable names for instance.

Also with Extended Basic, a disk system will allow you to load a program which exceeds the 12k allowed under the cassette loading system.

MEMORY EXPANSION

The 32k MEMORY EXPANSION CARD adds 32k of CPU RAM to your system. It is NOT usable unless a suitable module is used.

Extended Basic programs may be up to 24k when the 32k expansion is fitted, but the tape loader can only load programs up to 12k.

However, with extended Basic, when the 32k is fitted, in addition to the 24k for your program, you have about 13k for storing variables. Thus you may load a 12k tape program and not have to worry about the memory used for variables (for example, large arrays of data).

Also with Extended Basic, 8k is available to load machine code into, such as is needed in some of the TI module games (loaded from disk) or the utility programs which are becoming available.

The memory expansion is available to the Editor Assembler and the mini Memory modules for loading longer machine code programs from disk.

The Mini Memory can use the memory expansion as a data file.

Some TI Modules (such as Editor Assembler) require that the 32k be connected.

NB: The Personal Record Keeping module CANNOT use the 32k memory.

In extended basic a very small increase in speed is produced by using the 32k memory.

PRINTERS

The RS232 card is required to connect the computer to a printer. A number of independent suppliers are producing programs and hardware to permit printers to be connected without the box and RS232 card, but you should look closely at what their products can do, and try to see them working with your printer.

The TI RS232 card provides a potential three interfaces, two serial and one parallel.

A SERIAL connection is one in which the signal is sent one bit at a time. As a single letter is defined in one BYTE (8 bits) a serial interface has to break that down and send the bits one at a time.

The standard serial interface is known as RS232, and implies a standard connector and standard pin connections in that connector. However, there are variations in usage.

The TI RS232 interface permits you to set certain options in your Basic program, allowing a range of printers to be used. However, some printers have serial interfaces which operate at different signal levels, and you should

always try to see a printer working with the 99/4A before you buy.

There are a number of options available with the TI RS232 card, and you may need to check with your dealer the correct options to use with your printer.

The following are the options available with the RS232 interface:

BAUD RATE (eg speed) : 110,300,600,1200,2400,4800, or 9600
DATA BITS: 7 or 8
PARITY: Odd, Even or None
TWO STOP BITS: used or not
NULLS: used or not
CHECK PARITY: performed or not
ECHO: on or off
CRLF: on or off
LF: on or off.

These options should allow you to interface to almost any serial printer. The RS232 interface is bidirectional and can be used (if you have a P.O. approved modem) to link consoles by telephone for the exchange of programs or data. It is possible for data to be passed in both directions simultaneously using the RS232 interface.

For many printers, an RS232 interface is an option available at extra cost : the standard interface is PARALLEL. A parallel interface sends one byte at a time, and operates only in one direction. The TI RS232 card can send data at about 28000 Baud when using the parallel signal output, but the limitation will be the receiving device.

There are fewer options needed on the parallel interface, as a new signal is only sent when the printer tells the

computer it is ready for it by means of a 'handshaking' signal. It is possible to turn on or off the echo, automatic carriage return and automatic line feed features. Your choice will depend on the needs of your printer.

A few printers have a buffer memory, or one may be added: this allows the computer to send its data to the buffer and allows your program to continue while the printer takes data from the buffer and prints out your text.

NB: Although Centronics compatible, the TI parallel output uses a slightly different pin out, and your dealer will have to make a special cable for you. Ensure the printer you buy will work with your computer, before buying.

You DO NOT have to purchase the TI Printer. Most printers will work with the RS232 card.

SPEECH SYNTHESISER

The speech synthesiser has already been touched on in the discussions of Extended Basic and the Terminal Emulator Two modules.

The Speech Synthesiser requires a module to function. Some of the newer games modules (such as Parsec and Alpiner) use it, and you may use it in your own programs:

In TI BASIC: with Terminal Emulator 2 or Speech Editor

In Extended Basic: with the Extended Basic module.

Both Extended Basic and Speech Editor have limited vocabularies, but it is possible to extend them slightly by adding words together or curtailing them.

Terminal Emulator 2 gives the fastest and most realistic speech and permits pitch and emphasis to be changed.

A number of small companies are showing an interest in producing peripherals for the TI99/4A. These may be cheaper than the TI equivalent, but you must ensure that they satisfy your requirements before you buy, as they may not have all of the TI features.

You should always try to see any third party peripheral working with the computer before you buy.

NEW PERIPHERALS

TI have announced a new range of small low cost peripherals for their portable computer the CC-40.

An adaptor is due to be available for these to be used with the TI99/4A.

The peripherals announced are:

RS232 interface with serial and parallel ports.

WAFERTAPE drive: this uses special wafers containing endless tape loops. Faster than cassettes but not as fast as disks. May not support the same file handling commands as a disk drive.

FOUR COLOUR PRINTER: A very small printer which uses small pens to print with.

As so many modules and applications require an expansion memory, you may need to buy the larger box system (or a third party memory).

Many printers now permit you to "download" a screen display, using '8 pin bit image mode'. Most of the EPSON range of printers permit this for instance. Programs to USE these facilities are rare however, and one is therefore given below. This is in TI BASIC, but requires

the MINI MEMORY MODULE to function. The program will also run in EXTENDED BASIC.

Because the printer defines its characters vertically, but the computer defines its characters horizontally, this program will produce a picture of the screen on its edge : it is easier to do this than to rotate the image.

If the screen contains only text there is no advantage to using this program: Use GCHAR to obtain the characters and PRINT them one row at a time.

This program will print the characters as they appear on screen:

i. IN BLACK ONLY. ON pixels are printed, off pixels are not.
ii. Sprites are not copied.

```
100 OPEN #1:"PIO.CRLF"
110 REM  OR EQUIVALENT RS232 FILE NAME
120 PRINT #1:CHR$(27);"A";CHR$(8)
130 FOR [A=1 TO 32
140 PRINT #1:CHR$(27);"K";CHR$(192);CHR$(0)
150 FOR [B=24 TO 1 STEP -1
160 CALL GCHAR([B,[A,[CHAR)
170 IF [CHAR<33 THEN 300
180 CALL CHARPAT([CHAR,DEF$)
190 IF DEF$="0000000000000000" THEN 300
200 FOR [SEG=16 TO 2 STEP -2
210 [HEX=ASC(SEG$(DEF$,[SEG,1))
220 GOSUB 430
230 [PRINTDEF=[HEX
240 [HEX=ASC(SEG$(DEF$,[SEG-1,1))
250 GOSUB 430
260 [PRINTDEF=[PRINTDEF+[HEX*16
270 PRINT #1:CHR$([PRINTDEF);
280 NEXT [SEG
290 GOTO 340
300 FOR []=1 TO 7
```

```
310 PRINT #1:CHR$(0);
320 NEXT [)
330 PRINT #1:CHR$(0)
340 NEXT [B
350 PRINT #1:CHR$(13);CHR$(10
360 NEXT [A
370 PRINT #1:CHR$(27);"@"
380 PRINT #1:CHR$(7)
390 CLOSE #1
400 REM  NOW GO WHERE YOU WISH
410 REM  USING RETURN OR GO TO AS APPROPRIATE
420 STOP
430 [HEX=[HEX-48+([HEX>64)*7
440 RETURN
450 END
```

To permit the routine to be added to any program, the square bracket has been used in front of each variable: [.

The [is accepted as a valid character in variable names.

ESC K {CHR$(27);"K"} is used in the Epson printer, and some others, to select: 'Normal density 8 pin bit image mode'.

Appendices

GLOSSARY

ARRAY. A collection of variables referenced by a subscripted number.

ASCII. 'American Standard Code for Information Interchange' — standard code numbers for the characters used by the computer.

BASIC. 'Beginers All-purpose Symbolic Instruction Code'. An easy to use and widely used type of programming language.

BAUD. 'bits per second' — refers to the speed at which data is transferred to and from the computer.

BINARY. Our normal numbering system is DIGITAL, and uses numbers 0 to 9. A BINARY system uses only numbers 0 and 1, or OFF and ON. The computer works internally with signals which are OFF or ON.

BUG An error in a program, which causes incorrect or unwanted operation.

BYTE A group of 8 binary numbers (called BITS) used in computing to represent a character or command. Also used as a means of measuring a computer's memory capacity.

CONSTANT Used to describe a number or STRING, as distinct from a VARIABLE.

CURSOR A flashing character used by the computer to indicate that it is waiting for an input.

DISK A mass storage device used to store programs and data files.

FILE A collection of data records.

HEXADECIMAL
A numbering system with base 16. Instead of using number 0 to 9, it uses numbers 0 to 15, but letters A to F are used instead of 10 to 15.

LOOP A program line, or lines, which are performed a specified number of times.

PROGRAM A series of commands which tell the computer what to do.

RAM 'Random Access Memory'. Temporary storage in the computer, used for your programs. The contents are not retained when the console is switched off.

RECORD A collection of data elements. A group of records form a FILE.

RESERVED WORD
A word used by the computer as a command or function. such words cannot be used as variable names.

ROM Read Only Memory. Permanent memory which retains it's contents when the

	console is switched off. Contains the operating system of the computer.
RUN	An instruction to the computer to execute a program in its memory.
SCROLL	Movement of the screen display by one line upwards.
SOFTWARE	A name given to computer PROGRAMS.
STRING	A series of letters, numbers or symbols, treated as a single unit. A single number may be treated as a number OR as a string but cannot be both at once. 2 is a number. "2" is a string.
VARIABLE	A name or label which has a value which may be altered during a program.

KEY CODES

It is sometimes useful to enter ASCII codes from the keyboard outside the usual range — for instance when PRINTing a line of defined characters.

By switching the computer to PASCAL mode, using CALL KEY (4,A,B), the range of codes available is increased.

Although the usual function codes (eg cursor control) are deactivated in Pascal mode, upper and lower case characters are not affected.

The following table gives the codes available in PASCAL mode. The keys to be pressed are indicated thus:

A = Key A only
FA = FCTN and A together.
CA = CTRL and A together.
SA = SHIFT and A together.

CODE:	KEYS:	CODE:	KEYS:
0	C,	31	C9
1	CA	32	SPACE
2	CB	33	S1
3	CC	34	FP
4	CD	35	S3
5	CE	36	S4
6	CF	37	S5
7	CG	38	S7
8	CH	39	F0
9	CI	40	S9
10	CJ	41	S0
11	CK	42	S8
12	CL	43	S=
13	CM	44	,
14	CN	45	S/
15	CO	46	.
16	CP	47	/
17	CQ	48-57	0-9
18	CR	58	S;
19	CS	59	;
20	CT	60	S,
21	CU	61	=
22	CV	62	S.
23	CW	63	FI
24	CX	64	S2
25	CY	65-90	A-Z
26	CZ	91	FR
27	C.	92	FZ
28	C;	93	FT
29	C=	94	S6
30	C8	95	FU

CODE:	KEYS:		
96	FC	177	C1
97-122	a-z	178	C2
123	FF	179	C3
124	FA	180	C4
125	FG	181	C5
126	FW	182	C6
127	FV	183	C7
128	na	184	F.
129	F7	185	F.
130	na	186	F/
131	F1	187	C/
132	F2	188	F0
133	na	189	F;
134	F8	190	FB
135	F3	191	FH
136	FS	192	FJ
137	FD	193	FK
138	FX	194	FL
139	FE	195	FM
140	F6	196	FN
141	na	197	na
142	F5	198	FY
143	F9	199-	na
144-176	na		

To use:

ENTER PASCAL MODE WITH CALL KEY(4...

AFTER THAT USE THE KEYS INDICATED AND THAT CHARACTER WILL BE PRINTED: IT MAY NOT BE VISIBLE IF YOU HAVE NOT DEFINED IT.

In the section on advanced programming the use of single byte control codes is explained. The following are the meaning of the various codes. NB: They are not all available in TI Basic or Extended Basic.

CODE;	MEANING:		
129	ELSE	159	OPEN
130	:: (Ex Bas Separator)	160	CLOSE
131	! (Ex Bas tail rem)	161	SUB
132	IF	162	DISPLAY
133	GO	163	IMAGE
134	GOTO	164	IMAGE
135	GOSUB	165	ERROR
136	GOSUB	166	WARNING
		167	SUBEXIT
137	DEF	168	SUBEND
138	DIM	169	RUN
139	END	170	LINPUT
140	FOR	171-175	Not known
141	LET	176	THEN
142	BREAK	177	TO
143	UNBREAK	178	STEP
144	TRACE	179	,
145	UNTRACE	180	;
146	INPUT	181	:
147	DATA	182	)
148	RESTORE	183	(
149	RANDOMIZE	184	&
150	NEXT	185	Not known
151	READ	186	OR
152	STOP	187	AND
153	DELETE	188	XOR
154	REM	189	NOT
155	ON	190	=
156	PRINT	191	<
157	CALL	192	>
158	OPTION	193	+

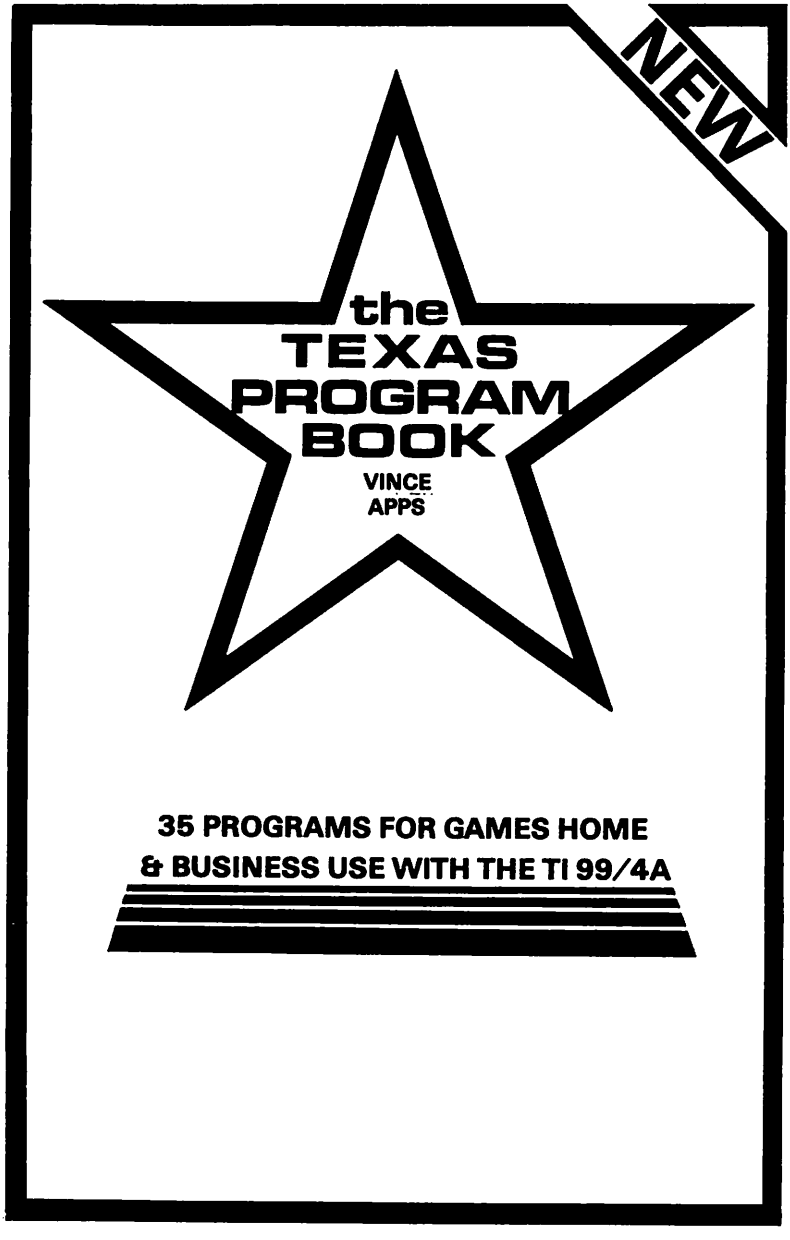
CODE:	MEANING:
194	-
195	*
196	/
197	^
198	unknown
199	'string follows'
200	'number follows' ALSO used with CALLs.
201	LINE NUMBER FOLLOWS
202	EOF
203	ABS
204	ATN
205	COS
206	EXP
207	INT
208	LOG
209	SGN
210	SIN
211	SQR
212	TAN
213	LEN
214	CHR$
215	RND
216	SEG$
217	POS
218	VAL
219	STR$
220	ASC
221	PI
222	REC
223	MAX
224	MIN
225	RPT$
226-231	Not known
232	NUMERIC
233	DIGIT
234	UALPHA
235	SIZE
236	ALL
237	USING
238	BEEP
239	ERASE
240	AT
241	BASE
242	Not known
243	VARIABLE
244	RELATIVE
245	INTERNAL
246	SEQUENTIAL
247	OUTPUT
248	UPDATE
249	APPEND
250	FIXED
251	PERMANENT
252	TAB
253	# (for files)
254	VALIDATE

Index

Page Nos.

If you are interested in cassette based software written by the author of this book please send a S.A.E. for prices and details to

Stephen Shaw
10 Alstone Road
Stockport
Cheshire


NEW
the
TEXAS
PROGRAM
BOOK
VINCE
APPS
35 PROGRAMS FOR GAMES HOME
& BUSINESS USE WITH THE TI 99/4A

GETTING STARTED with the TEXAS TI-99/4A

STEPHEN SHAW

Aimed at first time users,
this book takes you from setting up your machine
and guides you step by step
until you become sufficiently expert
to write your own programs.

This *'essential'* book
will help you
use TI Basic
understand Extended Basic
design programs
file data on cassette

Example programs are shown in the book
along with chapters on how to increase
the capabilities of your machine
through modules, printers, RAM packs
and disk systems.

THE AUTHOR

Stephen Shaw is a recognised authority
on the TI99/4A
and contributes regularly to the *TI User*,
Home Computing Weekly and *Computer & Video Games*.

ISBN 0-9465-7604-1

90000

9 780946 576043

PHOENIX
PUBLISHING ASSOCIATES

£5.95