TI LINES

Whilst reading TI-LINES you will find Peter reporting that ZXC is about to do HGT or that UUU is about to be on sale.

Alas TI-LINES contained a lot of rumours that have no basis and a large amount of vapourware.

Many things reported by Peter didn't happen. He was a very credulous reporter.

This does not detract from his lengthy articles, but do treat the reports of new and coming items with a large pinch of salt.

2021   bb

NEW LOOK!

TI-LINES

Volume 1, Issue 3, June 1st., 1984

TI-LINES is the monthly newsletter of the OXON TI USER group and the presentation of any material herein is copyright of each individual author.

TI-LINES is produced and published by:

Peter G. Q. Brooks
29 Kestrel Crescent
Blackbird Leys
OXFORD OX4 5DY

Telephone OXFORD 717985 after 8 p.m.

TI-LINES is available to blind/partially-sighted Users on audio cassette - contact the publisher for details. It is also available on associate subscription + Users outside Oxfordshire for £10 p.a. Back issues are £1 including post and packing.

Contributions to TI-LINES are welcomed. It is the responsibility of each individual author to ensure that their material does not infringe copyright, and that it is presented in as legible a form as possible, when it may be subject to retyping and limited editing. Any artwork should fit within a maximum A4 area and should not contain colour. Very high contrast line drawings are preferable and these may be produced by arrangement with the publisher.

Index to articles

EDITORIAL

Hello and welcome to the first of the subscribed copies of TI-LINES, and to a new format. I have managed to obtain reduction-copying facilities and it is my intention to replace the original A4 size issues with A5 booklets as soon as it is practical, thus making any collection of TI-LINES consistent.

As a result of Home Computing Weekly publishing a letter from me concerning Oxon TI Users I have had many enquiries from owners resident outside Oxfordshire who would like membership details. My letter mentioned the facility whereby TI-LINES is read onto cassette for the benefit of blind/partially-sighted Users, and my willingness to expand that side of things if there was sufficient demand. HCW readers took that to mean that they were being offered subsidised (£1.50 p.a.) membership as well, and a dozen letters arrived over the following few days. I sent a circular back to each one explaining the state of affairs, and offered Associate membership for £10 p.a. To date only one enthusiast has taken up the offer and to Mr B. MUNRO SCOTT a warm welcome. I hope that we can provide you with plenty of information and entertainment over the coming months and that you will feel free to contribute anything and everything which you think might be of interest.

The Elite Eight mentioned in the last editorial didn't quite make it but their numbers were made up by others. I also send complimentary copies out to ROBERT BATTS of TI who has been extremely helpful in many fields over the last several months; to CLIVE SCALLY of TI-99/4A EXCHANGE who organises a national User group; to PAUL DICKS, who is the founding father of the TI User groups in this country and who is still active - TIHOME still exists as a software collection which Paul administers as well as being a columnist on the newsletter which is put out by his successors; to STEPHEN SHAW, who, as I write, has just become the father of a bouncing baby boy - so to Stephen and Cathy congratulations and when will George start programming ?

Our total strength (measured in TI-LINES) is 14 and I hope that it grows - but not too much or I'll go broke!

I learned an object lesson last issue when I published DAVE HEWITT's modulator modification article. As a result of severe postal difficulties (the GPO lost not only his article but also a disk into the bargain) Dave sent me his original rough copy from which I extracted what I thought were all the pertinent details. However, I didn't feel competent to reproduce his diagram as I lack the specialist knowledge to be certain of not making any mistakes. That proved to be my downfall as I missed important data which had been placed on the diagram concerning the 47uF capacitor. An errata page elsewhere deals with that and with my second downfall (object lesson number 2): I didn't even copy my own notes carefully enough when producing the ONES & TWOS COMPLEMENT article last issue. There are two major errors in the explanation which make a mockery of the whole thing - well, almost. That too is dealt with in the errata section. My apologies both to Dave and to anyone who has been going bald trying to make sense of both articles!

STOP PRESS: For all the younger TI enthusiasts there will be a section devoted to their ideas and interests. It will be edited by DAVID BROWN of 59 Appleford Drive, Abingdon, so if you have anything please send it to him for consideration. His phone number is Abingdon 24813.

I've had a few questionnaires back and I'm beginning to get an idea of the general interests and equipment of members - well, of five of them anyway. Most have indicated that they would be willing to advise and assist fellow members and I will publish relevant details in the next issue. Thank you for your help and I will do my utmost to see that your needs are met.

Finally, I would welcome reviews from members of programs, books, etc., etc. and anything else which is publishable - articles, programs, letters, criticisms - anything which lets you play the game instead of just spectate.

BULLETIN BOARD

FOR SALE FOR SALE FOR SALE FOR SALE FOR SALE FOR SALE FOR SALE FOR SALE FOR SALE

GARY HARDING of Flat 2, 12 Belmont, Lansdown Road, BATH BA1 5DZ is selling some of his equipment - items which he will not be using because he has replaced them. He isn't on the phone, so please write to him in the first instance. Here is a list of his items:

Terminal Emulator II module:	£19
Speech Synthesiser:	£19
Personal Record Keeping module:	£19
Blackjack and Poker module:	£ 4
Complete collection of the Scott Adams Adventure cassettes:	£ 4 each
RS232 card:	£60
Microline 82A printer	£200

MRS. E. ROBERTS, Abingdon 21215, is also selling some equipment - a complete setup for £125, which she would like to sell all at once. It comprises:

TI-99/4A Console

Extended BASIC module

Speech Synthesiser

Joysticks

a number of modules including PARSEC, The ADVENTURE module, and PIRATE, VOODOO CASTLE, ADVENTURELAND, and various other cassette games.

Please contact Mrs Roberts if you are interested or know of someone who wants to buy a complete setup as either a starter or as a back-up.

- -

As a rider to the items above, if you are wondering about buying a Speech Synthesiser then there is a series coming shortly which will explore it as never before, covering not only the in-depth analysis of the process used to produce the high-quality speech which TI have kept quiet about for the last four years (Pitch-excited Linear Predictive Coding) but also details about how YOU can modify the speech data which is already provided (raising the pitch of the voice - or lowering it, and initial experiments with the REFLECTION COEFFICIENTS in altering the pronunciation) and about the production of all sorts of sounds and effects. This project has been three years in the making and is still far from finished.

- -

B E G I N N E R ' S   B A S I C   I I

By Peter Brooks (originally published in Tidings, V2.5, October 82)

T h e   F i r s t   S t e p s

This series will attempt to slowly introduce the novice to BASIC on the TI-99 and to the principles of 'programming'. It is not intended as a replacement for the TI manual, but will try to offer a different perspective so that your chances of understanding the commands and functions are increased. If, as the series progresses, there are still points which you feel have been inadequately covered, please write in with details of the problem area(s).

If you have ever used a calculator which possesses one or more 'memories', you will be familiar with the idea of 'storing' numbers in a machine. On calculators, however, the memories tend to be identified by numbers also: Memory 1, Memory 2, and so on, so that storing 26.5 in Memory 7 is achieved by keying '26.5 STO 7', where STO is the store command. To recall the contents of Memory 7 to the display, you might use 'RCL 7', where RCL is the 'recall' or 'retrieve' command.

On computers, a similar system exists. In this case though, the memories are not preset as they are on calculators, and their identification is made by 'name', not number. Thus the computer 'memories' can be called A or SOCK or MEMORY or EGBERT or even something which gives a clue as to what is being stored there - for example, if you needed to store the number 365 as part of a calculation involving Time in years, you might call the memory DAYS.

BASIC's way of storing and recalling numbers is different from that used on the calculator. To store 365 in DAYS, BASIC uses:

```
LET DAYS = 365
```

The number or 'expression' (equation) on the right of the equals (=) sign is 'assigned to' (stored in) the memory indicated on the left of the equals sign; in this case, the memory is called DAYS. On the TI computers, as on many others, the use of LET is optional - which means that if you leave it out, the computer won't throw a fit - so that:

```
DAYS = 365
```

is also acceptable. The memories in computers (which are really storage areas inside memory chips) are called VARIABLES, and are more versatile than on most calculators. You can use one sort of Variable to store numbers - NUMERIC VARIABLE - and another sort to store letters (letters, numbers as sequences of digits, punctuation marks etc.) called STRING VARIABLE. To enable the computer and you to distinguish between a variable which stores numbers from one which stores letters, the name of the letter-storing (STRING) variable has a dollar sign tacked onto the end. (There are other types of variable, but they are not available on the Texas.) The assignment (storage) of data to such variables is also carried out differently:

```
DAY$ = "MONDAY"
```

('$' is the closest that this typewriter gets to a dollar). Whatever is to be assigned to a string variable needs to be enclosed in quotation marks. There are other ways of assigning data to these variables which don't require the use of quotes, but we will delve into that later.

Another difference is that although you can assign a string of digits (a number!) to a string variable, that variable can't be used directly in expressions (equations). It CAN be used indirectly, but again, we will delve into that later.

If you find it difficult to think of the variables in terms of 'string' and 'numeric', it might help you to think of them as 'text' and 'mathematical' or 'words memories' and 'number memories'. As long as you are familiar with the terms NUMERIC and STRING that's all that matters.

We'll try out a few examples on the TI-99s. In order to avoid repeating instructions, I will 'indent' anything which is to be typed into the computer followed by ENTER, thus:

```
        ICKYPOO = 2
```

As you can see from the name of the variable, this is going to be a rather informal series. Switch the computer on and select TI BASIC. We will initially be using the computer in the so-called IMMEDIATE MODE, where the computer will execute any instructions there and then, instead of storing them away for later execution as in a PROGRAM.

Let us try out one or two things.

```
        DAYS = 365
```

If you have remembered the instructions earlier, you will have typed DAYS = 365 and then pressed ENTER. The computer will have created a Numeric Variable called DAYS and assigned the number 365 to it. The question now is: how can we recall the contents of the variable to the screen ? What is the recall or retrieve instruction ?

There is a command available, called PRINT, which allows us to print data on the TV screen. If PRINT is used with a numeric variable, it will fetch whatever number is currently assigned to that variable and put it on the screen.

```
        PRINT DAYS
```

The number 365 will be printed underneath the instruction. It appears with both a leading and a trailing space (one in front and one behind), which you cannot actually see. The leading space will be replaced by a minus sign if the number is negative. The trailing space prevents any following information from being confused with the number, a type of 'formatting' of the display. Formatting can involve a lot more, but again we can deal with that subject in more detail later in the series.

Let's try some other variables:

```
        NAME$ = "DONALD DUCK"
```

If we try printing the contents of the string variable called NAME$,

```
        PRINT NAME$
```

'Donald Duck' is printed underneath without any leading or trailing spaces, which is something to bear in mind for future use.

```
        PRINT NAME$ ; NAME$
```

Now, if you ENTERed this correctly with a semi-colon (;) between the two NAME$, your display should now show a line which reads 'DONALD DUCKDONALD DUCK'. The semi-colon is known as a PRINT SEPARATOR, and tells the computer that once it has printed something it should wait on the same line, as the next PRINT instruction is to have its information printed directly after the previous information. This also demonstrates the lack of spaces between the text! There are two other PRINT SEPARATORS: they are the comma (,) and the colon(:).

```
        PRINT NAME$ , NAME$
```

This time the second printing places the 'DONALD DUCK' about halfway across the screen from the beginning of the first printing. The comma separator tells the computer to begin printing the next piece of information at the halfway point, unless it has already passed that point, in which case it will print on the beginning of the next line down.

```
        PRINT NAME$ : NAME$
```

This time the two items are printed one under the other; the colon tells the computer that it should begin printing the next item on the next line. The colon is actually a very handy separator in that if you use several of them together you can print several blank lines between items, or even just print several blank lines after the last printed item, which causes the screen's contents to 'scroll' upwards; for example:

```
        PRINT :::::
```

A quick recap, then, on PRINT SEPARATORS: after the data in an instruction to PRINT, a semicolon means: 'hang about, there's more to go next to this lot'; a comma means: 'start the next lot halfway across the screen, unless you're already past it or the next lot is more than half a screen's width in length,in which case start at the beginning of the next line'; and a colon means: 'start the next lot at the beginning of the next line'.

The PRINT instruction is a very flexible one, and there are more facets to its format which we will look into later.

```
        A = 10
        A$ = "GREEN BOTTLES"
        PRINT A ; A$
```

The above is an example of the kind of thing that you can do with separators. You have told the computer to store the number 10 in a 'number memory' or NUMERIC VARIABLE called A, and to store the phrase "GREEN BOTTLES" in a 'word memory' or STRING VARIABLE called A$ (pronounced 'A-STRING'), and then to print the contents of the numeric variable A, to wait on the same line, and then to print the contents of the string variable A$ straight after the first item. The result on screen is: 10 GREEN BOTTLES. You may now gasp with amazement.

Now let us introduce the dreaded LINE NUMBERS. Suppose that we wanted to store the three statements above without executing them there and then, and check that the computer had stored them correctly ? Let's call them statements 1, 2, and 3 (complicated, eh ?):

```
        1  A = 10
        2  A$ = "GREEN BOTTLES"
        3  PRINT A ; A$
```

Note that by giving each statement a number, you have told the computer not to execute each statement as it is entered, but to store it and wait to be told when to do something else. The computer will list out the statements in the right order when you tell it to:

```
        LIST
```

and you can check to see if everything has been entered correctly.

Finally, we tell the computer that we'd like it to execute our statements, by giving it the instruction:

```
        RUN
```

whereupon 10 GREEN BOTTLES appears on screen. The computer screen colour changes from cyan (a sort of pale blue) to green for the time it takes to execute the three statements, and then returns to cyan when finished, as well as printing DONE to make sure that we know that the computer is ready to accept further instructions. The CURSOR (that little black square which keeps flashing on and off) also returns.

In effect what you have done is to enter a PROGRAM into the computer, admittedly not a particularly scintillating one, but a program nevertheless. Ah, but... I hear you say,...we've seen REAL programs in the magazines, and they don't have nice neat 1, 2, 3 line numbers, they have 10, 20, 21, 30, or 100, 105, 106, 110, 120, line numbers. How come ?

There's no great mystery. If you wanted to insert another statement in our little program to get the screen to scroll a little first before printing 10 GREEN BOTTLES, by adding PRINT ::::: as a first line, you'd need to rewrite the whole program, UNLESS...you'd already written the program with big gaps between the line numbers just so that you could insert any extra lines as and when you needed them. On some computers you have to do things this way, but on the 99s, as on many others, there is a command which can help you out:

```
        RESEQUENCE

        LIST
```

Well, well, our program has changed a little. The RESEQUENCE command, RES for short, which the computer will also respond to, has changed the first statement number into 100, the second into 110, and the third into 120. As standard, the computer starts renumbering at 100 and goes up in steps of 10. You can even specify what the first line number will be, and what the step or 'increment' will be. For example:

```
        RES 117 , 26

        LIST
```

See what I mean ? You can now add the extra line of PRINT ::::: as say line 17, and then RES again to smarten it all up. Experiment a little and see what you can find out!

- -

Any queries or contributions which members may have with respect to the BEGINNER'S BASIC series are welcomed. Eventually the series will be compiled into a booklet and sold, probably through the numerous small software houses which are still supporting the TI-99/4A.

The next issue will see coverage of the keyboard functions, editing, different versions of the 99s available, maths while PRINTing, INPUT, GOTO, and CALL CLEAR.

- -

E R R A T A

V1.2 pp 3 - 4 MODIFY YOUR MODULATOR by DAVE HEWITT

The exact location of the 47uF capacitor (or condenser - the terms are interchangeable) was omitted for complex reasons. It should be soldered on the one side to the inner cable of the video lead going to the video socket, and on the other side to the 1 kilohm resistor, taking care not to apply too much heat or the resistor's connection to the circuit board may come away.

In addition, there are at least two different types of modulator at large, and the article does not apply to modification of the type housed in a plastic casing.

- -

V1.2 pp 13 - 15 ONES & TWOS COMPLEMENT by PETER BROOKS

Page 14 shows that I don't pay attention to my own notes: fourth paragraph down on the subject of counting negative numbers - the range should be 0 to -127, not -128.

Page 15 shows that I can't copy my own notes either - the intention was to show that although the range was 0 to +127 for the positive side, it was actually 0 to -128 for the negatives.

Also on page 15, I neglected to add that the rule of adding the signed and unsigned numbers to total 256 only works for NEGATIVE numbers - you might have noticed that 3 + 3 has a hard time adding up to 256!

- -

L E T T E R S

Even though he has his hands full at the moment, STEPHEN SHAW (see Editorial) has responded to Gary Harding's plea for help in the Bulletin Board in V1.1. You may remember that Gary had problems with his printer and RS232. I asked for anyone in OTIU to come forward and help out, but no-one was able to. In the meantime I had written to Stephen and to Gary, and had learned that Gary's printer and RS232 had both received a clean bill of health from a BBC micro (it managed to drive the printer without any trouble) and AKHTER INSTRUMENTS (they checked his RS232 and found nowt wrong). Stephen wrote back with the following information, about which Gary had not been aware - and as it happens didn't make any difference to his problems - and which is well worth while reproducing here for the benefit of those who may eventually find the folding stuff to expand their systems.

There should be a sticker inside the RS232 manual which tells you that the details on how to connect up the pins are wrong. To connect the RS232 card to a printer you need an eleven line ribbon cable. Pins 1 to 9 are connected to each other (1 to 1, 2 to 2, etc.,) <u>BUT</u> TI pin 10 (handshake in) connects to PRINTER pin 11 (busy).

TI pin 16 could go to PRINTER pin 16, but it is easier and neater to connect TI pin 11 (logic ground) to PRINTER pin 16 (Ov LGC GND). No other printer pins need to be connected.

Stephen also noted that he now has three consoles with three different keyboards, and three different modulators!

I often receive current news from Stephen and others, and from time to time I will run a News spot in TI-LINES - probably just half a page - to make sure that we all keep up with things.

THE CONTROL AND FUNCTION KEYS

PART III

Peter Brooks

Before launching into the third (and possibly final) part of this long and rambling episode on the Control and Function keys (mostly Control, you'll probably have noticed) I need to clear up some unanswered questions from the last article.

If you entered and ran the program given on page 9 of V1.2 TI-LINES and left it running long enough, you should have observed two things (or possibly three - it depends on how long you ran the program). When the program begins running, the first thing that you should notice is the small shapes in the bottom left-hand corner of the screen. The rightmost shape will be changing shape pretty rapidly, and occasionally the whole group may flicker. The second thing which should soon make itself known is a sudden streak of 'insects' which begins at the bottom right hand corner of the screen and shoots across to the left. The third effect only appears after several more seconds, when the flickering shape at the left ceases moving for a few moments - there may also be a small 'ripple' through the line of insects, causing them to advance towards the left. This pausing occurs every minute or so, but the rippling effect may cease.

So what are they ? The first effect is caused by the computer 'working' on your program. Remember that I said that the User-definable Graphics (UDGs) occupied the same area of memory as the computer used while working ? The shape-changing is due to the UDGs acting as a window on the execution of the loop in lines 150 - 170. In effect, the computer is counting while we watch.

The second effect, the streak of insects, is a little more difficult to explain - mostly because of a lack of detailed information on the processes used by the 99/4A when storing data. When a program instructs the computer to store data in a variable (whether numeric or string doesn't really matter), the computer locates an area in memory which is not being used and stores the data there. It keeps a record of where it has got to as far as storage space goes, so that when the program instructs the computer to store something else, the computer then locates the next free area to store it in. In the program we are using, the computer is instructed to store the same data over and over again. Once it has finished filling the TØ() array, it has to start all over again, but being stupid (as <u>all</u> computers are) it doesn't use the areas which it has finished with, it keeps on using fresh areas until it runs out of memory.

What happens then is the third effect, called GARBAGE COLLECTION (you'll find a large number of terms in Computing which relate to domestic activities - another is HOUSEKEEPING). This is what occurs when the computer pauses. It is 'clearing out' if you like, all the data which is not currently required. This probably involves resetting the internal record of where free areas of memory are located, rather than by actually removing the data. Considering the speed at which computers work when not coping with programs in BASIC, the 99/4A seems to spend rather longer doing its household chores than one might expect.

On top of all this, (as if it wasn't enough), the 'HELLO MA' shape which you see appearing in the UDGs is likely to be in transit - in the process of being stored temporarily in a work area or a 'buffer' (a kind of reception area or waiting room, if that helps you to picture it) - rather than actually being in its final position.

Right, that's got that out of the way, on with the business of the day. So far we have limited our discussion to those tokens which are available directly from the keyboard. However, it is possible to obtain some (if not all) tokens (and therefore characters) using an 'indirect' method.

Before entering into a detailed discussion of the technique and its use as a research tool, let us look briefly at the EDIT mode in TI BASIC. This mode is entered when you type EDIT and a line number followed by ENTER (or FCTN X or FCTN E), or when you type the line number followed by FCTN X or FCTN E. When you do this, the numbered statement (if it exists) is presented on the screen in its entirety - including the line number - and the cursor sits over the first character in the statement. (In Extended BASIC, it is possible to place the cursor over the first digit in the line number using a little trick with FCTN 8 - REDO - and edit the line number as well).

If you then press ENTER or FCTN X or FCTN E without having previously pressed any other keys (or almost any other), the existing copy of the line currently held in memory is left unaltered, but more importantly, no check is made of the line you are 'entering'. If you do type something - even simply overtyping a single letter - the computer reacts as if you were entering the line for the first time. Similarly, if you change a line, then decide to leave it as it is, unless you use FCTN 4 and BREAK out of the editing the computer will examine the line when you enter it, and this can be a very, very slow process in a large program.

This re-examination can be demonstrated easily using tokens. Last issue I offered a quirk for general investigation. It involved entering a REMARK consisting of a large number of CTRL Us to obtain a very long statement. This time, produce a REM containing only 20 CTRL Us and enter it. Call it up for editing with (say) FCTN X, and press ENTER (or FCTN X or FCTN E - all three act like ENTER except under certain circumstances).

Nothing devastating happens. Call it up again, and this time, with the cursor positioned over the R in 'REM', type R again. Note that you haven't changed the line - just retyped a single letter. Press ENTER now, and bang! LINE TOO LONG appears.

Now create another REM, this time containing a single CTRL /. Enter it and then call it up for editing. Note that there seem to be far more characters than you originally entered. (See TI-LINES V1.2, page (v)). With the cursor over the R in 'REM', type R (to con the computer into scanning the line again) and press ENTER. Call the line up for editing yet again, and once more the statement will have changed. Why should this be ?

If you examine the tables in V1.2, pages (i) to (v), you'll notice that certain key codes have 'UNUSED' written next to them. These are codes for which there is no token assigned. The computer, however, still attempts to find a token, and as the token words (LET, PRINT, etc.) are probably related to the token codes (141, 156) and their location in memory, it finds 'garbage' and prints that. Page (v) shows the codes of the garbage for those unused codes which are accessible directly from the keyboard. CTRL / gives a sequence of characters whose codes are 197, 62, 181, ... 126. This is what you saw at the first editing. When you typed R, you caused the computer to scan this sequence, and subsequently when you called the 'new' line up for editing, the computer replaced those tokens (codes greater than ASCII 127) with, in some cases words or symbols, in others, further garbage.

Looking at the sequence for CTRL / on page (v) and the lists of tokens on pages (i) to (ii):

197	:	^	62	:	>	181	:	:	251	:	PERMANENT
182	:	)	55	:	7	253	:	#	254	:	Unused
183	:	(	255	:	Unused	87	:	W	223	:	Unused
135	:	GOSUB	121	:	y	255	:	Unused	245	:	INTERNAL
79	:	O	126	:	~						

A small number of the characters here are not obtainable directly from the keyboard: 223, 245, 251, 253, 254, and they in turn will produce sequences, possibly also containing codes not directly obtainable from the keyboard. I have yet to tabulate these secondary (and tertiary, quaternary, etc....) codes - perhaps readers might like to compile their own tables and submit them to me, and any important differences will be published in a later TI-LINES.

At this point you might be saying to yourselves - So what ? None of these characters can be redefined by us using TI BASIC, so why bother ?

There are two reasons. The first is universal: because it can be done. Experiment for all you are worth and you'll always end up knowing more than when you started. The second is because you now have a valuable research tool. You can place special codes in listings, which TI never intended you to do, and this has two consequences. Firstly, you can 'embed', or hide, special characters in your program listings which can be of immense help if you ever have a dispute over illicit copying (piracy). This involves ASCII code 0, and the technique for using it is discussed later. Secondly, you can now begin looking for undocumented subprograms (an article later this year will begin to look at this intriguing area of research).

For example, the PERSONAL RECORD KEEPING and STATISTICS modules possess CALLs A, D, G, H, L, P, and S. There is nothing too remarkable about that - the CALLs are well documented now, if not widely known. (They form the basis of Enhanced BASIC, on which subject there will be a series of articles starting soon). However, RICHARD BLANDEN of Wokingham has informed me that, using machine code access through other equipment, he has uncovered more subprograms on those two modules. Using this new research tool in TI BASIC without the need for additional equipment, we can begin to investigate matters for ourselves.

How, though, can we isolate the characters we want from a REM statement ? Page (iv), TI-LINES V1.2 shows one way of doing it. You can either edit a REM statement to fit your purpose:

Enter: 100 REM CTRL /

Call it up for editing and change REM to A$=" and then run the cursor as far right as it will go and then add a trailing " to give:

100 A$ = " see the list on the previous page

or you can make the REM contain everything you'll need:

Enter: 100 REM A$ = "CTRL /"

You edit line 100 and delete the letters R-E-M and then enter the new line.

If you type in the program on page (iv) of issue V1.2 you can then place any single CTRL character between the quotation marks in line 100, edit the line, deleting the REM, then enter it and then RUN. You'll get a printout of the ASCII codes of the characters which make up the garbage.

When however you need to isolate a single character, you must carefully delete those characters which lie either side of the one you want until you are left with just that character within the quotes.

Try and obtain character code 254 on its own from the sequence produced by CTRL /. You need to edit to get the garbage between the quotes. Then delete the 'REM' letters, skip over the first quotation mark and carefully delete 7 characters. Move the cursor one position to the right using FCTN D and continue deleting up until the right hand quotation mark. Press ENTER and then RUN the program. If you don't get 254 printed on screen, you've slipped up somewhere. You will not be able to re-edit the line, because whatever is now within the quotes may be replaced by tokens when you try to LIST or EDIT. You will have to set up the line afresh and go through the process all over again.

We come finally to CHR$(0) and its special properties. As far as the INTERPRETER is concerned, CHR$(0) is what it uses to signify the end of text in a REM statement (an end of text marker). The Interpreter is the built-in program which runs your BASIC programs.

When you put a CHR$(0) in a REM statement, the interpreter thinks that it has found one of its own end of text markers, so any text which appears after the CHR$(0) will not be listed. It will still exist in memory, though, and will be saved to disk, cassette, or other storage medium - for example, the MiniMemory module. You can therefore place copyright notices and other data in a REM which cannot be seen but which can be recovered through the use of PEEKVing with MiniMemory or through PEEK and Extended BASIC, or even through Editor/Assembler. It won't appear in any listings either.

There is one drawback (or possibly two). Firstly, if a copy of your program is made using either direct tape-to-tape copying, or by loading and then saving to a copy tape, your hidden ("embedded") copyright notice will be transferred. However, if the pirate makes a listing of your work and then enters that at his own keyboard, obviously no copyright information pertaining to you is going to be added.

The second possibility is that your program might be edited before being copied. Bearing in mind the technique that we have been investigating, where we make use of the fact that editing a line causes it to be re-scanned, you can see that all an illicit copier has to do to inactivate any copyright information which may have been hidden is to edit every REM line in the program. All that has to be done is for any REM line to be called up for editing and 'R' to be pressed in order to bypass any embedded text.

The program listing below has had two pieces of text embedded in it. Its purpose is simply to 'look at itself' and print out what it finds on the thermal printer. Due to the tokenisation process, much of the printout is nonsense, but you can see quite plainly what the embedded material is.

It has been rumoured that at least one UK software firm has had its programs pirated; perhaps the use of CHR$(0) in future might make life a little less difficult for those trying to protect their property from thieves.

```
100 OPEN #1:"TP.U.E",OUTPUT
110 REM  EXAMPLE PROGRAM
120 REM  COPYRIGHT PGQB 1984
130 REM
140 REM
150 REM  PRINTOUT OF PROGRAM TO
THERMAL PRINTER
160 REM
170 REM
180 FOR L=16060 TO 16383
190 CALL PEEKV(L,B)
200 PRINT #1:CHR$(B);
210 NEXT L
220 CLOSE #1
230 REM
240 REM
```

```
?I  ?P ?T  ?  I>_  ?  I?|  ?
?( ?_ X?_ N?_ D?_|| L  16060
16383 $    EMBED OBTAINED THROUG
H FCTN 0   8    EMBEDDED COPYRIG
HT NOTICE NOT VISIBLE IN LISTING
S  |   |   AT .1 | L     .1  _
B T   PEEKV_L_B  |   *  PRINTO
UT OF PROGRAM TO THERMAL PRINTER
  |       COPYRIGHT PGQB 1984  ||
 EXAMPLE PROGRAM  ||  .1 _ATP.U.
E__
```

The CHR$(0) was obtained using the editing technique we have been examining, through FCTN 0 (zero) - see page (v), V1.2 TI-LINES - where the unwanted characters were deleted to leave CHR$(0) and then the text was added following that character. Alternatively, FCTN ; could have been used - it too provides CHR$(0) on editing.

In the next issue I will follow up CHR$(201) - the LINE NUMBER token for anyone who is interested.

B O O K   R E V I E W

N.E.C. 30 HOUR BASIC by Clive Prigmore 256 pages £5.95 National Extension College

(First published in Tidings V3.2 May 1983)

The National Extension College (NEC) has produced a course in BASIC for around £5.95 claiming to teach you BASIC with or without access to a micro, in 30 hours. The book is written by CLIVE PRIGMORE, and the copy I have is the fourth printing for 1982. There are at least two versions available, so make sure that you get the right one; the other is written specifically for the ZX81.

This version is intended for all others (except the ZX80, probably because that micro doesn't have 'floating point' maths and other goodies, although there are probably still primitive micros around which suffer equally from a lack of facilities), but as it uses primarily the BBC Micro and is intended for use with the late BBC Computer Programme (note spelling), you might guess that it is not ideal for the 99/4A.

First impressions are quite favourable. It is a hefty book, well-designed with an over-large plastic ring binding which enables you to open the thing and lay it flat by your micro. It is a little larger in area than TI-LINES, and about 1cm thick. It contains 256 pages (what a coincidence) of very small type, some of which has been highlighted in a penetrating Kermit green.

The blurb on the back cover as usual is filled with half-truths and downright cobblers (my own book is no exception). For example: 'Microcomputers are the tool of the 80s. BASIC is the language that all of them use. So the sooner you learn BASIC, the sooner you will understand the microcomputer revolution.' ALL of them ? The Jupiter ACE, defunct though it appears to be (and not for lack of popularity but lack of additional hardware, especially from third party suppliers - a familiar story), obviously does not exist, then - it doesn't use BASIC, but a much faster language called Forth. And since when does learning a language have a direct bearing on your understanding of a hardware revolution ?

Eventually they do come clean inside, and you learn that the book can't actually teach you all about BASIC - there is Part 2 yet to come - but what it does teach, it teaches well. The approach to arrays is novel (in my experience) treating them as lists initially, although you may think otherwise.

It is a good book if you are used to the self-imposed discipline of isolated study; I would recommend that you use the book in a group otherwise, or better, with someone who has a smattering of any computer language and a simple understanding of the idea of 'programming'. (if you possess a degree you'll find this book a doddle!)

There are differences between the versions of BASIC catered for in the book's text, and the supposed standard BASIC which the TI computers use, so if you are an abject beginner you may have some problems here, notably with POS, VAL, RND, and so on,as they are interpreted differently (POS doesn't figure at all).

The book is divided into eleven sections: a 'How To Use This Course', 9 course units, and an index. The text begins poorly - hands up anyone who knew what Microsoft BASIC was before they started programming. Er...hands up anyone who knows what it is even now ? Still, it does improve, and the highlighting helps in many cases, although the use of the green bits isn't consistent - sometimes on the first use of a word or phrase, sometimes not.

There are lots of short routines to be keyed in and experimented with, and the early part of the book appears to run parallel with my own BEGINNER'S BASIC, so it can't be ALL bad!

I don't think that I could recommend this book for anyone under, say, 16 who didn't really have more than a smattering of Computer Science under their belts, but it is a very good reference book on BASIC and programming, especially on the so-called structured approach to programming. If you have some grey areas which the TI BASIC book hasn't fully reached, well, this isn't exactly Heineken but it'll do.

There are also a few introductory routines and explanations on Sorting and Searching, which to my mind are essential tools for any kind of serious programming - not that I'm suggesting that there is a kind of frivolous programming.

One point I did note which irritated me a little is that the text recognises RET and RETURN as keys for entering information, but CR or ENTER don't figure at all. Presumably the ZX81 version does at least mention NEWLINE.

A final hint: if you do buy the book, don't flick through it, work through it in a methodical manner, or you'll be put off by the apparent complexity towards the end. I have a feeling that trying to absorb a little of everything at a glance is precisely the thing which puts many budding programmers off; it's like flicking through a detailed car maintenance manual and expecting to understand the operation of a carburettor system as a result.

- -

Editor's note: I would like to run at least one review of a book or books every two months or so. If any member would like to put pen to paper and give their considered opinion of any books which they have, TI-related or not, I would be happy to receive them.

- -

C L O S E   F I L E

The other evening as I was about to leave work I received a call from Wendie Pearson, Listings Editor at Personal Computer News - one of the weekly magazines - asking for details of Oxon TI Users. I found that she knew nothing of the TI-99/4A or its peripherals - she thought that the Speech Synthesiser was a cassette tape - and I hope that I provided her with the correct information and that she made a note of it. (She had assumed that as I catered for the blind/partially-sighted user, they would be able to buy the Terminal Emulator II/Speech Synthesiser combination from me; it's a nice idea but unworkable at present).

The most interesting point came when she asked me if I had any further information on the company who had started making 99/4As again. I know of one machine which is compatible with TI BASIC - the CORTEX, which is an electronics construction project published by Electronics Today International - and I did hear rumours, unsubstantiated, that TI were going to re-enter the market later in 1984 with an up-graded, up-market bug-free version of the 99/4A, but this is the first that I have heard of another company producing them. Watch this space.

The Speech Recognition system which I mentioned last issue can be ordered from HOWARD GREENBERG of ARCADE HARDWARE - tel: 061 225 2248 - I don't know what the price will be or the availability, but I know Howard and he will be honest with you about any likely delays. He also markets a wide range of 'firmware' - modules - and hardware, and he considers himself a User first and a businessman second, which may be bad for business but it's good for customer satisfaction.

One final piece of news: after July 1984 PARCO ELECTRICS will be marketing a UK-made Extended BASIC module. Price not known at the time of publication.

Enjoy your programming, Pete Brooks