INDEPENDENT NATIONAL USER GROUP

FOR THE BBC MICROCOMPUTER

BEEBUG NEWSLETTER

VOLUME 1 NUMBER 6 OCT 1982

CONTENTS

MEMBERSHIP NOW EXCEEDS 10,000

BRITAIN'S LARGEST SINGLE-MICRO USER GROUP

EDITORIAL

Last month we reported that we were unhappy about Acorn's intention to charge around £15 for replacing the 0.1 operating system. We also published a sample of letters received on the subject, and said that we were taking legal advice on the situation. The new trade magazine 'Microscope' appears to have taken this to mean that we were actively planning a test case. On the front page of their 23rd Sept issue they wrote that "the Independent BEEBUG User group..... plans to initiate a test case in the small claims court against Acorn". They also quoted Chris Curry's reply - he "plans to fight the move, if it comes to court". We have written to Microscope correcting the several errors in their reporting, and we hope that they will publish our reply.

We have now received a response from our legal advisors, and on the basis of this we are re-opening our discussions with Acorn. We will let you know the outcome in the next issue.

.....

We are pleased to announce that BEEBUG's membership has now reached comfortably above the 10,000 mark, and we think that this makes us the largest computer user group in the UK.

This month we have included an extended section of Acorn News to include two releases that Acorn have kindly allowed us to publish. These are concerned with the enhancements to be available on the new operating system; and the bugs that have been found in the Basic ROM (Basic issue 1). There is a separate article on the Tube and the second processor options.

We also begin an occasional series "BBC Basics" for the less advanced user. This month a number of features of BBC BASIC are introduced by way of the development of a Reaction Timer program; though it is intended that the format of articles will be varied in future issues, and will not be limited to explorations of BASIC. We would also draw attention to the Debugging series, which in part aims to clear up some of the things that go wrong when programs are typed from listings.

.....

BEEBUG's software library aims to provide reasonably low priced programs to members, and at the same time aims to encourage writers of the best software with lump sums or royalty payments. One other way in which the library can be of use is in maintaining and improving present programs, and passing on these improvements in the magazine. This month, in "POINTS ARISING" we give a modification to STARFIRE to make it run on operating systems 1.0 and greater. In the "NEGATIVE INKEY" article in this issue, we give improvements to both MOONLANDER and STARFIRE to allow them to use the negative INKEY function, so improving key response. Next issue we will be publishing a routine to allow STARFIRE to run with joysticks.

You may have noticed that there have been no new titles in the library for a month or two - but we are adding four more next month, and details of these will be published in the November issue. There are three new games cassettes, including the best Asteroid-type arcade game that we have come across. It is in machine code and requires 32k. The other program is a comprehensive cassette-based file management program that will allow you to keep data bases of pretty well unlimited variety, and to sort them on any field that you wish. Finally we would draw your attention to our free software offer (see enclosure) - Free software with every 2 new memberships that you can drum up.

Thanks also to all those who made it to see us at the PCW show. We hope to be at the "Computer Fair" in April.

.....

Please note that to give us a Christmas holiday there will be no issue in January, rather than December as previously indicated.

BBC BASICS

In this, the first of a series of articles for the less advanced user, David Graham looks at the development of a simple reaction timer, illustrating the use of procedures, integer variables, long variable names, INKEY, GET and the REPEAT UNTIL structure. A full listing of the program under development is given at the end of the article, and the reader may find it helpful to refer to this as the program is developed.

The logic involved in a primitive reaction tester is pretty straightforward, and the problem of creating a program to perform this function provides an ideal testbed for introducing a scattering of concepts from BBC Basic. In essence the program to be discussed will generate a prompt on the screen, and then start a timer which will run until the user presses a key. The reaction time will then be calculated and printed to the screen.
To begin with, enter the following lines

```
10 REM REACTION TESTER
20 MODE 7
```

Line 20 sets mode 7, though any mode would be acceptable. The advantage of mode 7 is that it uses very little memory (1k only) leaving plenty of room for programs and data. The reaction tester is not a long program, but it is worth gaining familiarity with this mode.

The next task of the program is to wait for a few seconds, and then flash the prompt on the screen. Try adding

```
80 TIME=0:REPEAT UNTIL TIME>=500
100 PRINT TAB(18,13)"TEST"
```

I have missed lines 30 to 70, and line 90, because they will be used in later development. Taking line 100 first,it just tabs along 18 spaces, and down 13, so as to print the word "TEST" roughly in the centre of the screen - since in mode 7 there are 40 horizontal character positions by 25 lines. Line 80 of the program uses two features of BBC Basic to effect a time delay. First it sets the internal user clock to zero, then it uses the REPEAT UNTIL structure to wait until the clock reads 500 units or greater. Since the clock counts in one hundredths of a second, this gives a 5 second time delay. If you are familiar with other Basics you may have expected to see 80 FOR A=1 TO 10000:NEXT A. This uses a FOR...NEXT loop to count time. One of the disadvantages of this is that the time delay caused by such a loop will differ in different versions of the BBC machine.

In the reaction timer program we will be using the time delay loop for other things as well, and we can make it available to other parts of the as yet unwritten program by putting it into a procedure. Procedures are one of the bonuses of BBC Basic that make it easy to introduce a degree of structuring into a program. The new user guide pp 230 and 329 gives details on this, but essentially you can place any frequently used routine at the end of a program between the words DEF PROCx and ENDPROC, where x is the name that you call your procedure. Then every time you put PROCx in the main body of the program,there will be a jump to the procedure, and then a jump back once the procedure has been executed, just as with a subroutine in ordinary Basic. One advantage of procedures over subroutines however is that you do not need to specify line numbers in the procedure call.

We will put a procedure call into the program at line 80 to give a 5 second time delay. We will call it PROCwait. To do this, enter the following lines.

```
80 PROCwait
1000 DEF PROCwait
1010 TIME=0
1020 REPEAT UNTIL TIME>=500
1030 ENDPROC
```

Now, every time the program encounters the word PROCwait, it will jump to line 1000

and execute the wait routine, and when it hits the ENDPROC at line 1030 it will automatically jump back to the command following PROCwait in the main body of the program.

In this particular case we did not really need to use a procedure to achieve our ends, but the point is to illustrate the use of the procedure generally. Procedures can help clarify and neaten program structure, and you will not usually go far wrong if you try to keep the main body of your program as short as possible, and to call a whole series of procedures defined below the main part of the program - see article on structuring in the June issue of BEEBUG for further details.

Line 100 of the program puts the word "TEST" on the screen. This is the cue for the user to hit the space bar as quickly as possible. And at this point in the program we need to set the timer going again, and stop it when the space bar is pressed. Lines 110-140 achieve this, and also print out the reaction time:

```
110 TIME=0
120 A$=GET$
130 testtime=TIME/100
140 PRINT TAB(8,16)testtime;SPC(2)"SECONDS"
```

Line 110 resets the timer. Line 120 uses the GET$ function, this causes the program to pause until a key is pressed, and then assigns whatever is entered to the variable A$ - in this case the contents of A$ becomes a space if the space bar is pressed. We will not actually be making use of the variable A$, but it must be there to satisfy the syntax. We are just using GET$ to hold up the program until a key is pressed. Fortunately, although the Basic program pauses here, the internal timer is not stopped. Line 130 introduces a variable 'testtime', and sets it equal to the new reading of the TIME divided by 100 (to convert 100ths of a second back to seconds) - note here the use of long variable names to make the program easier to read. Line 140 tabs along to a clear space, and then prints the contents of 'testtime' followed by 2 spaces, followed by the word "SECONDS".

The program should work in the form so far developed, but we can introduce some minor sophistications. Firstly, if you change line 120 to read `120 IF GET$<>" " THEN 120` this will cause the program only to react to the space bar, whereas previously any key would get a response. Secondly, as the program stands, it will not detect if the space bar is left pressed down, so giving a very fast reaction time. The way to avoid this is to check to see whether the space bar was pressed in the time immediately before the prompt "TEST" is printed. Inserting the following line will achieve this:

```
90 IF INKEY$(0)=" " THEN PRINT"CHEAT":VDU 7:GOTO 80.
```

The INKEY$ function is just like GET$, except that you can specify how long the computer will wait before moving on. The number in brackets after INKEY$ gives this time in hundredths of a second. When it is set to zero, the keys are checked without waiting at all. The rest of the line executes a 'beep' (VDU7) and prints the word "CHEAT" before returning to line 80 if the space bar was pressed during the waiting period.

As you can guess, INKEY$(0) is ideal for testing for keys in a real time game, where, for example the Z and X keys might be used for left and right movement as in "Snapper" and "Magic Eel" (See the "Negative INKEY" article in this issue for developments of this technique).

Input Buffer

If you now try the reaction tester program, and test the cheat detector, it should perform as required, but if you have held the space bar down for any length of time, it will keep registering a cheat condition, even though you have stopped cheating. This illustrates the action of something called the 'INPUT BUFFER'. Whenever you press keys on the keyboard, even during the running of a program (when nothing appears on the screen) the keys pressed are detected by the operating system and stored in an area of memory called the 'input buffer'. The next time a GET,

INKEY or INPUT is used the data is really read from the input buffer not from the keyboard. As a test put in a the following short program:

```
10 PRINT"You have 5 secs to type something"
20 TIME=0:REPEAT UNTIL TIME>500
30 INPUT"You have just typed",message$
```

If you obey the instructions you should see that the data has got into the variable message$ even though it was typed before the program had reached the INPUT statement in line 30, and before the input prompt had appeared on the screen.

This is of relevance in the 'reaction tester', because the reason why it repeatedly registered "CHEAT" was that by pressing the space bar for a length of time you had put a series of 'space' characters into the buffer. Each time INKEY$(0) was executed it pulled off just one of these space characters from the input buffer string. The way round this is to 'flush' the input buffer of all its characters before the cheat test restarts. This can be done with one of the machine operating system *FX calls. Insert 70 *FX 15,1 (Note that *FX calls should be placed singly at the end of a Basic line - this one is on its own, so there is no problem). You need also to change line 90 to read GOTO 70 at the end, rather than GOTO 80.

Yes/No Procedure

As the program stands, you must type 'RUN' each time you want to repeat a test. This can be streamlined by introducing a "Do you want another go?" question, and checking the keys with GET$ for a 'Y' or 'N' answer. This sort of thing is best handled in a procedure or function (because you will almost certainly want to use it again elsewhere in this program or future ones, and you don't want to have to rethink the process). Inserting the following lines achieves this using a procedure.

```
200 PRINT TAB(10,22)"Another go? (Y/N)"
210 PROCyesno
220 IF answer$="Y" THEN RUN
230 END

2000 DEFPROCyesno
2010 answer$=GET$
2020 IF answer$<>"Y" AND answer$<>"N" THEN 2010
2030 ENDPROC
```

The actual procedure resides at lines 2000-2030. It uses GET$ to wait until a key is pressed, and sets the variable 'answer$' to that key. (Note that strictly speaking GET$ is an unnecessary feature in Basic because line 220 could be replaced by:

```
220 answer$=INKEY$(0):IF answer$="" THEN 220
```

However, there are many features in Basic that are unnecessary but are included to make programming a little less tedious).

If the key was neither a 'Y' nor a 'N', it goes back and waits for another key press. This is achieved in 2020. If either 'Y' or 'N' was pressed, the program continues, and ENDPROC takes it back to the procedure call point in the program-ie line 210. Here, if the answer was 'Y' the program is re-run, if not it ends. The same procedure can of course be called an unlimited number of times during such a program, and it can be used for yes/no answers to different questions. Of course the advantage of using GET$ over the INPUT statement is that you don't need to press 'return' after each entry.

BEEBUG has started a procedure/function library to gather useful procedures and functions, and the index to this was given in the Sept '82 issue p28, where incidentally, there is a more sophisticated yes/no function.

The last improvement that I want to introduce to this program this month involves allowing a run of five reaction tests, and a calculation of the average reaction time. This is achieved by introducing a REPEAT UNTIL loop at line 40. The additions are as follows:

```
  30 number%=0:total=0
  40 REPEAT
  60 number%=number%+1
 150 PROCwait
 160 total=total+testtime
 170 UNTIL number%=5
 180 average=total/number%
 190 PRINT TAB(10,20)"Average = ";average;" seconds"
```

Line 30 sets two variables 'number%' and 'total' to zero. The % after a number means that this is an 'integer' variable, it cannot take decimal values, ie it can be set to 0, 1, 2, -4, 100 etc but never 1.5. BBC Basic handles integer vriables much more quickly than the other sort of numeric variables (floating point), and they also take up less space in memory (though this is of insignificant importance unless they are integer arrays), so it is worth getting into the habit of using them where possible. The REPEAT loop, which begins at line 40 repeats the main body of the reaction test until it reaches the UNTIL statement at line 170 - ie until number%=5. Each time the repeat loop is followed, line 60 increments number% by 1, giving 5 tests before the loop is terminated.

Line 160 just keeps a running total of the reaction times taken, and when the set of 5 tests is completed, line 180 calculates the average by dividing the total by 5 (since number% will equal 5 by then), and line 190 prints the average so calculated.

There are of course many further sophistications that may be added to this program - such as sound and colour displays, random delays before the prompt is flashed up - though for the moment at least, this will be left for you to experiment with.

```
  10 REM REACTION TIMER
  20 MODE7
  30 number%=0
  40 REPEAT
  50 CLS:PRINTTAB(13,6)"REACTION TES
TER"
  60 number%=number%+1
  70 *FX15,1
  80 PROCwait
  90 IFINKEY$(0)=" " THEN PRINT"CHEA
T":VDU7:GOTO70
 100 PRINTTAB(18,13)"TEST"
 110 TIME=0
 120 IF GET$<>" " THEN 120
 130 testtime=TIME/100
 140 PRINT TAB(8,16)testtime;SPC(2)"
SECONDS"
 150 PROCwait
 160 total=total+testtime
 170 UNTIL number%=5
 180 average=total/number%
 190  PRINT TAB(10,20)"AVERAGE =  ";
average;"   SECONDS"
 200 PRINTTAB(10,22)"ANOTHER GO ? (Y
/N)"
 210 PROCyesno
 220 IF answer$ = "Y" THEN RUN
 230 END
 240 ENDPROC
1000 DEFPROCwait
1010 TIME=0
1020 REPEAT UNTIL TIME >=500
1030 ENDPROC
2000 DEFPROCyesno
2010 answer$=GET$
2020 IFanswer$<>"Y"ANDanswer$<>"N" T
HEN 2010
2030 ENDPROC
```

D.E.G.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

Henrietta Holocaust has sent the following tip:
Black on White Those readers whose TVs are clearer in reverse video (black on white) may be interested to know that removing (or switching) link S26 (by the video processor IC6, which is the thing with a heat sink on it) provides a permanent hardware fix.

Block move In a machine code monitor 'block move' is a very useful facility. This feature can be achieved by using the *SAVE followed by the *LOAD commands. Full details of the syntax are to be found in the user guide.

SCREEN TO CASSETTE (16k/32k)

Many people have asked us to provide a routine to dump the screen to cassette. So here we provide a PROCedure to do just that. It uses the *SAVE command to save the screen byte by byte, and the tape is reloaded with *LOAD""

Place the procedure in a suitable place in your program. However we don't put a message on the screen telling you to press "record and return" otherwise this would corrupt your beautiful screen display; instead you should have your cassette player ready with 'record & play' depressed, (Motor control would be an advantage here). When you hear a 'beep' from the speaker you have to press the 'space bar' to start the recording, when you have done so the start of the recording process will be announced by a different 'beep'. When the recording has finished you will hear yet another 'beep'.

It doesn't take long to save the "teletext" screen, but it takes nearly 6 minutes to save a MODE 0 screen.

We have provided a sample program to put stuff on the screen, call the procedure to save it, and then display it again. The command to display it is simply *LOAD"SCREEN" in any mode. As it loads you will see the picture being built up a line at a time.

We must disable the cassette file messages, this is done with the *OPT command, a full description can be found on page 398 of the "user guide". But to summarise *OPT 1,0 turns off the cassette file messages, and *OPT 1,1 turns them on again. The PROCedure starts at line 1000. The values after the *SAVE"SCREEN" xxxx yyyy are the start and finish addresses for the screen in memory. We gave these addresses in Issue 2 page 5. Use the same addresses whether you have a 16 or a 32k machine.

```
  5 INPUT"Mode",mode%
 10 MODE mode%
 15 IF mode%<>7 THEN 200
 20 FOR point%=1 TO 200
 30 x%=RND(39):y%=RND(24)
 40 c$=CHR$(RND(26)+64)
 50 PRINTTAB(x%,y%);c$;
 60 NEXT point%
 70 GOTO 300
200 FOR point%=1 TO 200
210 x%=RND(1279):y%=RND(1023)
220 DRAW x%,y%
230 NEXT point%
300 PROCscreendump(mode%)
320 CLS
330 PRINT"READY TO LOAD SCREEN FROM
CASSETTE"
335 PRINT"REWIND CASSETTE AND PRESS
PLAY"
340 *LOAD"SCREEN"
350 *OPT 1,1
999 END
1000 DEF PROCscreendump(mode%)
1005 *OPT 1,0
1006 SOUND 1,-15,100,10
1007 REPEAT UNTIL GET$=" "
1008 SOUND 1,-15,200,10
1010 ON mode%+1 GOTO 1020,1020,1020
,1030,1040,1040,1050,1060
1020 *SAVE"SCREEN" 3000 7FFF
1025 GOTO 1070
1030 *SAVE"SCREEN" 4000 7E7F
1035 GOTO 1070
1040 *SAVE"SCREEN" 5800 7FFF
1045 GOTO 1070
1050 *SAVE"SCREEN" 6000 7F3F
1055 GOTO 1070
1060 *SAVE"SCREEN" 7C00 7FE8
1070 SOUND 1,-15,50,10
1080 ENDPROC
```

S.W.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SENDING PROGRAMS VIA THE TELEPHONE

A visitor to the BEEBUG stand at the PCW show reported good results with sending 300 baud recordings via the telephone. If you want to try this just make a recording of the program at 300 baud using *TAPE 3 (Don't forget to reset this afterwards with *TAPE). Position the telephone microphone near the recorder speaker and avoid touching anything while the tape is playing. At the receiving end, position the telephone earpiece near the recorder microphone. Time the tape first so that both ends know how long to wait.

Program tested on 0·1 and 1·1 O.S.

MEMORY DISPLAY UTILITY (16k)

by Chris Bingley

It is useful for a multitude of reasons to be able to look at the contents of the machine's memory. The program listed below not only reads the contents of blocks of memory, but displays these in four different forms: namely decimal, hexadecimal, binary and ASCII. Each form has its uses, and to show the use of the ASCII form, try inspecting memory from &8000 for example.

To do this, run the program, and it will prompt for two pieces of data: a start address, and the number of locations to be inspected. Both values may be entered either in decimal or in hex (preceding the hex value with an ampersand '&'). If you enter &8000 for the start address, and say, 1000 for the number of locations to be inspected, you will see the following printed vertically in the ASCII column of the display:'BASIC (C) 1981 Acorn', or at least you do with operating system 0.1. Incidentally the program uses paged mode, so you don't have to take it all in at scrolling rate! The shift key activates the scroll.

If you inspect locations 32877 decimal (&806D hex) you will find the list of the beeb's reserved words(this again applies to OS 0.1). In the cases that we have checked, the byte following the reserved word is the token used to signify that word by the Basic interpreter; and what we are looking at is a so-called 'look up table' for the interpreter.

If you Escape from the program, and re-Run it so as to inspect addresses from 3584 decimal onwards (3584 is the value of PAGE - see BEEBUG number 2 page 15), you can look at the way in which the beeb has stored the actual program that is running in the machine. Locations 3585 and 3586 contain the values 0 and 10 (dec), signifying that 10 is the number of the first line in Basic. Address 3587 contains 244 decimal (F4 hex). This is the token for 'REM' (see new User Guide page 484). There next appears **STORE DUMP** listed in literal ASCII text - again this is listed vertically at one character per memory location. In the following location is the value 13. This is the code for Carriage Return, and signifies the end of the Basic line. Addresses 3604 and 3605 contain the values 0 and 20, indicating that the next line in Basic is numbered 20, and so on.

```
  10 REM**STORE DUMP**
  20 MODE 6: PROCaddr: CLS
  30 DIM bit%(8)
  40 PRINT"  Addr  Addr Val Val Val
    $"
  50 PRINT"  Dec   Hex  Dec Hex Bina
ry"
  60 VDU 28,0,24,39,3: REM create te
xt window for data
  70 VDU 14: REM enter page mode
  80 FOR i%=first% TO last%
  90 IF i%>65535 THEN PRINT"Address
out of range": GOTO 190
 100 @%=6: REM set field width to 6
 110 PRINT i%, ~i%;: REM print addre
ss in dec and hex
 120 con%= ?i%: REM memory contents
 130 @%=4: REM set field width to 4
 140 PRINT con%, ~con%;" ";: REM pri
nt contents
 150 PROCbits
 160 IF con%>32 AND con%<127 THEN PR
INT" "; CHR$con%;: REM print characte
r if valid
 170 PRINT
 180 NEXT i%: PRINT
 190 VDU 15: REM leave page mode
 200 INPUT"More(Y or N)",ans$
 210 IF ans$="Y" THEN PROCaddr: CLS:
GOTO 70
 220 VDU 26: REM restore default win
dow
 230 @%=2570: REM restore default fi
eld
 240 CLS
 250 END
 260 DEF PROCaddr
 270 PRINT"Enter start address and n
umber of"
 280 PRINT"bytes, in hex(with prefix
 '&') or"
 290 PRINT"decimal.  Use 'shift' key
 to scroll"
 300 INPUT"when screen is full."',fi
rst$,bytes$
 310 first%= EVALfirst$
 320 last%= first% +EVALbytes$ -1
 330 ENDPROC
 340 REM
 350 DEF PROCbits
 360 LOCAL i%,byte%
 370 byte%= con%
```

```
380 FOR i%=1 TO 8
390 bit%(i%)= byte% MOD 2
400 byte%= byte% DIV 2
410 NEXT i%
420 FOR i%=8 TO 1 STEP -1
430 PRINT CHR$(bit%(i%)+48);
440 NEXT i%
450 ENDPROC
```

If you wish to examine the actual machine code programs resident inside the Beeb's ROMs and EPROMs then you need what is called a disassembler. This is a program which displays memory contents rather like the one listed here, but which translates the data it finds into the machine code mnemonics of the 6502 microprocessor, thus translating the code from machine readable form into something more appropriate to human beings. This is not just a simple matter of translating codes because commands in 6502 code vary in length, and use a variety of what are called 'addressing modes'. If you require a disassembler, we can recommend the one supplied on the BEEBUGSOFT Utilities 1 cassette.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

Program tested on 0·1 and 1·1 O.S.

BASIC....FASTER BASIC....MACHINE CODE

The string sort function published in the July Newsletter page 10 is interesting but rather slow in operation. Four sets of capital letters in keyboard order took 37 seconds to sort. Use of the "?" operator to scan the input string and set up the output is faster and improves the speed to 11 seconds.

```
10 CLS
20 DIM Z% 255, R% 255
30 REPEAT
40 INPUT LINE'"String to sort"'$Z%
50 Y%=0 :TIME=0
60 FOR L%=32 TO 126
70 FOR X%=0 TO LEN($Z%)
80 IF Z%?X%=L% R%?Y%=L% :Y%=Y%+1
90 NEXT X%,L%
100 R%?Y%=13
110 PRINT$R%'"Time = ";TIME/100;" sec"
120 UNTIL FALSE
```

From this it was not difficult to write a machine code program, which does the job in less than 0.2 seconds!

```
10 DIM Z% 255, R% 255, CODE 100
20 CLS :MAX=&7B :PROCassemble
30 REPEAT
40 INPUT LINE'"String to sort"'$Z%
50 TIME=0
60 CALL start :PRINT''$R%
70 PRINT'"Time = ";TIME/100;" sec"
80 UNTIL FALSE
90 DEF PROCassemble
100 FOR PASS=0 TO 3 STEP 3 :P%=CODE
110 [OPT PASS
120 .start LDX #&FF :LDY #0
130        LDA #&20 :STA &70
140 .c     INX      :LDA Z%,X
150        CMP #&0D :BEQ d
160        CMP &70  :BNE c
170        STA R%,Y :INY :JMP c
180 .d     INC &70  :LDA &70
190        CMP #MAX :BEQ e
200        LDX #&FF :JMP c
210 .e     LDA #&0D :STA R%,Y
220        RTS
230 ]
240 NEXT PASS
250 ENDPROC
```

This program shows how a fast assembler program can be written after first establishing the approach or 'algorithm' in BASIC using the ?,! and $ operators. This is a facility which has been available to FORTH programmers for a long time but is only now becoming available to the BASIC programmer.

The above program is fine if you want to sort a sequence of letters into order, but the more normal problem is a sequence of words in a string array. Has anybody tackled this in assembler? Cedric T Marshall

ACORN NEWS

New ROMs for Old

New plot routines in the new operating system ROM, and a new version of BASIC.

OS 1.2 ROM

Much has been made of the fact that there is shortly to be a new operating system ROM for the BBC Micro. We reported last month that this would be system 1.1 or greater. We have now learned that it is 1.2 that is being ROMmed, and that the device should be available by December. As well as correcting the major bugs that we have reported in earlier issues, the new operating system will incorporate a number of useful enhancements. The new FX calls included are covered in the manual, but the new ROM also includes some extremely useful graphics routines that are not described or defined in the currently available manual.

The new manual states that a number of codes in the PLOT command have been reserved for future expansion. Some of these have now been allocated. Generally speaking they are 'fill' routines of one kind or another.

Here we give (with permission from Acorn) details of these. We have been trying them on our 1.1 OS, and they are quite impressive. A demonstration program is given at the end of the fill routines.

FILL ROUTINES

Although the BBC Computer can fill triangles with great simplicity, there are many shapes which are tricky to fill using just triangles, (especially if invert or EOR action are used) but very easy to draw the outline. This is usually because the shape is only known in terms of lines and arcs, and not in terms of triangles radiating from a point. So assume that the outline has been drawn (somehow) on the screen. How can it be filled in? The new Machine Operating Systems for the BBC Computer have some primitives with which it is simple to construct non-recursive convex polygon and recursive pierced and concave polygon fill algorithms. The primitives are as follows:

1. New PLOT instructions &48 to &4F (72-79).
   The action of these instructions is to start from where specified by X,Y (&48-&4B are relative, &4C &4F are absolute); and search along the X-axis left and right while the pixels are in the current background colour. Then the two points X1,Y and X2,Y are set as the last two points visited and a line is drawn between them (&48,&4C it isn't; &49,&4D in foregound colour and action; &4A,&4E inverted; &4B,&4F in background colour and action).
2. New PLOT instructions &58 to &5F (88-95).
   The action of these instructions is to start from where specified by X,Y (&58-&5B are relative, &5C &5F are absolute); and search along the X-axis right only until the pixel is the current background colour, this new point X1,Y is set as the most recent point visited, and X,Y is set as the previous one visited. A line is drawn between them (&58,&5C it isn't; &59,&5D in foregound colour and action; &5A,&5E inverted; &5B,&5F in background colour and action).
3. New OSWORD call &0D
   This call returns the preceding two positions of the graphics cursor, the most recent position X,Y and the other X',Y' are returned as 16-bit integers converted back to user coordinates by reversing the scaling and subtracting the current origin position (VDU 29). The order is:

0	X' low	1	X' high	2	Y' low	3	Y' high
5	X low	5	X high	6	Y low	7	Y high

Example of a program to fill a shape

A diamond shape is plotted during the lines 20-40, then they are filled during lines 90-110 (the STEP 4 is because the vertical resolution is every 4 lines).

```
10 MODE 4
20 MOVE 500,100:DRAW 900,500
30 DRAW 500,900:DRAW 100,500
40 DRAW 500,100
90 FOR Y%=100 TO 900 STEP 4
100   PLOT &4D,500,Y%
110 NEXT
```

BASIC II

Acorn are also ROMming 'Issue II' Basic. This new version, which again should be available by December, removes a number of minor bugs in Basic I - the majority of which most people are totally unaware. We reproduce below (with permission from Acorn) a listing of these bugs. It may prove useful to be aware of these, and interesting to explore some of them. We should perhaps add that in our view these bugs are not of the same order as those in operating system Ø.1.

Issue II BASIC

1. ELSE no longer leaves a byte on the hardware stack in ON....GOTO/GOSUB
2. INSTR no longer leaves the shorter string of INSTR("AB","ABC") on the software stack.
3. The lexical analyser is now called correctly from EVAL, so that EVAL("TIME") now works.
4. ABS can take the absolute value of integers without bit 31 set, without returning a string (i.e. PRINT -ABS1 works).
5. ASC":" can be used in the assembler without confusing it.
6. A new statement OSCLI has been introduced, shortest abbreviation OS., token value &FF. It takes a string expression and gives it to the operating system. E.g. OSCLI "KEY "+STR$Z+"This is key "+STR$Z. It has no unique errors of its own, just the normal "Type mismatch" error.
7. The value of the token for OPENIN has changed to &8E and now just opens files for input only, using the &40 open. A new keyword OPENUP (open for update) has been introduced with the old token value &AD which opens files for update using the &C0 open. This results in old programs automatically changing to OPENUP when they are LOADed into the new interpreter.
8. The next bit in OPT (OPTs 4 to 7) controls whether code is generated by the assembler will be put at the program counter P% or at the code origin O%. If it is set, code will be put at O%, and both O% and P% incremented, if unset only P% will be used. Current implementation results in all OPTs greater than 3 causing code to be put at O%.
9. The LN and LOG functions are completely recoded to make them more accurate, and avoid the LN(2E-39) bug.
10. The fix routine avoids the INT1E38 bug.
11. The SIN/COS functions are completely recoded to make them more accurate.
12. The binary to decimal string conversion routines, used for PRINT and STR$, have been changed to allow the use of 10 figures of precision on printing. The initial value of @% is now &0000090A to give the same results on startup. Note that @%=10 now gives the internal default of 10 figures. The changes allow the maximum positive integer 2147483647 to be printed (and indeed get 2^33 right). STR$ when not controlled by @% uses the new 10 figure default, which will result in it giving different answers to before. E.g. 7.7 (a recurring binary fraction) will be converted to 7.699999999.
13. Four new operations are available in the assembler. These are EQUB, EQUW, EQUD, EQUS. They have no errors of their own apart from the "Type mismatch" error. They take a single argument and put its value into the assembly code. EQUS puts all the characters of a string into the code without a carriage return like $P%, if you need a carriage return just add it to the string. EQUS may be used together with FN to implement macros as shown in this example:-

```
DEF FNOSBYTE(A,X,Y)
IF A>127 [OPT Z:LDY#Y:]
[OPT Z:LDX#X:LDA#A:JSR&FFF4:]
=""

 .....
[OPT Z
EQUS FNOSBYTE(&99,33,44)
EQUS FNOSBYTE(2,1,0)
]
```

14. The new startup REPORT is the way to tell the new version of BASIC from the old one. The new one has (C)1982
15. The bug associated with ON ERROR GOTO 9999 has been removed.
16. A MODE change now resets COUNT.
17. BASIC will now only execute if A contains 1 on entry at 8000. There is no entry at 8003 any more.
18. BASIC is of type 60; this means that it contains its own tube relocation address after all the standard sideways ROM items. Of course, with the general release this will be 00008000 (the default for no relocation), but this gives more trouble free production of relocated versions to run at 0000B000.
19. The version number is 01
20. The INPUT ';' is introduced, and functions as the ',' does.
21. Fatal errors have been introduced. Errors whose ERR is zero, cause an ON ERROR OFF effect while they are being processed.
22. STOP has been redefined as a fatal error, this causes the "STOP at line 0" message to be corrected to" STOP".
23. The "No room" error is a fatal error.
24. The standard error handling procedures do not use stack space any more. The situation of running out of all free space no longer causes error messages to be printed out followed by a "No room" message.
25. The allocation of space for strings has been made slightly more efficient, it can cope with REPEAT A$=A$+"*":UNTIL LENA$=255 and only allocate 255 bytes. It still has problems if B$ is being done alternately with A$.
26. A new error 45 "Missing #" arrives if PTR, EOF, BGET, BPUT, EXT have a # missing.
27. DIM P% -2 gives a "Bad DIM".

SUNDRY HARDWARE NOTES -

Joysticks Beeb look-alike joysticks are now available from Microage, rumour has it that these are the same stock that Acorn will be selling, it is just that they got their shipment faster.

Teletext The Teletext add-on unit has had a few teething problems, but it is well on the way to completion - well in advance of the Prestel unit. We saw one of the Teletext acquisition units working at Acorn last week, and it managed to download programs from teletext pages without any apparent difficulty. The only problem in our view is the program length per page. This is limited to the equivalent of a screenful of teletext - or 1k. Now it is not worth the trouble of setting up a downloader just for the sake of programs of this length, and the probable solution will be to use a batch of pages late at night to send complete programs of useful length. The other problem, and this has implications for the long-term viability of the project, is how are the programs to be financed? If you are not going to use the downloading facility, then you might as well buy a TV with a teletext adaptor as it works out a good bit cheaper. (Rumours indicate that the Beeb teletext acquisition unit may cost around £200).

Speech synthesis The speech synthesis unit is coming along slowly - it is at present working through a ROM emulator with a vocabulary of around 200 words. The words are quite clear, and can be typed in from the keyboard more or less as spoken. There is at present no means of altering intonation, so that words like "The" are often overstressed.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

Get into MODE 7 before loading long programs, or use a header.

NEGATIVE INKEY

(How to improve key detection)

Here Rob Pickering draws attention to an extremely useful facility of the INKEY function which allows fast multiple key detection - particularly useful for games, and also allows the detection of the 'shift' and 'control' keys.

INKEY(n) is normally used to read a single character from the keyboard. INKEY(n) will give the ASCII code of a character pressed, and INKEY$(n) will give the character itself in string form. As was explained in the PUG, the number 'n' in brackets can be a constant or variable of positive value, used to indicate a maximum wait period. The command would wait until either a key has been pressed or until the end of the maximum wait period.

The full manual gives further details on using INKEY(), by stating that if 'n' is made negative INKEY behaves in a very different way and can be used to detect when a particular key is depressed. Each key on the keyboard has a unique negative value associated with it, which is unrelated to the ASCII value. This unique value may be used to detect when that key is held down.

For instance, the negative value representing either of the shift keys is -1, try the following:

```
10 IF INKEY(-1)=TRUE THEN PRINT"SHIFT-KEY PRESSED..."
20 IF INKEY(-1)=FALSE THEN PRINT"SHIFT-KEY RELEASED..."
30 GOTO 10
```

The above will show simply whether or not the SHIFT key is being pressed. As you can see from the above, the INKEY(-n) will evaluate as TRUE (which is -1) when the key in question is pressed, and FALSE (which is 0) when not pressed. A simpler way to express line 10 would be 10 IF INKEY(-1) THEN PRINT.... because of the way the expression itself evaluates down to TRUE or FALSE anyway. (You may always do this when testing only for values of TRUE or FALSE).

USES AND ADVANTAGES

(1) Testing the keyboard by using INKEY(-n) is independent of the keyboard buffer, so that when testing for a particular key it is not necessary to keep clearing the input buffer. (See "BBC Basics" elsewhere in this issue for an explanation of the input buffer).
(2) Keys such as SHIFT and CAPS-LOCK can be detected, whereas they produce no ASCII code and cannot be detected with the normal INKEY function.
(3) Keys held down simultaneously can all be detected, whereas only one code can be produced at any one time for the normal INKEY function.

The latter advantage is perhaps the most generally useful, especially in games. As a demonstration, you may like to try the following three lines:

```
10 IF INKEY(-66)  PRINT"A DEPRESSED"; ELSE PRINT"A RELEASED";
20 IF INKEY(-101) PRINT" B DEPRESSED" ELSE PRINT" B RELEASED"
30 GOTO 10
```

The above program will show when the 'A' and 'B' keys are depressed, either individually or together. I don't think this is otherwise possible.

If you are one of the many people who bought or entered the MOONLANDER program, listed correctly in the April newsletter, then you may like to make a few small modifications to use the negative INKEY function, which I think improve it tremendously. Alter lines 450-470 as follows:

```
450 IF INKEY(-104) THEN YS=YS+0.04:FUEL=FUEL-1
460 IF INKEY(-103) THEN YS=YS-0.04:FUEL=FUEL-1
470 IF INKEY(-68)  THEN K=K-0.01:  FUEL=FUEL-2
```

These few alterations will allow you to generate both vertical AND horizontal thrust at the same time, not possible in the original.
The same can be done for "Starfire" (Software Library) with the following changes (incorporated on the cassette from 20th Sept) though space does not allow a full explanation.

```
Delete lines 420-460
Replace lines  640 PROCkey
               3000 PROCkey
Add lines      6000 DEF PROCkey:LOCAL t%,u%
               6010 RESTORE 6020
               6020 DATA 58,139,42,138,26,136,122,137,33,70
               6025 DATA 114,49,115,50,116,51,0,0
               6030 READ t%,u%:IF t%=0 ENDPROC
               6040 IF INKEY(-t%) A%=u%:ENDPROC
               6050 GOTO 6030
```

Obviously there are numerous applications for this form of the INKEY() function, and we hope that you will find good use for it.

The full list of negative values which correspond to each key is given on page 275 of the full user's guide. If you still don't have this, then you could use a FOR..NEXT loop to determine which value gives a TRUE evaluation as you press a key.

For the curious, you may like to see what happens when you use INKEY$(-n) instead of INKEY(-value) described above. (Remember that it produces a string, not a number, so watch out for the old "Type mismatch...").

Rob Pickering

LOGIC (PART 2)

Operation of the IF statement

Without giving a precise and accurate syntactic definition of the IF statement we can summarise its syntax as: IF 'condition' THEN 'statement'. Consider the 'condition' part which is simply a logical expression which evaluates to TRUE or FALSE. For instance A=B evaluates to TRUE if A is 4 and B is 4. You can try this out with a few PRINT statements:

```
PRINT 4=4  gives -1 (TRUE)
PRINT 4<>2*2  gives Ø (FALSE)
PRINT 2=2 OR 3=9 gives -1 (TRUE)
PRINT 7>9 OR 4<5 AND 3=4  gives Ø (FALSE)
```

The last one needs some thought. There are rules of precedence in logic just as there are in arithmetic, and if there is a choice the computer always evaluates NOT, AND, OR/EOR in that order with OR/EOR having equal priority. The moral here is for YOU to bracket them in the order that you want them done.

You may be able to see why the expression A=B=Ø will not set A and B both equal to zero, instead it gives a 'no such variable' message. If you were to set B=Ø first then try A=B=Ø then you would get A=-1 because B=Ø is TRUE and TRUE has the value -1, so A=TRUE, so A=-1.

Complex logical expressions can be used as long as they can be evaluated. For instance: rank$="CO" OR (age>65 AND sex$="M")
Each part is evaluated - let us pretend that rank$="CO" is FALSE, age>65 is TRUE, and sex$="M" is TRUE, then the expression reduces to FALSE OR (TRUE AND TRUE). This can be evaluated using the truth tables given previously - just use Ø for FALSE and 1 for TRUE (forget that TRUE really is -1 for the moment). F OR (T AND T) is the same as Ø OR (1 AND 1) which is the same as Ø OR 1, which is the same as 1, which is TRUE.

A little extra thought will tell us that a statement such as IF age THEN... is also acceptable because if age evaluates to zero then the condition is FALSE, and if the condition evaluates to -1 (or any other value for that matter) it is TRUE.

Flags

Occasionally in a program we require to know whether some action has taken place or not. For example in a 'bubble sort' we want to continue running through the list of numbers to be sorted until no swaps have taken place, then we know that the list must be in order. So we set swapflag=FALSE and only set it to TRUE if a swap has taken place (ie in PROCswap). We keep repeating the routine until no swaps have taken place (ie till swapflag=FALSE). However the statement 'UNTIL swapflag=FALSE' could be replaced by 'UNTIL NOT swapflag' as this has the same logical meaning. The array list(1) to list(number) is now sorted into ascending order.

```
REPEAT
  swapflag=FALSE
  FOR element%=1 TO number-1
    IF list(element%)>list(element%+1) PROCswap
  NEXT
UNTIL NOT swapflag
END
DEF PROCswap
LOCAL temporary_store
 temporary_store=list(element%)
 list(element%)=list(element%+1)
 list(element%+1)=temporary_store
 swapflag=TRUE
ENDPROC
```

[NOTE: The "Bubble" sort is probably the slowest and most inefficient sort that you can have, it is only worth considering for small arrays. We will give the listing for "Quicksort" in a future issue, in our useful PROCedures section.]

Shortcuts in the IF statement

It is fascinating to re-write some of the IF statements in the following ways. It is not to be encouraged because it makes the programs difficult to read, on the other hand it can provide a more economical way of representation.

Logically the statements on the left can be replaced by the statements on the right:

```
IF F<>0 THEN ....                    IF F THEN ....
IF A=3 THEN C=C-1                    C=C+(A=3)
IF Q=6 OR B=3 THEN A=B ELSE A=0      A=-B*(Q=6 OR B=3)
IF A=9 THEN PRINT"0" ELSE PRINT"1"   PRINT -NOT(A=9)
```

Explanation of the 2nd statement above: On the right hand side the part of the statement 'A=3' evaluates to 0 if FALSE and -1 if TRUE, so the statement becomes C=C+(0) or C=C+(-1) ie C=C or C=C-1. I leave you to work out the 3rd pair of statements.

Further Reading: "Computer Science" by C.S.French published by DP Publications covers logic to a greater depth, and will go further into Venn Diagrams, Karnaugh maps, De Morgans laws etc. It will show you how to prove (amongst other things) that: NOT(A AND B) is equivalent to NOT A OR NOT B

The program below is useful if you would like to experiment with logical expressions. (For example if you are doing an 'A' level in Computing Science it can help with your homework). It will generate truth tables for logical expressions of pretty well any length. The only restriction is that the variables used must be single capital letters starting from A. Do not miss any out, for example B AND A OR C is OK whereas B AND A OR D is not because you would then get a truth table for all the variables A-D. To use it just run the program and enter the expression. You can use any of the following operators in the expression: ( ) = < > OR, AND, EOR, NOT, TRUE, FALSE.

Once you have typed in the program, if you have a printer, try listing it using the LISTO7 option; the result is really quite pretty.

```
  10 REM Logic expression evaluator
  20 REM   by Sheridan Williams
  30 REM----------------------------
  40 ON ERROR GOTO 500
  50 CLS
  60 INPUT'"Logical expression ",
expression$
  70 IF LEN(expression$)<3 END
  80 NV=FNnvariables
  90 CLS
 100 FOR A=1 TO NV
 110 PRINT CHR$(A+64);" ";
 120 NEXT
 130 PRINT"X"
 140 FOR A=1 TO NV+1
 150 PRINT"- ";
 160 NEXT
 170 PRINT
 180 IF NV>0 FOR A=0 TO -1 STEP -1
 190 IF NV>1 FOR B=0 TO -1 STEP -1
 200 IF NV>2 FOR C=0 TO -1 STEP -1
 210 IF NV>3 FOR D=0 TO -1 STEP -1
 220 IF NV>4 FOR E=0 TO -1 STEP -1
 230 IF NV>5 FOR F=0 TO -1 STEP -1
 240 IF NV>6 FOR G=0 TO -1 STEP -1
 250 IF NV>7 FOR H=0 TO -1 STEP -1
 260 IF NV>8 FOR I=0 TO -1 STEP -1
 270 IF NV>0 PRINT FNvalue(A);
```

```
280 IF NV>1 PRINT FNvalue(B);
290 IF NV>2 PRINT FNvalue(C);
300 IF NV>3 PRINT FNvalue(D);
310 IF NV>4 PRINT FNvalue(E);
320 IF NV>5 PRINT FNvalue(F);
330 IF NV>6 PRINT FNvalue(G);
340 IF NV>7 PRINT FNvalue(H);
350 IF NV>8 PRINT FNvalue(I);
360 PRINT FNvalue(EVAL(expression$))
370 IF NV>8 NEXT I
380 IF NV>7 NEXT H
390 IF NV>6 NEXT G
400 IF NV>5 NEXT F
410 IF NV>4 NEXT E
420 IF NV>3 NEXT D
430 IF NV>2 NEXT C
440 IF NV>1 NEXT B
450 IF NV>0 NEXT A
460 PRINT"Logical expression was"
470 PRINT"X = ";expression$
480 GOTO 60
490 END
500 IF ERR=17 PRINT:END
510 PRINT''"An error has occurred
, maybe it's       because you
 typed the expression"
520 PRINT"incorrectly, please check.
 The"
530 PRINT"expression was:-"'expression$
540 GOTO 60
550 DEF FNvalue(X):IF X
 THEN ="T " ELSE ="F "
560 DEF FNnvariables
570 LOCAL letter_code,count
,position,temp$,x,operator$
580 DATA TRUE,FALSE,AND,EOR,OR
,NOT,),(,=,<,>," ",*
590 temp$=expression$
600 RESTORE
610 REPEAT
620 READ operator$
630 REPEAT
640 x=INSTR(temp$,operator$)
650 IF x>0 temp$=LEFT$(temp$,x-1
)+MID$(temp$,x+LEN(operator$))
660 IF temp$="" PRINT"Error in
 logical expression":STOP
670 UNTIL x=0 OR LEN(operator$)
>LEN(temp$)
680 UNTIL operator$="*" OR
 LEN(operator$)>LEN(temp$)
690 count=0
700 FOR letter_code=65 TO 72
710 position=INSTR(temp$,
CHR$(letter_code))
720 IF position<>0 count=count+1
730 NEXT letter_code
740 =count
```

S.W.

PROCEDURE FUNCTION LIBRARY

This month we give the listing of a short procedure to produce double height text when in the teletext mode. Lines 1000-1060 represent a demonstration program which calls the procedure (listed from 20000) to produce the double height word "HEADING". As it stands the procedure does not cater for coloured text, though such a facility could be easily added. One more point, the text should not go over a screen line. To print on more than one line you need to re-call the procedure for each new line.

This procedure, unlike those previously published, was taken (with permission) from a 51 page booklet entitled "101 Procedures and Functions for the BBC Microcomputer". It is published by Douglas Computer Services, 75 Caterham Drive, Coulsdon, Surrey, CR3 1JP, and costs £3.00 plus 40p postage. A book of procedures of this kind is obviously an excellent idea, and could prove to be extremely useful to many Beeb users. But we found the selection offered here a bit disappointing in two respects. Firstly the choice of procedures was not in our view well thought out - though we would stress that this must be to a certain degree a matter of taste, and you could find that the selection offered was just what you were looking for! Secondly, the standard of programming was not as high as we would have hoped, and the writing showed a less-than-complete awareness of the facilities offered by the Beeb. Notwithstanding, the volume seems to represents quite good value at £3.

```
1000 REM DOUBLE
1010 REM TO PRINT DOUBLE HEIGHT CHAR
ACTERS IN MODE 7
1020 MODE 7
1030 PRINT TAB(15,10);
1040 word$="HEADING HEADINGHEADING H
EADINGHEADING HEADINGHEADING "
1050 PROCdouble(word$)
1060 END
```

```
20000 DEF PROCdouble(word$)
20010 LOCAL column,row
20020 column=POS
20030 row=VPOS
20040 PRINT CHR$141;word$
20050 PRINT TAB(column,row+1);CHR$141
;word$
20060 ENDPROC
```

DEBUGGING (PART 2)

by Rob Pickering

This series is all about how to track down those little bugs in your programs which can seem all too elusive to beginners and more experienced users alike. Each issue will take a look at some common error messages and the likely causes. In this issue I look at some of the more common bugs which creep in when a working program is typed in from a printed listing. Quite a lot of members encounter these kind of problems when entering programs from the newsletter. The chief offenders appear to be "Moonlander", "3D Noughts and Crosses", and "Chunky Invaders". All of these published programs however, were listed directly from the computer, and are error free, but errors creep in after many an hour's weary typing.

Last month I mentioned a single pitfall that can occur when a vital space is omitted in an IF...THEN instruction. Here are four simple syntax traps and two more subtle ones.

(a) Letter "l" and Digit "1".

It is often difficult to distinguish between the lower case letter "l" and the digit "1", but a computer just won't let you get away with guessing. If you get an error message concerning a line in which you have guessed, then look at the program line numbers which will almost certainly contain a "1" digit somewhere, then you can compare it with any character which comes into question.

(b) Letter "O" and digit zero

Even more common than the above is confusing the letter "O" with the digit zero. Different printers tend to distinguish them in different ways, again, look at the line numbers to see what a zero looks like. If you want to look for this sort of error in your own program then I suggest that you use a mode other than seven (TELETEXT MODE) when examining the listings. This is because TELETEXT mode does not show much visible difference between the two characters, whereas other modes cross the circle with a line to make it distinguishable as a zero and the two are then easy to tell apart.

(c) DOT and COMMA

Because the dot (full-stop) and comma keys are right next to each other it is all too easy to press the wrong one and not notice. If you are entering a DATA statement with numbers separated by commas then entering a dot by mistake may not produce the error you expect. You might have entered one number with a decimal point instead of two inetegers; consider:

```
10 DATA 10,20,30,40,50,60,70,80,90
10 DATA 10,20,30,40.50,60,70,80,90
```

and perhaps you will also see just how hard it would be to spot such an error, especially without the correct line with which to compare it. This may give an unexpected "Out of data" message, due to there being only eight numbers in the second version as opposed to the nine integers in the correct top line. Beware also if the data is supposed to contain some numbers which are floating point, making it extremely difficult and sometimes almost impossible to spot extra decimal points, eg:

```
10 DATA 14.6,17,6,98.2,3 may be typed as: 10 DATA 14.6,17.6,98.2,3
```

Again this would cause an "Out of data" message.

(d) UPPER CASE and LOWER CASE

As you will probably be aware the BBC BASIC treats upper and lower case characters as being distinct from each other. This is not the case on most other versions of BASIC, so those of you who are used to using another micro will have to beware of entering KEY WORDS in lower case, or using upper case at one time and lower case another time when trying to refer to the same variable. It is very easy to forget to take the CAPS LOCK off after putting it on, so lookout for mixing your cases.

It is easy to underestimate the importance of (a)-(d) above; they may seem obvious and even trivial but it's suprising how many times they are the sole cause of errors. Use them as a check-list each time you start debugging a program. And now to the first specific problem of the series.

(1) The MOONLANDER program in the April newsletter contained the line:

```
950 READ K,L
```

and many people complained about it causing the error message "No such variable at line 950". This confused me to say the least. As far as my logic was concerned, this was totally impossible, because the "no such variable..." message indicates that you are using a variable which has not previously been assigned a value. But line 950 only assigns variables K and L, it does not seem to reference any! Eventually the cause came to light; to explain, try the following:

```
10 NUM1=50
20 READ VAR1,VAR2
30 PRINT"values are ";VAR1,VAR2
40 STOP
50 DATA NUM1,1/3
```

It seems that if you read a numeric variable from data, and the data contains a variable or expression, then it reads instead that value from the data. Sorry but I can't make it sound any less complicated.

The point is, if you accidentally put alphabetic characters into data statements, you can produce the "No such variable..." error. This is because the BASIC interpreter assumes that when assigning to a numeric variable, any alphabetic characters must mean that you're referring to a variable. But because the accidental variable name does not actually exist, it quite rightly tells you so.
[Acorn have said that, although undocumented, this feature will remain in future releases of BASIC. Ed]

The most common way to cause this error is to enter a letter "O" in data instead of a zero digit. Also you should beware of the case when you are actually using the variable "O" elsewhere in the program, which would not result in an error message, but would simply result in the wrong value being assigned; this would be even harder to find !

(2) One problem causing a "Syntax error..." in lines which make use of BASIC keywords such as TAB(n) or INT(n) is due to extra spaces being placed where they should not be. Nearly all of the keywords requiring an argument (n in the brackets above) require these brackets in order for the computer to recognise them.

If a space is placed between the keyword and its opening bracket, the word is not syntactically correct. Thus;

```
PRINT TAB(10)"TEST" is correct,
while  PRINT TAB (10)"TEST" is not correct,
```

and would produce a "Syntax error..." message if used.
Unfortunately, some magazines have made it policy to insert spaces at various points in order to achieve greater readability of program listings. I'm not sure whether this is acceptable on other BASICs but it certainly is not acceptable in BBC BASIC.

Next month I shall continue with specific errors, and how to trace them. Included will be the problems caused by array subscripts, and I will take as an example the "3D Noughts & Crosses" program of the April issue.

Rob Pickering

Program tested on 0·1 and 1·1 O.S.

UNION JACK (16k/32k)

by David Stonebanks

This program produces a whole series of displays of the union flag in a wide variety of shapes and sizes, including a nice multiple display. The graphics work effectively and produce quite striking results on a colour monitor. As it stands the program requires a 32k machine, but you need only change line 20 to MODE 5 to make it work in 16k.

```
 10 ON ERROR VDU23;11,255;0;0;0
:STOP:REM restore cursor
 20 MODE1:REM high res graphics,
use MODE 5 on 16k micro
 30 VDU23;11,0,0;0;0;:REM delete
 cursor
 40 black=0:red=1:blue=2:white=3
 50 VDU19,blue,4,0,0,0
 60 VDU29,640;512;:REM set graphics
 origin to screen centre
 70 FOR hsize=5 TO 60 STEP 10
:REM horizontal size
 80 vsize=hsize:REM square
 90 PROCunion_jack
100 PROCdelay(100)
110 NEXT hsize
120 CLS
130 FOR hsize=5 TO 50 STEP 10
140 FOR vsize=5 TO 60 STEP 10
150 PROCunion_jack
160 PROCdelay(100)
170 NEXT vsize
180 CLS
190 NEXT hsize
200 CLS
210 hsize=10:vsize=10
220 FOR X=160 TO 1200 STEP 320
230 FOR Y=100 TO 900 STEP 200
240 VDU29,X;Y;:REM move graphics origin
250 PROCunion_jack
260 NEXT Y
270 NEXT X
280 PROCdelay(200)
290 GOTO 20:REM start again
300 DEFPROCunion_jack
310 PROCbox(blue,12,8)
320 PROCdiag(white,9,8,-12,-6,12,8,-12,-8)
330 PROCdiag(white,12,8,-12,-8,12,6,-9,-8)
340 PROCdiag(white,-9,8,-12,8,12,-6,12,-8)
350 PROCdiag(white,-12,8,-12,6,12,-8,9,-8)
360 PROCdiag(red,10.5,8,0,1,12,8,0,0)
370 PROCdiag(red,0,0,-12,-8,0,-1,-10.5,-8)
380 PROCdiag(red,0,0,-12,8,0,-1,-12,7)
390 PROCdiag(red,0,0,12,-8,0,1,12,-7)
400 PROCbox(white,2,8)
410 PROCbox(white,12,2.5)
420 PROCbox(red,1,8)
430 PROCbox(red,12,1.5)
440 ENDPROC
450 DEFPROCbox(colour,halfx,halfy)
460 GCOL0,colour
470 MOVEhalfx*hsize,halfy*vsize
:MOVE-halfx*hsize,halfy*vsize
480 PLOT85,halfx*hsize,-halfy*vsize
:PLOT85,-halfx*hsize,-halfy*vsize
490 ENDPROC
500 DEFPROCdiag(colour,x1,y1,x2,y2,x3,y3,x4,y4)
510 GCOL0,colour
520 MOVEx1*hsize,y1*vsize
:MOVEx2*hsize,y2*vsize
530 PLOT85,x3*hsize,y3*vsize
:PLOT85,x4*hsize,y4*vsize
540 ENDPROC
550 DEFPROCdelay(t)
560 T=TIME+t:REPEAT UNTIL TIME>T
570 ENDPROC
580 END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

3-D LETTERS FOR HEADINGS

This short demonstration by G F Weston shows how to obtain a 3-D effect with text. VDU 5 must be used so that the cursor may be offset by a fraction of a character before overprinting in black. GCOL 1,1 is used to OR together the colours so that the black dosn't overwrite the white lettering.

```
10 REM 3-D HEADINGS
20 REM by G.F.Weston
30 REM
40 MODE 5
50 VDU 5
60 VDU 19,0,4;0;19,1,0;0;
70 MOVE 450,600
80 PRINT"BEEBUG"
90 GCOL 1,1
100 MOVE 442,592
110 PRINT"BEEBUG"
```

IT'S QUICKER BY TUBE

THE PROBLEM

The BBC machine, even in its most currently expanded form - that is to say a model B with discs, printer, and other interfaces, is only really half a computer. If you have ever pondered (or worried) over the amount of free memory in the machine when it runs in the higher resolution graphics modes - especially with discs - then you may have begun to suspect as much yourself. In modes 0, 1, and 2 the graphics uses 20k of memory. The 12k of RAM remaining (32-20=12) is unfortunately not all free for storing Basic programs and associated variables. The bottom 3.5k is used by the operating system, leaving around 8k (8704 bytes) for Basic. However if you are using discs, the disc work space will use up a further 3k; and the extended character definitions available in the later operating system take a further 1.5k, leaving just 4k of memory for storing Basic programs and their variables.

Now obviously you cannot do much in 4k, and it is not just a question of buying an add-on memory board because the 6502 processor in the Beeb is working to full stretch. It can only address 64k of memory - and this is already totally filled with 32k of RAM and 32k of ROM (to be technically correct a little of the ROM addressable memory has been taken over with input/output addressing - things are that tight).

TEMPORARY SOLUTION

The temporary solutions are to go easy on the high resolution graphics modes and keep in the Teletext mode wherever possible, so saving 19k of memory over modes 0, 1, & 2. (Refer to our double Teletext article in the July issue for an introduction to this extremely useful mode). If you have programs of even moderate length that use the higher resolution modes, then you may find that they will not work when the disc is booted up - so you may have to continue to load them from cassette. Even if you do not use discs, you are only left with around 8k for programs that use the first 3 graphics modes, and this usually leaves things so tight that you dare not use too many REM statements, or include user instructions, or make use of the excellent long variable name facility on the Beeb.

THE TUBE - LONG TERM SOLUTION

The long term solution is to use the Tube. This is a fast means of communication through which a model B is able to communicate with a second microprocessor system. In such a configuration the model B would act essentially as an input/output device for the second processor. It would handle keyboard, discs, printer, screen, clock etc, while the second processor would do the rest - executing the central processing function of running the program proper, and interpreting the commands issued in whatever high level language was in use (eg Basic, Pascal, etc). The advantages of such a system would be that much of the second processor's full addressable memory (64k or more) would then be available for the user's programs. And secondly, considerable gains in speed could result - partly because you would have two processors carrying out different tasks at the same time, and partly because the second processors envisaged run at a higher speed than the Beeb itself.

A further advantage is that there is (or will be) a choice of second processor. In the pipeline is a fast 6502A-based processor board, a Z80 unit (which can run CP/M), and a 16 bit National semiconductor 16032 based machine which can directly address up to 16 megabytes of memory.

Of course it is good to know that the Beeb is fully expandable in this way. But looked at from a financial point of view it is bad news to learn that you have in fact only bought half a computer. As to the cost of the other half, prices are not yet finalised, but it is thought that the 6502A unit will cost around £200 (includes

VAT). This seems very reasonable, but nevertheless represents a considerable further outlay.

NEW INTERPRETERS

In order for the second processor to run in Basic or any other language, it must have access to an interpreter; and interpreters are currently being written for the Z80 and the 16032. The 6502A second processor unit is able to initiate a copying-over routine when first switched on, which makes a copy in its own RAM of the model B's 16k Basic ROM. Transfer is extremely fast, and this only takes about one second, though at first sight it does seem rather wasteful to have to keep two copies of the same 16k of code.

PROGRAM COMPATIBILITY

The New User Guide stresses the need to make programs compatible with the Tube - and to do this you need essentially to avoid direct access to memory via the ? or ! operators (ie via PEEK or POKE). The reason for this is that the second processor has no way of directly accessing the first processor's memory - though this is possible through the use of OSBYTE, OSWORD and FX calls. For this reason if you use the ? operator to send characters directly to the screen, for example in a program for the Beeb, that same program cannot run from across the Tube, because the screen memory addresses will not be directly accessible by the second processor that is actually running the program. Moreover the screen addresses POKEd will now correspond to RAM used for an entirely different function on the second processor board (eg storing the program that you are running).

It is possible to perform most operations in a Tube-compatible way on the Beeb, though there may be some inherent loss of speed when this is done; and secondly, there are some vital calls (such as those to access any area of RAM on the first processor, and to access the user port) which are not implemented on operating system 0.1. If you are writing in assembler, then your programs could only possibly work across the Tube if the second processor being used is a 6502 (or 6502A). The Z80 and 16032 would make no sense whatsoever of 6502 op codes, and would just hang up if confronted with a program in 6502 machine code. Perhaps for this reason, even Acornsoft have not attempted to make machine code programs such as Snapper or Defender work across the Tube. In these programs, direct access is made to the screen in machine code, so achieving a speed which would probably not be possible in a Tube-compatible way. We should add that our program writers have also come up against these difficulties, and that BEEBUGSOFT programs such as MAGIC EEL, and one of the machine code games to be announced next month will not work across the Tube (this of course should not be taken as an indication that any of the others will work across the Tube - we will not know until systems are available for testing). As to programs which you may be writing, it would be as well to heed Acorn's warnings, and make them Tube-compatible where possible. But, particularly in the case of machine code programs, it might be as well to follow the lead of Acornsoft, and not let the dictates of the promised Tube cramp your style too much.

D.E.G.

PLEASE NOTE the address on the adjacent advertisement should be 36A and not 364 as advertised last month. We apologise for any inconvenience this may have caused.

SOFTWARE REVIEW

Program: CHESS
Price: £9.2Ø
Reviewer: Ian Bell

Supplier: Program Power
Requirements: Model A.

I found that the delivery time was good and it is an excellent chess package except for one let down. It uses teletext graphics which produces a rather ugly display. Perhaps this is to be expected with a program that works on a model A, but I hope that a model B program will contain better graphics. Program Power say that this is in the works.

There is the usual player choice of colour, and illegal move rejection. Moves are made using algebraic notation ie e2-e4 etc. There are six levels of play. The more exotic features are 'en passant', delete last move, (useful if you have made a wrong move), and a replay feature where the program remembers the game and can display it move by move so you can see where you went wrong. It is also possible to 'set up' any initial position and play from there. In the 'Blitz' chess option you must make your moves in 1Øsecs (all games display a clock of your time) or forfeit it. The program gets slightly confused if you are in check when forfeiting your move. The program plays a strong game; level 1 takes about 5secs to move, it plays so well that it may be a disadvantage since a learning player will probably get thrashed on this the lowest level. Level 2 is much more challenging, and you cannot afford to make any mistakes when playing this; response time is about 2Øs. I have not attempted level 3 since it does take the better part of a minute or two to move. This is a very "clean" program, it was easy to load, and you can resign at any time by pressing 'escape'. Apart from the display I find it difficult to criticise this program. I would have liked a winking character on the higher levels to reassure me that it hadn't stopped working.

In short, excellent value for money unless you have your heart set on a high resolution display. Especially recommended for model A users.

Program: SPACE PIRATES
Price: £8.ØØ
Reviewer: Lloyd Gosden

Supplier: BUG-BYTE
Requirements: Model A.

This is an arcade type game. I had no trouble in loading it once I had cleaned the heads and pinch roller of my cassette - a task I find frequently necessary. The game is relatively fast.

At first I thought the game was very simple, but soon found that a great deal of mental agility was required. The player controls a space ship which can be moved forwards or rotated clockwise or anti-clockwise. The ship fires at 'robotic space pirates' and the player gains a point for each pirate destroyed. The pirates will destroy your ship if they get near enough, or they will steal your 'life pods'. If your space ship is destroyed then a life pod will re-incarnate it! Adequate, clear instructions were included. The game is in black and white, but the graphics were fairly impressive, and the sound effects were good. I always feel that more use could be made of sound on the BBC micro.

At £8.ØØ I thought the game was expensive, but it's good fun and addictive.

Program: PLANETOID (formerly DEFENDER) Supplier: Acornsoft
Price: £9.95 Requirements:Model B, or Model A with 32k & 6522VIA.
Reviewer: I.G.Nicholls & Sheridan Williams

This game is certainly very good, I could even use superlatives like 'terrific', however it depends a great deal on the user. Some people are put off by its complexity and would consequently not rate it so highly. Defender is an arcade type game, where you have to kill the descending alien ships before they carry off the humanoids that live on the mountainous surface of the planet. If one of the humanoids is being lifted towards the top of the screen by a 'lander' you must shoot to kill it, (missing the humanoid of course) and airlift this humanoid back to the surface. If the lander and the humanoid reach the top of the screen it turns into a 'mutant' which will then seek you out very quickly. There are 'baiters' as well, and after you have survived the first attack wave, there are 'bombers', 'pods' and 'swarmers'. All of these 'nasties' fire missiles at you and move independently about the screen. In all there are 7 controls that you must master. These are: Up, down, accelerator, fire, reverse, hyperspace and 'smart bomb'. As you can probably imagine it takes considerable practice and patience before one has mastered the game, and it is this aspect that may put people off. However, we both found the game totally addictive.

The graphics are very good, but do really require a colour monitor, those with only a colour TV or B&W set will find that they are missing vital information summarised in the top of the screen. There are some quite exhilerating sounds, especially at multiples of 10,000 score. As long as you consider yourself fairly dextrous this is a really good game.

Program: GRAPHS and CHARTS Supplier: Acornsoft
Price: Cassette - £9.95, Book - £7.50 Requirements: Model A
Reviewer: Norman Kirkby.

This is a full package. It covers histograms, pie charts, 2 and 3-D graphs, 3-D surface plotting, 2 and 3-D contour surfaces, and stereo graph and surface plotting. There are 23 programs and procedures grouped in three levels. Level 1 consists of 6 fundamental procedures which you can append to a program you are writing using the *EXEC command (explained in BEEBUG p16 issue 3). These cover eg. plotting a histogram bar, the fundamentals of a 3-D plot. They provide the greatest freedom of all but you must do some program writing to bind them together. Level 2 consists of 9 procedures, and they are more fully worked out. You have to write less, but you have less freedom (although a feature of the package is the ease with which you can amend routines to suit, aided by many helpful remarks and consistent variable names, line numbers and built-in choices). All the procedures range from 5 to 384 lines in length, and example programs are given in the book showing how to use each procedure.

Level 3 consists of 8 fully written ready-to-run programs demonstrating all but the stereo plots. You input data. The book lists the programs and procedures and gives helpful suggestions and explanations. The cassette contains all the programs except the level 2 example programs mentioned above which are very short. The package is up to the high Acornsoft standard. Screen presentation is very good and a deal of skill is obvious throughout. I recommend you to get both the book and the cassette (the latter because some of the programs are very long). The package will appeal to anyone wanting to obtain programs covering these areas for teaching, small/medium business applications, research, self education and hobby use. All but the histogram and pie chart routines naturally require a mathematical knowledge. All run on a 16k machine, but some will need "MODE 1" changed to "MODE 5". If you just

want to plot 2-D graphs, BEEBUGSOFT's SUPERPLOT is a better alternative (and cheaper).

[You may not believe this, but the above is a totally unsolicited assessment from an independent reviewer. Ed]

Program: INVADERS
Price: £6.90
Reviewer: D.E.Graham

Supplier: IJK Software
Requirements: MODEL B

This is a fairly standard space invaders type game with formations of invaders that move across and then down the screen while you shoot them out of the sky with your trusty laser. The game is in colour, and is apparently well written. Key control is smooth, and alien movement begins slowly but gets very fast towards the end of a screen. A nice feature of the program is the variety of options that can be chosen before the start of the game. These include faster laser fire, faster invaders, faster bombs and invisible invaders. The speeding up achieved by these options is considerable, so that if you chose to have faster laser fire, but leave the invaders normal, then you could quite easily put on a pretty creditable performance. Invisible invaders was a bit disappointing in that their bombs were also invisible, which made it rather too difficult to play for my liking.

My only reservations about the game are on the graphics, which were much better than adequate, but left me thinking that a higher resolution mode should ideally have been used, but this would have resulted in a less colourful display. This program represents quite a good buy.

POINTS ARISING

JOYSTICK NOTE

Quite a few people have written to say that they have bought joysticks and connected them to their BBC micro and they fail to work with such programs as Snapper, Starfire etc. The point is that joysticks will only work if the software is designed for use with them. Most programs for the BBC machine that are available at present have no joystick input routine.

Next month we will be reviewing joysticks, and will cover such matters more fully - as well as giving a joystick routine for "Starfire".

MINI TEXT EDITOR

In the June issue we gave a 'Mini word processing' program. We also said that we would publish a routine to remove the line numbers on printing, Apologies for not including it earlier but space is at a tremendous premium and the modification requires a re-listing of the whole program, and some further notes. However, because of the demand, we will publish it in the November issue.

STARFIRE CORRECTIONS

The following corrections will allow Starfire to work on all versions of the operating system:-

```
 420 *FX4,1
1680 MOVE 256+kills%*64,1010:PRINT CHR$230
2345 COLOUR 5
2350 VDU4:PRINT TAB(5,10)"GAME OVER"
```

John Webb has pointed out two problems concerning programs in the July and September issues. They manifest themselves in subtle ways:

CHUNKY INVADERS

Line 40 needs to be modified to: `40 SCORE=0:H%=0:F1=0:F2=0:C%=0`
because C% is not set to zero when the program is run.

HIGHER/LOWER

The program jumps out of a FOR...NEXT loop at line 100, resulting in a "Too many FORs" error eventually, so modify line 100 to read: `100 IF C<MB J=9:GOTO 130`

FUNCTION: Days Between

On p28 of the Sept issue we gave a function for finding the number of days between two given dates. Although there is no error in the function R.J.Little has written pointing out that the years 2100, 2200, and 2300 are NOT leap years, but 2400 IS. Hence the function is not valid up to 28/2/2400 but only up to 28/2/2100. To correct the article just change both occurrences of 2400 to 2100.

Henrietta Holocaust points out that in the July issue p 25 'Input Function' that to lock out all lower case keys the 97 in line 1221 should be 96. Note also that to leave all the lower case keys enabled replace the 97 with 122.

LOCAL USER GROUP INDEX

BEEBUG is happy to act as an information point for local groups. If you would like to be in our index, just drop a line to us and mark the envelope "Local user groups".

Aldershot/Farnham
B. Carroll
The Cottage
42, Manor Road
Aldershot GU11 3DG

Aylesbury
Chris Pyves
Aylesbury Computer Club
12 Bishop's Walk
Aylesbury
Bucks. HP21 7LF
(0296 26347)

Bedfordshire
D. L. Evans
23 Hitchin Road
Henlow Camp
Bedfordshire.

Brighton Area
Jim Price
Bedford House
27/28 St Georges Rd
BRIGHTON
Sussex.

Bromley Area
D.D.Singer
86 Southborough Road
Bromley, Kent.
BR1 2EN

Cambridge
John Harris
Bottisham Acorn User Group
1 Rowan Close
Bottisham, Cambs.
(Tel: 0223 811487)

Chatham
C. Rutter
Medway Acorn Users group:
St John Fisher School
Ordnance Street
Chatham. ME4 6SG
(Tel: Medway 42811)

Mid-Cornwall
Mid-Cornwall College of F.E.
Palace Road
St Austell
(Tel: Par 2399 any time)

Chelmsford & Mid Essex
Jonathan Stone
200 Chignal Road
Chelmsford
Essex
(Chelmsford 59665 or
59708 (Answering machine))

Dudley Area
Roger Luff
(Tel: Kingswinford 288721)

Fareham & Portsmouth
Peter Smith
23 Sandy Close
Petersfield, Hants
(Tel: Petersfield 4059
evenings only)

[Group meets at 7pm on the 3rd Monday of each month at the Portchester Community Centre.]

Fareham & Portsmouth
Bill Tettle
Portchester School
White Hart Lane
Portchester
Fareham, Hants
PO16 9BT

Gloucester
Laurie Dann
10 Highbank Park
Gloucester GL2 9DY
(Tel: Gloucester 21245)

London (West)
West London
Personal Computer Club
Contact: Graham 01-997 8986
or Neil 01-997 9437

Merseyside
Fred Shaw
Merseyside Micro Group
14 Albany Avenue
Ecclestone Park
Prescot
Merseyside. L34 2QW

Norwich & District
Paul Beverley
Norwich City College
Room B12a
Ipswich Road
NORWICH NR2 2LJ

Peterborough
Tony Goodhew
Sir Harry Smith College
Whittlesea
Peterborough

Preston
D Coulter
8, Briar Grove
Ingol, Preston
Lancs. PR2 3UR

Stoke-on-Trent
POTBUG
M G Forster
8 St Georges Ave
High Lane
Tunstall
Stoke-on-Trent
(Tel 818499)

Woking
John Gardner
22 St John's Road
Woking, Surrey

EDUCATIONAL SUB-GROUPS

Cheltenham
P.J. Heslip
Stroud Teachers' Centre
Arthur Dye School
Springbank Road
Cheltenham, Glos.

Warrington
D.H.Maddock
18, Brian Avenue
Stockton Heath
Warrington
Cheshire

OVERSEAS

AUSTRALIA
Lindsay Thachuk
Scotch College
Carruth Road
Mitcham
S. Australia, 5062
Tel: (08) 272 3682

NORWAY
M. Christiansen & K.H.Howells
Marienlystveien 2 - Stavne
N-7000 Trondheim
Norway.

NEW ZEALAND
B.M.Wilkinson
Einstein Scientific
177 Willis Street
PO Box 27138
Wellington
Tel:851-055

Program tested on 0·1 and 1·1 O.S.

CALENDAR GENERATOR (16k)

by Michael Quinion

From early times the calculation of the calendar and, in particular, the determination of the date of Easter, have traditionally been among the most important of arithmetical tasks. The following program will do both these tasks for the modern Gregorian calendar (the starting dates differ for each country because different countries started using the Gregorian calendar at different times). This program assumes a starting year of 1753.

This program is very well written, and is different from the usual run of games and utilities. It illustrates the use of modular arithmetic in a way I've not seen elsewhere. It also demonstrates the considerable increase in speed you achieve if you use integer variables throughout the program. It requests a year and month and then displays the calendar for that month. You can easily change year and month, or you can step forward or backward by a month at a time, and this is achieved at a really surprising speed.

```
1010 REM   ** GREGORIAN CALENDAR **
1020 REM   (C) Michael Quinion 1982
1030 REM   (Version 1.6 of 7/9/82)
1035 ON ERROR GOTO 3000
1040 REM ==========================
1050 MODE7:*FX144,255
1060 *FX4,1
1070 A%=0:B%=0:C%=0:CHARA$="":CHARA%=
0:D%=0:DAYNO%=0:E%=0:EDAY%=0:ELINE%=0
1080 EMONTH%=0:F%=0:G%=0:H%=0:I%=0:J%
=0:K%=0:L%=0:LEAPFLAG%=0:MAXL%=0
1090 MAXV%=0:MINL%=0:MINV%=0:MONTHPRI
NT$="":MONTH%=0:NUM$=STRING$(8," ")
1100 O%=0:Q%=0:R%=0:S%=0:SHIFT%=0:STA
RT%=0:T%=0:U%=0:X1%=9:X2%=X1%-2:X3%=1
1110 XI%=0:Y1%=5:Y2%=Y1%+2:Y3%=22:YEA
R$="":YEAR%=0:YEARNO%=0:YI%=0
1120 DAYS$="                    1  2
 3  4  5  6  7  8  9 10 11 12 13 14"
1130 DAYS$=DAYS$+" 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31"
1140 DOW$=" M  T  W  T  F"+CHR$(129)+
" S  S"
1150 R$="  D: new date  F: forward  B
: back"
1160 DIM MONTHLEN%(11), MONTHNAME$(11
), MONTHNO%(11)
1170 FORZ%=0TO11:READ MONTHLEN%(Z%),M
ONTHNAME$(Z%),MONTHNO%(Z%):NEXT
1180 REM ================
1190 REM   MAIN PROGRAM
1200 REM ================
1210 PROCsetscreen
1220 PROCdate
1230 PROCeaster:PROCyearno:PROCleapye
ar
1240 PROCdaystart:!&FE00=&102000A:PROC
printmonth
1250 PROCchoose:!&FE00=&10720A
1260 ONFLAG%GOTO1220,1230,1240
1270 END
1280 REM ========
1290 REM   DATA
1300 REM ========
1310 DATA 31,January,1,28,February,4,
31,March,4,30,April,0
1320 DATA 31,May,2,30,June,5,31,July,
0,31,August,3
1330 DATA 30,September,6,31,October,1
,30,November,4,31,December,6
1340 REM ==============
1350 REM   PROCEDURES
1360 REM ==============
1370 DEFPROCeaster
1380 A%=YEAR%MOD19:B%=YEAR%DIV100:C%=
YEAR%MOD100:D%=B%DIV4:E%=B%MOD4:I%=C%D
IV4
1390 G%=(8*B%+13)DIV25:H%=(11*(B%-D%-
G%)-4)DIV30:F%=(7*A%+H%+6)DIV11:K%=C%M OD4
1400 J%=(19*A%+B%-D%-G%+15-F%)MOD29:L
%=(32+2*E%+2*I%-K%-J%)MOD7
1410 EMONTH%=((90+J%+L%)DIV25)-1:EDAY
%=(20+J%+L%+EMONTH%)MOD32:ENDPROC
1420 DEFPROCyearno
1430 Q%=YEAR%DIV100:R%=YEAR%MOD100:S%
=R%DIV12:T%=R%MOD12:U%=T%DIV4
1440 YEARNO%=(S%+T%+U%+(19-Q%)*2-(Q%>
19))MOD7:ENDPROC
1450 DEFPROCleapyear
1460 LEAPFLAG%=TRUE
1470 IFYEAR%MOD400=0THENENDPROC
1480 IFYEAR%MOD100=0THENLEAPFLAG%=FAL
SE:ENDPROC
1490 IFYEAR%MOD4=0THENENDPROC
1500 LEAPFLAG%=FALSE:ENDPROC
1510 DEFPROCdaystart
1520 DAYNO%=YEARNO%+MONTHNO%(MONTH%)+
6
1530 DAYNO%=(DAYNO%+((MONTH%=0ORMONTH
%=1)ANDLEAPFLAG%))MOD7
```

```
 1540 MONTHPRINT$=LEFT$(DAYS$,18+MONTH
LEN%(MONTH%)*3-(MONTH%=1ANDLEAPFLAG%)*
3)
 1550 MONTHPRINT$=MID$(MONTHPRINT$+STR
ING$(30," "),19-DAYNO%*3):ENDPROC
 1560 DEFPROCprintmonth
 1570 ELINE%=(((EDAY%+DAYNO%)*3)DIV21)
*2
 1580 FORZ%=Y1%TOY1%+1:VDU31,X1%,Z%,14
1,130
 1590 PRINTLEFT$(MONTHNAME$(MONTH%)+ST
RING$(8," "),11)+YEAR$:NEXT
 1600 VDU31,X2%,Y2%,131:PRINTDOW$
 1610 START%=1:FORZ%=2TO12STEP2:PRINTT
AB(X2%,Y2%+Z%)MID$(MONTHPRINT$,START%,
21)
 1620 START%=START%+21:NEXT
 1630 IFMONTH%=EMONTH%THENVDU31,X2%+18
,Y2%+ELINE%,129
 1640 ENDPROC
 1650 DEFPROCdate
 1660 PRINTTAB(X3%,Y3%)"      Year?"+ST
RING$(28," ")
 1670 PROCgetinput(X3%+11,Y3%,4,4,1753
,2999):YEAR$=NUM$:YEAR%=VAL(NUM$)
 1680 PRINTTAB(X3%+20,Y3%)"Month?":PRO
Cgetinput(X3%+27,Y3%,1,2,1,12)
 1690 MONTH%=VAL(NUM$)-1:PRINTTAB(X3%,
Y3%)STRING$(30," "):ENDPROC
 1700 DEFPROCchoose
 1710 PRINTTAB(X3%,Y3%)R$;
 1720 O$=GET$
 1730 IFO$="D"ORO$="d"THENFLAG%=1:ENDP
ROC
 1740 IFO$="F"ORO$="f"THEN1770
 1750 IFO$="B"ORO$="b"THEN1820
 1760 GOTO1720
 1770 REM===MOVE FORWARDS?
 1780 IFMONTH%=11ANDYEAR%=2099THEN1720
 1790 MONTH%=(MONTH%+1)MOD12
 1800 IFMONTH%=0THENYEAR%=YEAR%+1:YEAR
$=STR$(YEAR%):FLAG%=2:ENDPROC
 1810 FLAG%=3:ENDPROC
 1820 REM===MOVEBACKWARDS?
 1830 IFMONTH%=0ANDYEAR%=1753THEN1720
 1840 MONTH%=(MONTH%+11)MOD12
 1850 IFMONTH%=11THENYEAR%=YEAR%-1:YEA
R$=STR$(YEAR%):FLAG%=2:ENDPROC
 1860 FLAG%=3:ENDPROC
 1870 DEFPROCsetscreen
 1880 FORZ%=1TO2:PRINTTAB(X1%-2,Z%-1)C
HR$(141)CHR$(131)"Perpetual  Calendar"
:NEXT
 1885PRINTTAB(10,2);"by  M. B. Quinion"
 1890 VDU31,1,3,147:PRINTSTRING$(34,CH
R$(44))
 1900 FORZ%=Y3%-1TOY3%+1STEP2:VDU31,1,
Z%,146:PRINTSTRING$(34,CHR$(44)):NEXT
 1910 ENDPROC
 1920 DEFPROCgetinput(XI%,YI%,MINL%,MA
XL%,MINV%,MAXV%)
 1930 NUM$="":VDU31,XI%,YI%,32,32,32,3
2,32,32,8,8,8,8,8,8
 1940 CHARA$=GET$:CHARA%=ASC(CHARA$)
 1950 IFCHARA%=13THEN2000
 1960 IFCHARA%=127THEN2040
 1970 IFLEN(NUM$)>=MAXL%THEN1940
 1980 IFCHARA%<48ORCHARA%>57THEN1940
 1990 PRINTCHARA$;:NUM$=NUM$+CHARA$:GO
TO1940
 2000 REM===CHECK FOR CORRECT INPUT
 2010 IFLEN(NUM$)<MINL%ORLEN(NUM$)>MAX
L%THEN1930
 2020 IFVAL(NUM$)<MINV%ORVAL(NUM$)>MAX
V%THEN1930
 2030 ENDPROC
 2040 REM===DELETE A CHARACTER?
 2050 IFLEN(NUM$)=0THEN1940
 2060 VDU8,32,32,8,8:NUM$=LEFT$(NUM$,L
EN(NUM$)-1):GOTO1940
 3000 CLS:*FX 4,0
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

COLOURED TEXT AND LISTINGS IN MODE 7

It is not normally possible to have coloured text or background whilst editing a new program in mode 7, however this short program from Simon Wilkinson allows this to be done.

The scrolling area of the screen is reduced by three character positions at the left hand edge which are used to store the codes for the teletext colours. Coloured text on a black background may be obtained with the loss of only two columns.

```
10 MODE 7
20 N = 132      :REM Blue text
30 B = 134      :REM Cyan background
40 CLS
50 FOR Y = 0 TO 23
60 VDU B,157,N,13,10
70 NEXT Y
80 VDU 28,3,23,39,0
```

Program tested on 0·1 and 1·1 O.S.

ALIENS (16k/32k)

by Ben Heller

The ALIEN-ATTACK program is a game which has been around for quite a long time now. This version has been specially written for the BBC micro.

The object of the game is to shoot the aliens before they land on Earth. They fall from the sky, first one at a time, then several at once. You have to be very fast and very accurate in controlling your gun base, because you will only have time to fire once at each alien. Both the current score and high-score are displayed continuously at the top of the screen, so make sure that you have the *TV command set so that you can see the scores. (See BEEBUG issue 3 p16).

As it stands the game will fit into an unexpanded model A providing you delete the instructions in lines 950-1080 inclusive and the REMs in lines 10-40 inclusive. The program is reasonably structured, with a short loop of PROC calls forming the main program, followed by each of the procedure definitions. It uses only two colours, chosen so that it is easily visible in colour or monochrome. Speed has been given precedence, so any alterations should be made with this fact in mind.

HINTS:- the main problem in entering the program will be getting the character definitions EXACTLY correct. The listing has been made in such a way that no number is split over lines, a comma between numbers is always the last item on the end of a partial line of strings of data. Ensure tht you count the number of values after every VDU 23 command; failure to have exactly nine values following will give very unpredictable results. Any printed spaces between quote marks which are critical have been changed to SPC(n) commands, any remaining spaces are non-critical. Finally, to satisfy the many requests, the listing has been renumbered to enable it to be entered with the AUTO command. The listing is taken from the working program.

Main Procedures and their Function

PROCEDURE NAME	USE OF PROCEDURE
INIT	Initialises the variables, defines the characters and envelopes.
INSTRUCT	Displays the titles and playing instructions before the start of each game.
BASE	Test the keyboard (with negative INKEY) for the cursor left or right keys being pressed. If pressed, the firing base will be moved accordingly.
WAIT	A simple time delay of fixed duration.
ALIEN	Moves the alien one position down the screen. It also tests to see if a bullet has hit an alien, or if an alien has reached the bottom of the screen.
BULLET	Moves the single bullet up the screen, no test is made to see if it hits anything since this is part of the ALIEN procedure. If the bullet has reached the top of the screen then it is cancelled.
SHOT	A routine which is called when the ALIEN procedure detects that an alien has been shot. It increases the score and cancels the alien.
DEAD	A routine called when an alien reaches the Earth. A question is issued for another game, and the variables are reset for the next game.

Major Variables

VARIABLE	USE OF VARIABLE
BX%	X coordinate of the firing base.
XB%	X coordinate of the bullet.
YB%	Y coordinate of the bullet.
SC%	Current game score.
HS%	Current high-score.
AX%()	Array containing X-coordinate of the aliens.
AY%()	Array conatining Y-coordinate of the aliens.

[NOTE: long variable names were not used in order to save space].

```
  10 REM      'ALIENS' PROGRAM
  20 REM            32k
  30 REM BY BEN HELLER, CLEVELAND.
  40 REM
  50 MODE4
  60 VDU 19,1,3;0;
  70 VDU 19,0,4;0;
  80 DIMAX%(10),AY%(10)
  90 PROCINIT
 100 PROCINSTRUCT
 110 PRINTTAB(6,0)"0000"
 120 PRINTTAB(20,0)"0000"
 130 VDU 23;8202;0;0;0;
 140 PROCBASE
 150 PROCWAIT(100-SC%*2)
 160 PROCALIEN
 170 PROCBULLET
 180 PROCBASE
 190 PROCBULLET
 200 PROCBASE
 210 GOTO140
 220 DEFPROCINIT
 230 BX%=19:SC%=0:HS%=0:YB%=0:XB%=-1
 240 FORCO%=1TO10
 250 AX%(CO%)=1+RND(36):AY%(CO%)=3
 260 NEXT
 270 AY%(1)=-20
 280 VDU 23,224,0,0,0,0,127,127,127,
127
 290 VDU 23,225,16,56,124,254,255,
255,255,255
 300 VDU 23,226,0,0,0,0,252,252,252,
252
 310 VDU 23,227,16,16,16,16,16,56,
16,56
 320 VDU 23,228,1,3,7,15,255,255,
255,240
 330 VDU 23,229,255,255,255,255,255,
255,255,255
 340 VDU 23,230,128,192,224,240,255,
255,255,15
 350 VDU 23,231,240,255,255,255,15,
7,3,1
 360 VDU 23,232,255,255,255,255,195,
255,255,255
 370 VDU 23,233,15,255,255,255,240,
224,192,128
 380  ENVELOPE 2,1,-2,0,0,100,0,0,
0,0,0,0,0,0 :REM 8 ZEROS
 390  ENVELOPE 1,2,0,0,0,0,0,0,9,0,
-2,-2,120,80
 400  ENVELOPE 3,129,2,4,6,28,14,7,
0,0,0,-80,80,80
 410 ENDPROC
 420 DEFPROCBASE
 430 B1%=BX%
 440 IFINKEY(-26)ANDBX%>1 BX%=BX%-1
 450 IFINKEY(-122)ANDBX%<37BX%=BX%+1
 460 PRINTTAB(B1%-1,31);SPC(3);
 470 PRINTTAB(BX%-1,31);:VDU224,225,
226
 480 ENDPROC
 490 DEFPROCBULLET
 500 IFYB%=0AND NOTINKEY(-1) ENDPROC
 510 IF YB%=0 XB%=BX%:SOUND 0,1,7,5:
SOUND 1,2,20,9
 520 PRINTTAB(XB%+1,30-YB%);:VDU127,
11,227
 530 IFYB%=28 THENYB%=0:VDU127,11 :
ENDPROC
 540 YB%=YB%+1
 550 ENDPROC
 560 DEFPROCALIEN
 570 SOUND2,-8,((SC%+1)DIV4)MOD255,1
 580 FORN1%=1TOSC%/10+1
 590 IF ABS(XB%-AX%(N1%)) <2 ANDABS(
```

```
30-YB%-AY%(N1%)) <2 PROCSHOT
  600 IFAY%(N1%)<3THEN630
  610 PRINTTAB(AX%(N1%)-1,AY%(N1%));:
VDU 231,232,233,11,8,8,8,228,229,230,
11,127,127,127
  620 IFAY%(N1%)>30 PROCDEAD
  630 AY%(N1%)=AY%(N1%)+1
  640 NEXT
  650 ENDPROC
  660 DEFPROCSHOT
  670 PRINTTAB(AX%(N1%)+2,AY%(N1%)-1)
;:VDU127,127,127,11,9,9,9,127,127,127
  680 SC%=SC%+1
  690 AY%(N1%)=-RND(50) : AX%(N1%) =
RND(36)+1
  700 PRINT TAB(XB%,30-YB%);SPC(1) :
YB%=0:XB=-1
  710 SOUND3,3,50,10
  720 PRINTTAB(8,0);SC%;"0"
  730 IFSC%>HS% HS%=SC% :PRINTTAB(22,
0);HS%;"0"
  740 ENDPROC
  750 DEFPROCWAIT(WT%)
  760 FORWA%=1TOWT%:NEXT
  770 ENDPROC
  780 DEFPROCDEAD
  790 SOUND 0,-15,7,25
  800 FORC1%=150 TO 1 STEP-5 :SOUND1,
-10,C1%,1:NEXT
  810 PRINTTAB(9,15);"Another game ?"
  820 *FX15,1
  830 A1$=GET$
  840 IFA1$="N"ORA1$="n" CLS:END
  850 CLS
  860 PROCINSTRUCT
  870 PRINTTAB(6,0);"0000";TAB(20,0);
"00";HS%;"0"
  880 BX%=19 :YB%=0 :XB%=-1 :SC%=0 :
N1%=10
  890 FORCO%=1TO10
  900 AX%(CO%)=1+RND(36):AY%(CO%)=3
  910 NEXT
  920 AY%(1)=-20
  930 ENDPROC
  940 DEFPROCINSTRUCT
  950 PRINT ' TAB(17,5)"ALIEN"
  960 PRINTTAB(1,10)"The Earth is";
  970 PRINT" being invaded by aliens"
  980 PRINT'" and only you can stop";
  990 PRINT" them by using"
 1000 PRINT'" The Laser Cannon."
 1010 PRINT''''" To    Fire   press";
 1020 PRINT"...<SHIFT>"
 1030 PRINT'" Use the cursor keys";
 1040 PRINT" to move left/right"
 1050 PRINT''"PRESS THE SPACE BAR";
 1060 PRINT" TO START..."
 1070 *FX15,1
 1080 A$=GET$ : CLS
 1090 ENDPROC
```

DISCOUNTS

BEEBUG have arranged discounts for members at a number of retail outlets who supply computer books, software, hardware and services. We are continually negotiating further discounts.

H. Banton 8 Princess Road Urmston Manchester M31 3SS Tel:061 747 7014	Printed cards to label the user-definable keys. £1.80/10,£2.40/20 £4.60/50 10% member discount.	Opus Supplies Birchtrees 10 Beckenham Grove Shortlands Bromley BR2 0JU	Computer furniture 20% discount. Send for brochure.
Electronics Applied 4 Dromore Road Carrickfergus Co Antrim BT38 7PJ (Tel: 65973)	Cassette leads for the BBC machine 5% discount	Technomatics Ltd 15 Burnley Road LONDON NW10 (Tel: 01-452 1500)	Hardware, components and connectors 5% discount.
Happy Memories Gladestry Kington Herefordshire HR5 3NY (Tel: 054422 618)	Computer Hardware 10% off all 'one-off' prices. Quantity prices may be negotiable.	Wallace J & S 9 Barn Close Crewkerne TA18 8BL	Flexible plastic dust covers. 3.95. 25% discount to members.
Mine of Information (Mail Order) 1 Francis Avenue St Albans Herts (Tel: 0727 52801)	Computer Books 5% discount	Watford Electronics 33, Cardiff Road Watford Herts. (Tel: Watford 40588)	5% off all items

It is advisable to telephone before placing an order, to check availability. Members should simply quote their membership number with their order, though members taking discount will not necessarily be given credit card facilities, (you must check this). We are not acting for these companies, nor receiving payment from them, and cannot be held responsible for their services.

BEEBUGSOFT

BEEBUG SOFTWARE LIBRARY

Next month we will be bringing out 4 new cassettes, including a very impressive "Asteroids" type of game in machine code, and a comprehensive cassette-based file management package to handle data bases of all kinds.

GAMES 1 £3.50

STARFIRE (32k) This is a very well written star-wars type game, with first class sound and graphics. The object is to get the enemy ships within your sights and destroy them with your lasers before they use up too much of your protective force field. You should try this at warp 3!

The improvements to Starfire given in this magazine are incorporated in the cassette as from 20th September.

GAMES 2 £3.50

MOON LANDER (16k) Land the module in the crater with a speed less than 15km/hr. The program has fuel and speed screen readouts, with left-right and vertical thrust controls. The craft leaves a vapour(!) trail so that you can see your trajectory. There are two degrees of difficulty.

3D NOUGHTS & CROSSES (32k) 3 dimensional noughts and crosses played on a 4 x 4 x 4 board. It has a colour 3 dimensional graphics display, and the machine plays a pretty good game.

GAMES 3 £3.50

SHAPE MATCH (16k) A two player game with nice visuals - a joint pattern recognition and memory test. Remember the positions of the shapes and match them up to win.

MINDBENDER (16k) A Beeb implementation of a "Mastermind" type game ("Mastermind" is a trademark of Invicta Plastics). You can choose the number of columns (1-10), and the machine assigns any of 7 coloured letters to each column. You make your guesses, and the machine tells you your score each time. There is a special facility to give the same combination to two or more players, so that the game can be used competitively in a completely fair way. This is a very good test of your logical powers, especially with 10 columns.

GAMES 4 £3.50

MAGIC EEL (32k) Excellent fast moving arcade-type game. Guide the Eel around the screen gobbling up everything it comes across, it gets longer and faster with every bite. When you have eaten through the first frame, this is replaced with harder ones - faster with more obstacles. Fast and addictive - don't be put off by the low price.

UTILITIES 1 - £4.00

DISASSEMBLER (16k) Read the machines ROMs (and EPROMs) with this nicely written disassembler. It gives you the full 6502 mnemonics. There is also a column with ASCII codes of each byte, allowing you to spot embedded text immediately. To use it just enter the start address, and page mode is engaged, displaying 20 lines or so at each press of the space bar.

REDEFINE (16k) A very useful graphics tool. Redefine allows you to build up user defined graphics characters. You enter points on an 8 x 8 grid using the cursor control keys, the program constantly displays the character so you can see it build up, and also the VDU 23 command that you need to use to create that character in your program. The program will literally save you hours.

MINI TEXT ED (32k) A mini word processor substitute. It uses the machine's Basic editing facilities to allow text to be entered, edited, saved, loaded, and printed to the screen or printer. The printer routine contains a subroutine to remove the Basic line numbers. Note that this will not, as it stands, right-justify, and will not automatically close up the text after editing. It nevertheless provides a very useful facility.

APPLICATIONS 1 - £5.20

SUPERPLOT (32k) Produces tailored screen representations of any function entered. This can be achieved in any of the three major coordinate systems: Cartesian, Polar, or Parametric. SUPERPLOT comes complete with a 7-page instruction booklet. Explore the world of graphic representation.

All prices now include VAT. Please add 50p postage and packing to all orders regardless of size. Post to BEEBUG Software, 374 Wandsworth Road, London SW8 4TE. The above postal rates are for the UK only. Overseas enquiries welcome.

IF YOU WRITE TO US

Back Issues (Members only)

All back issues are kept in print (from April 1982). Send 80p per issue PLUS an A5 SAE to the subscriptions address. This offer is for members only, so it is essential to quote your membership number with your order.

Subscriptions Address

BEEBUG
Dept 1
374 Wandsworth Rd
London
SW8 4TE

Subscriptions

Send all applications for membership, subscription renewals, and subscription queries to the subscriptions address.
Membership costs: £4.90 for 6 months (5 issues), £8.90 for 1 year (10 issues)
European membership £15 for 1 year.
Elsewhere - Postal Zone 1 £18, Zone 2 £20, Zone 3 £22

Software (Members only)

See details overleaf. Available from the subscriptions address.

Contributions and Technical Enquiries

Please send all editorial material to the editorial address opposite. If you require a reply it is essential to quote your membership number and enclose an SAE.

Editorial Address

BEEBUG
PO Box 50
St Albans
Herts
AL1 2AR

Ideas, Hints & Tips, Programs, and Longer Articles

Substantial articles are particularly welcome and we will pay for these! But in this case, please give us warning of anything that you intend to write. In the case of material longer than a page, we would prefer this to be submitted on cassette in a format similar to that produced by our Mini Text editor (June 82 newsletter or Utilities 1 cassette) or as a Beeb datafile providing that you use the bugs fix on page 21 of the July issue, or you have the 1.0 (or later) operating system.

In Next Month's Issue

A new upper case character set in software for the Beeb, giving 26 characters per line (instead of 20) in modes 2 and 5; a Sound and Music feature; Joystick review, including programming ideas; an article on Sort Routines in BBC Basic; how to add the serial port; Mini Text Editor with numberless printout routine - plus, we hope - Acorn's further thoughts on the ROM charge.

BEEBUG NEWSLETTER is edited and produced by Sheridan Williams and Dr David Graham.
Production editor Phyllida Vanstone
Thanks are due to Rob Pickering, and John Yale for assistance with this issue.

Printed by Heathprint (St Albans) Limited
ISSN 0263-7561