INDEPENDENT NATIONAL USER GROUP
FOR THE BBC MICROCOMPUTER

BEEBUG

VOLUME 1 NUMBER 9 FEB 1983

CONTENTS

GENERAL CONTENTS

Editorial 2
Ideas or Animation 3
Some Notes On Program Saving 5
Quick Call 7
Machine Code Screen Dumps for the Epson and Seikosha Printers 8
Spotlight 11
Using Files 12
Book Reviews 15
BBC/O Software Competition Results 16
Disc Rewinding 18
BBC Basics - Procedures 21
Software Update 24
Program Compactor Revisited (16k) 25
Single Key Memory Display (16k) 27
Reading Five Discs (16k) 28
Fainte Ailing 31
Geomancy (16k) 32
Hardware Offer 36
1-2 O.F. ordering Details 37
Notice Board 38

PROGRAMS

Master Study 5
Rotating Cube 6
Seikosha Screen Dump 9
Epson Screen Dump 10
Spirograph 14
Windy Field 15
Compacter Revisited (16k) 25
Single Key Memory Display (16k) 27
Five Discs (16k) 29
Geomancy (32k) 32

HINTS, TIPS & INFO

System Calls 7
*FX200 Call 11
Sideways Scroll 14
*FX18 Call 28
Defining Keys 28
Last Line Detection 28
ON ERROR Bug 28
More Joysticks 31
Simple Musical Keyboard 26
Worn Cassette Load Problems 28
Retrieving Listing (Telewriter) 31
Moving the Graphics Origin 31
Disc Cassette Break 31

BRITAINS LARGEST MICRO USER GROUP

MEMBERSHIP NOW EXCEEDS 14,000

EDITORIAL

1.2 OPERATING SYSTEM ROM

As we have been saying for some time, we have been in negotiation with Acorn over the pricing policy for the replacement operating system ROM for the Beeb. Acorn's projected replacement cost started out at around £15, then rose to around £25, and perhaps as a planned concession in the pouring of proposal, dropped again to £18 + VAT (=£21.60). This is the current price for a dealer upgrade, though if your current operating system is in fact only the dealer replacement will be performed free of charge.

As a result of our discussions, Acorn have agreed to let BEEBUG members have a replacement ROM at around half price. The ROM must be ordered direct from BEEBUG on our new software distribution scheme in high sycophancy and costs £5.10 + VAT (=£5.87). We are reasonably satisfied with this price, although it approaches that might be termed a "nominal" replacement charge. Though £3.00 would have been nicer!). Moreover, the 1.2 is really rather a good piece of firmware. We have had advance samples of the ROM for a number of weeks now, and the Beeb one certainly has a useful number of features not expected from the original specification machine. For example extra graphics "fill" routines (see listing 6) and the many extra *FX calls - see the new User Guide for details.

The ROM will be supplied with simple fitting instructions. No soldering is required, but you will need to remove the keyboard (2 or 3 screws). Full details of how to order the new ROM are given on page 31 of this issue, together with an official statement from Acorn. Very large numbers of ROMs have been set aside for us over the next two months, and we are very surprised at the same time as the dealer network. Nevertheless, supply can never be guaranteed, and for this reason we are limiting the offer to one per member/ship in the first instance.

BEEBUG MAGAZINE

As you will see, this issue is in two parts. The main body of the magazine is accompanied by a twenty page supplement (unless ordered as a back copy). This supplement contains advertising plus a series of items that will in general be updated monthly. We have received numerous requests for advertising space in the magazine, and it was this which prompted us to produce a trade supplement. There is now a vast array of products available for the BBC micro, and we hope that the supplement will provide a useful function in informing readers of an interesting cross-section of products.

On the other hand we have kept this separate from the main body of the magazine in this trial format, so that the latter may be kept active as an unencumbered reference book. To this end we are currently investigating a binder offer, and will be producing an index to volume one in the April 1983 issue.

We hope that you will find the supplement useful, and you may wish to use our new Members' Corner page, or Personal Ads column in the future. In any case this issue contains considerably more editorial material than ever the last issue - since advertisements have been removed from the main body of the magazine, as have regular items such as discounts pages, user groups and so on.

In this issue, we give the results of our second competition - with prizes totalling over £400. At the same time - on page 28 of the supplement - a new competition is announced with prizes of a Microvitec colour VDU monitor, five Wordwise word processors and five copies of Acornsoft's Creative Graphics package. We are most grateful to Torterm Ltd., Computer Concepts, and Acornsoft for the donation of these prizes.

by David Graham

Program tested on 0-1 and 1-1 O.S.

IDEAS ON ANIMATION

by Colin Opie

This article describes how to produce remarkably smooth animation with BBC graphics, and is based on a clever technique used in a rotating cube program written by Roger Wilson and Laurence Hardwick of Acorn.

The first part of the article introduces the concept of logical plotting using GCOL and VDU19. The technique is then used to produce an animation sequence; and finally a listing of the rotating cube program is given. Note that if you are using a 16K machine, you should change Line 100 in fig.3 and Line 120 in fig 5 to MODE 5. Even if you have trouble in completely following the theory, the programs are well worth a look, and the procedures employed can be adapted for other uses.

LOGICAL AND PHYSICAL COLOURS

A concept which is inherent in the use of GCOL and VDU19 is that of the difference between a physical colour on the screen (e.g. red, yellow) and a logical colour (ie. the number code which the computer uses internally and is willing to associate with a physical colour). If you look in the New User Guide (p223-on COLOUR) you will see that the maximum number of physical colours available in the graphics modes is sixteen, which includes 'flashing' combinations. Different internal logical colour codes are used for foreground and background colours enabling us to determine precisely what will happen when a character (graphics or text) is printed.

By way of example we will look at the options open to us when using screen mode 5. In this mode we are only allowed a maximum of four different colours on the screen at the same time. If we don't do anything to change the 'status quo' of this mode we will get the options given in fig 1. Note that the values 0-3 and 128-131 are the internal logical colour values used by the computer. Now lets take a look at GCOL and VDU19 to see how we can manipulate these facts to our advantage.

Foreground	Background	Physical Colour
0	128	black
1	129	red
2	130	yellow
3	131	white

fig. 1 logical colour values for mode 5.

GCOL AND VDU19

The basis for this article lies in these two commands, so it is worth taking a good look at these. GCOL enables us to say how we want all future graphics to be plotted and which colour to use. It has two arguments and has the general form:
GCOL P,C ; where P is the plotting method (in the range 0-4) which is to be used, and C is the value of the logical colour to use for all future graphics work. Clearly in mode 5 any of the values given in fig. 1 (above) would be valid. The parameter 'P' can be as given in fig 2, and the position of the plotting operation will be determined by the PLOT and TAB commands. This latest table produces at least three new operations which we need to know about, those of OR, AND, and EOR plotting. We will take each of these in turn and concern ourselves, for the sake of simplicity, with just foreground colours.

P	Plotting Method
0	plot the colour
1	OR the colour with current screen value
2	AND the colour with current screen value
3	EOR the colour with current screen value
4	invert the colour currently on the screen

Fig. 2 methods of graphic plotting

P=0: In this mode we will simply plot the colour specified by GCOL. If the command GCOL0,1 is used then the future foreground colour will be in red. If we use GCOL0,2 then it will be yellow. It does not matter what colour was present on the screen at some position prior to another plot in the same position. The new colour will take over.

P=1: Here it is the logical OR of the new colour and the previous colour which will be used in the plotting operation. Type in the following program:

```
10  MODE 5: GCOL0,1
20  PLOT 5, 300,300
30  GCOL 1,2: DUMMY=INKEY(100)
40  PLOT 5,0,0
50  END
```

First of all you will see a red line -i.e. COLOUR 1- (because of the GCOL0,1 in line 10), and then one second later you will get a white line, though one might at first sight expect a yellow line (ie. colour 2). Initially each point on the line was at logical colour 1 (RED). We then set the logical colour to 2 (YELLOW), but with OR plotting; and "1 OR 2" has the logic value 3, (you can verify this by typing "PRINT 1 OR 2" in Basic). This is why we get a white line (logical colour 3) instead of a yellow one. (If this logical binary operation is new to you then see the articles on LOGIC in BEEBUG vol. 1 nos.5,6 (Sept & Oct)).

P=2: This will use the logical AND of the colours. If we change line 30 (above) to read: 30 GCOL2,2: DUMMY=INKEY (100)
we can see the effect of this. Colour 1 (RED) AND Colour 2 (YELLOW) gives Colour 0 (BLACK) because "1 AND 2" has the logic value zero, (again see BEEBUG nos. 5 and 6 if this is unclear). Thus after one second is up the line disappears even though our specified foreground colour is yellow.

P=3: Using EOR plotting (and therefore a value of GCOL3,2) we will get the same result as with P=1 because "1 EOR 2" has the logical value 3. (To test this type "PRINT 1 EOR 2" in Basic).

P=4: This ignores the current colour setting and merely inverts the colour presently set. The inverse (or logical NOT) of colour 1(RED) is colour 2, hence we get a yellow line.

Now let's take a look at the VDU19 command. This has the general form of:

VDU19,L,A;0;

where L is the logical colour we are referring to, and A is the code for the actual colour we wish to assign to it. Fig.1 shows the values set for screen mode 5 if we don't do anything to change it. Even though we are only allowed four different colours at any one time, we can, via the VDU19 command, say which of the full range of 16 physical colours we wish to use for each permitted logical colour.
For example: 12 VDU19,1,4;0; will set colour 1 to BLUE
14 VDU19,3,13;0; will set colour 3 to flashing MAGENTA/GREEN.
If you insert these two lines into your test program and reset GCOL in line 30 to GCOL0,2 you will see the effect of this. The interesting thing about the VDU19 call to alter physical screen colours is that it does so instantly. We can take advantage of this in a number of ways. In particular we can draw a new image in physical colour black, and then make it suddenly appear all at once, just by using an appropriate call to VDU19.

ANIMATION TECHNIQUES APPLIED

As an example of how these two commands can be used effectively we will take a look at a couple of programs for producing animation, and compare their effects. Type in the program in fig. 3 using the line numbers given. The animation is done by merely drawing the boat in a colour and then drawing it again in black to erase it, and so on. If you run this program you will notice that the picture flickers furiously and does not look very nice at all.

PROCSAIL is the procedure which effectively does all the work. If we change this procedure to use GCOL more effectively, we can get a smoother display. Change the PROCSAIL procedure and add in the PROCSHOW procedure as given in fig 4. To try and see what is happening consider the following.

The way this new technique works is to draw the new ship each time in physical

colour black, so that the jerky drawing process is invisible. We then suddenly make it appear by using the appropriate VDU19 call. You may have used the same idea when drawing a graphics screen for a game etc. - to prevent the user from seeing the drawing process. In broad terms the same technique applies here, but there is a complication - each of the new ships overlaps the previous one, so we actually use three different logical colours. Colour 1 is used for the first display and colour 2 for the second. On the third display we want to erase the display drawn in colour 1 but not the one in colour 2. This can be achieved by an EOR operation using the value we want to erase, ie. colour 1. Notice that this will give us (1 EOR 1) for the first display (ie. 0 = black) which is what we want. In the second display on the overlapped bits we get (3 EOR 1) which gives us colour 2, which is also what we want. Moreover the whole of the second display is now in colour 2. By 'showing' the relevant colour at each stage the shape can be made to move, hence our extra procedure PROCSHOW.

**Note* To use two colours for two 'displays' we actually require three logical colours, hence we lose the ability to have so many different colours on the screen at the same time. So in using the technique we gain in appearance but lose a little in multi-colour capability. Obviously this poses far fewer problems in MODE 2 which can display up to 16 physical colours all at the same time.

ROTATING CUBES

We are most grateful to Roger Wilson and Laurence Hardwick of Acorn for permission to publish the program which appears as fig 5. This uses exactly the same technique to generate a rotating cube. It is the main loop in lines 360-510 which is doing the actual work, and notice how it too needs to access a PROCSHOW procedure in order to highlight only the logical colours that are pertinent at any one time. We will be looking further at 3-D graphics and rotation next month. Meanwhile you can type in the program and sit back and watch. In case you want to experiment, here is a list of the main procedures and variables:

PROCEDURES (CUBE)

D,DRAW Draws the 3-D shape. This uses GCOL to set appropriate logical operations in order to create and erase each view.

MOVE Calculates the 2-D mapping coordinates for each new position, from the stored 3-D coordinates.

SHOW Enables given colours to be displayed, hence showing a particular view.

VARIABLES (CUBE)

TX,TY,TZ Linear transformations in the three planes. If they are all zero then the cube remains in the same position, though rotating.

L,M,N,F Affect the displayed 'shape' of the cuboid. The values given display (would you believe) a cube.

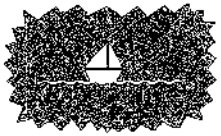
NN,TIME These enable a display to be given in the top left hand corner of the time (1/100 sec) for each move. Notice that this is about a second.

COLOU% Defines the colour of the rotating cuboid.

CX,CY,CZ These are the rotation factors in the three planes. If they were made to be all zero then the cuboid would not rotate.

Fig 3 "SHIP1"

```
100  MODE 1
110  VDU19,3,6;0;
120  GCOL0,3:PROCSEA
130  PROCSAIL
140  VDU19,3,7;0;
150  END
160  DEF PROCSAIL
170  GCOL 0,2
180  FOR X=0 TO 1279 STEP 4
190  PROCDRAW(X,105,5)
200  PROCDRAW(X,105,7)
210  NEXT X
220  ENDPROC
340  DEF PROCDRAW(X%,Y%,C%)
350  PLOT 4,X%,Y%
360  PLOT 4,X%+80,Y%
370  PLOT 80+C%,X%+80,Y%+80
380  PLOT 4,X%+88,Y%
```



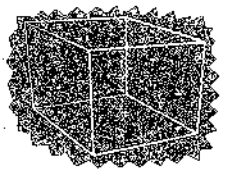
```
390  PLOT 4,X%+88,Y%+80
400  PLOT 80+C%,X%+128,Y%
410  PLOT 4,X%,Y%-8
420  PLOT 4,X%+128,Y%-8
430  PLOT 80+C%,X%+20,Y%-48
440  PLOT 4,X%+24,Y%-48
450  PLOT 4,X%+132,Y%-8
460  PLOT 80+C%,X%+108,Y%-48
470  ENDPROC
480  DEF PROCSEA
490  FOR X=0 TO 1279 STEP 4
500  Y=50+5*SIN(X/10)
510  PLOT 69,X,Y
520  NEXT X
530  ENDPROC
```

Fig 4 Additions for "SHIP2"

```
160 DEF PROCSAIL
170 VDU19,3,3;0;
180 FOR X=0 TO 1279 STEP 16
190 GCOL1,1
200 PROCDRAW(X,105,5)
210 PROCSHOW(1)
220 GCOL1,2
230 PROCDRAW(X+8,105,5)
240 PROCSHOW(2)
250 GCOL3,1
260 PROCDRAW(X,105,5)
270 GCOL1,1
280 PROCDRAW(X+16,105,5)
290 PROCSHOW(1)
300 GCOL3,2
310 PROCDRAW(X+8,105,5)
320 NEXT X
330 ENDPROC
540 DEF PROCSHOW(S%)
550 P%=3:REM YELLOW
560 VDU19,S%,P%;0;
570 VDU19,3-S%,0;0;
580 ENDPROC
```

Fig 5 "ROTATING CUBE"

```
100 REM**INIT SCREEN AND VARIABLES
110 NN=TIME
120 MODE1
130 VDU29,640;512;
140 TX=0:TY=0:TZ=0
150 CX=-.025
160 CY=0.025
170 CZ=0.025
180 REM**SET STORAGE ARRAYS
190 DIM B(7,2,1)
200 DIM B%(7,1,1)
210 REM**READ IN START POSITION
220 FOR P%=0 TO 7
230 FOR C%=0 TO 2
240 READ B(P%,C%,0)
250 NEXT
260 NEXT
270 DATA 30,30,30,-30,30,30,-30,-30,30,
30,-30,30
280 DATA 30,-30,-30,30,30,-30,-30,30,-30,
-30,-30,-30
290 REM**SET UP PROCMOVE CONSTANTS
300 F=4
310 L=130:M=140:N=80
320 S=L*L+M*M
330 R=SQRS
340 R=R/2
350 T=S+N*N:Q=SQRT*3
360 REM**MAIN LOOP
370 REPEAT
380 GCOL1,2
390 PROCDRAW(0)
400 PROCSHOW(2)
410 GCOL2,2
420 PROCDRAW(1)
430 PROCMOVE(0,1)
440 GCOL1,1
450 PROCDRAW(1)
460 PROCSHOW(1)
470 GCOL2,1
480 PROCDRAW(0)
490 PROCMOVE(1,0)
500 UNTIL INKEY 0<>-1
510 END
520 REM**PROCEDURE DECLARATIONS
530 DEF PROCD(P%,N%,A%)
540 PLOTA%,B%(P%,0,N%),B%(P%,1,N%):ENDPROC
550 DEF PROCDRAW(N%)
560 PROCD(0,N%,4)
570 FOR P%=1 TO 7:PROCD(P%,N%,5):NEXT
590 PROCD(2,N%,5)
600 PROCD(3,N%,4)
610 PROCD(0,N%,5)
620 PROCD(5,N%,5)
630 PROCD(6,N%,4)
640 PROCD(1,N%,5)
650 PROCD(7,N%,4)
660 PROCD(4,N%,21)
670 ENDPROC
680 DEF PROCMOVE(A%,B%)
690 LOCAL P%:FOR P%=0 TO 7
700 B(P%,0,B%)=B(P%,0,A%)+CZ*B(P%,1,A%)
+CY*B(P%,2,A%)+TX
710 B(P%,1,B%)=B(P%,1,A%)+CX*B(P%,2,A%)
-CZ*B(P%,0,B%)+TY
720 B(P%,2,B%)=B(P%,2,A%)-CX*B(P%,1,B%)
-CY*B(P%,0,B%)+TZ
730 U=B(P%,0,B%):V=B(P%,1,B%):W=B(P%,2,B%)
740 t=R*(T-U*L-V*M-W*N)
750 B%(P%,0,B%)=T*(V*L-U*M)*F/t:B%(P%,1,
B%)=Q*(W*S -N*(U*L+V*M))/t
760 NEXT
770 VDU30:PRINT TIME-NN:NN=TIME
780 ENDPROC
790 DEF PROCSHOW(N%)
800 COLOU%=7
810 VDU19,N%,COLOU%;0;
820 VDU19,3-N%,0;0;
830 VDU19,3,COLOU%;0;
840 ENDPROC
```



SOME NOTES ON PROGRAM SAVING

by Andrew Donald

The programs on the Welcome tape are all copyable, using the SAVE command, with the single exception of PHOTO. Investigation of why this should be has led to some useful insights.

What happens is this. When loaded the program shows a length of Ø9F4 bytes. If this program is now SAVEed the length will be shown to be Ø39C bytes, Ø558 bytes are missing.

The missing bytes are additional code required for the image plotting in the program. The BASIC itself does end at Ø39C, and the SAVE command will only operate on the BASIC portion.

In order to save this program it is necessary to dump the BASIC and extra code as a unit and set the BASIC portion into operation when run. Looking at the memory map shows BASIC starts at ØE00 and we need to save Ø9F4 bytes from here. This may be done with the command *SAVE "PHOTO" ØE00 +09F4 The default starting address for this code will be ØE00. That puts the total code onto the tape but how do we operate it?

The User Guide says to load a block of machine code and run it we should use the *LOAD or *RUN commands. However, since the bottom of this code is BASIC and since the tape format for both types of dump is acceptable to the BASIC loader, the command CHAIN"PHOTO" will load the entire block and then execute the bottom end BASIC. This is how the program operates on the Welcome tape.

The consequence of this facility is that BASIC and machine code, and/or data may be mixed in a more compact form than is possible with the usual mnemonic assembler allowed for in the interpreter. USR routines may be written above the BASIC: but take care to set LOMEM to prevent corruption of machine code when BASIC variables are assigned.

Also a machine code program may be kept in standard BASIC format by including a single BASIC line] CALL <address> and dumping with *SAVE from ØE00 to the end of the code.
[We suspect that the extra code that needs to be saved is not actually a machine code PROGRAM but DATA. However the same still applies for saving. Ed.]

QUICK QUIZ

QUICK QUIZ

What numbers between zero and 999 have the special property that they are the sum of the cubes of their digits?

For example 234 is NOT a solution because 2 cubed + 3 cubed + 4 cubed does not equal 234.

The object of the quiz is to produce the fastest BASIC program to achieve this (checking all the way from zero to 999).

Please do not send in your results, we will publish our program and best times next month.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SYSTEM CALLS

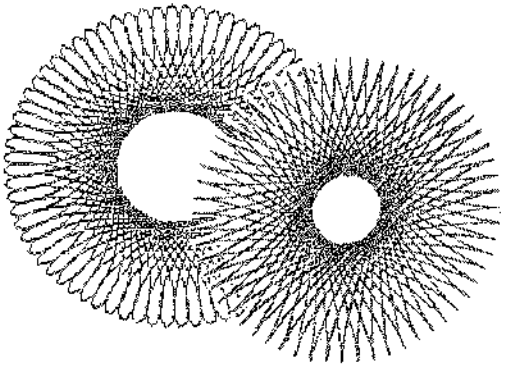
Did you know that all commands preceded by '*' (eg. system calls such as *FX, *KEY, *CAT etc.) can be in lower case? Andrew Crawhurst. Note also the abbreviation "*." for "*CAT".

MACHINE CODE SCREEN DUMPS FOR THE EPSON & SEIKOSHA

Here at last are the promised machine code screen dumps for the Seikosha and Epson printers. They are many times faster than the Basic dumps published earlier. Remember to take great care in entering the code, because machine code is much much harder to debug than Basic. Note that the Seikosha dump will not, as far as we know, work on a Seikosha 250.

SEIKOSHA SCREEN DUMP (GP80A and GP100A) by Terry Bromilow

The program to be described is a machine code dump for the Seikosha GP80A and GP100A which will print in any graphics mode of the BBC machine. The program is most welcome as it enables you to print any horizontal strip of the screen, and takes only about 2.5 minutes to print a full screen. An additional feature is that if the screen to be drawn consists of several colours, it is possible to specify that only items of a certain colour should be printed. (Obviously the printer will print everything the same colour). When using mode 0, with its 640 dots horizontally, a compromise has had to be reached, as the Seikosha will print only 480. The solution adopted is to print every other dot.



ABOUT THE PROGRAM

The program consists mainly of data statements, which are converted into machine code and cunningly stored in line 29480. That is why line 29480 consists, as it is typed in, of only X s. It is therefore essential that this line is typed in carefully as listed. Do not type in too few X s, there should be 168, although extras are okay.

Included in the program is a demonstration print of the exponential decay of a sine wave. It is by running this demo program that the machine code is generated, so do copy all of the listing, not just part of it. Once the code has been generated, the rest of the program can be deleted, leaving a very small and efficient machine code screen dump, which is activated as a procedure.

Storing the code in line 29480 makes it very easy to append to other programs, but results in restrictions on the 6502 instruction codes used. The codes 01 and 06 could not appear in it and there could be no internal absolute addresses, to enable it to work from any location. Also the appending process shown below must be used, since *SPOOL will not work with this program. The advantages in implementing the machine code this way are that no dedicated RAM is used, and a fully relocatable program results which may be further edited after the appending process. The procedure uses zero page locations &84 and &8F but these are saved and restored by lines 29120 and 29460 which may be omitted if not required.

TO SET UP THE PROGRAM

The following needs only to be performed the first time that the program is used.

1) Type in the program as listed with great attention to the data statements, and save a listing of the full program to tape or disc.

2) Ensure that your printer is connected to the computer.

3) Now run the program. It should immediately prompt you for a mode. Select any mode (0,1,2,4 or 5) and enter the apropriate number followed by <return>.

4) The demonstration display should now commence. An exponential decay graph will be drawn on the screen, and then on the printer.

5) When the printer has finished, delete the following lines from the program;
Delete lines 29000 to 29096 inclusive
Delete lines 29130 to 29410 inclusive

6) Now save the resultant machine code, which is stored in the procedure called PROCGPR. So use SAVE "GPR" (not *SAVE).

HOW TO USE THE NEW PROGRAM

The screen dump is now in the form of a procedure, so to activate it we need to simply call it from the program that has produced the screen that is to be printed. This is done as follows;

1) Load in the program that will create the screen to be printed. This is accomplished in the normal way, using LOAD"progname".

2) Check that all the line numbers of this program are below 29000. If they are not just issue the RENUMBER keyword.

3) Now type in PRINT ~ TOP-2 <return>.
This will print a three or four character number on the screen. I will for the moment call this number XXXX.

4) Now reload the procedure created earlier using the following;
*LOAD"GPR" XXXX <return>
where XXXX is the above mentioned number.

5) Now type END <return>

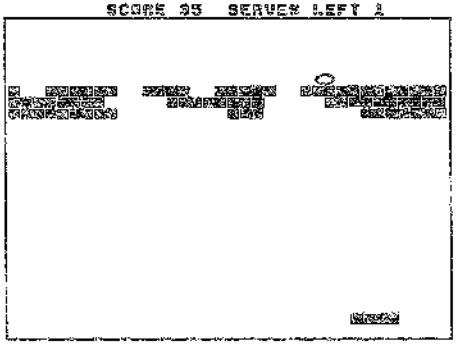
HOW TO CALL PROCGPR

If you now list the program, you will find that PROCGPR is located at the end of your original program, ready for use. It is ok to add new lines to the program or alter it in the normal way, without

Printout from RACER (Issue 7)

fear of damaging the machine code. It is totally relocatable. All that remains is to call PROCGPR from your program at whatever stage you want the screen image to be dumped to the printer.

It is possible that you may wish to decide while your program is actually running, at what stage you want the screen printed. This is very simple and below is listed one way of achieving this.
Insert at an early stage in your program;
Say line 5 5 ON ERROR GOTO 28900
And then also 28900 PROCGPR(1023,0,15) : END
Using these two lines, you could allow the screen to develop and then hit escape at an appropriate stage, whereupon the screen would immediately be printed. You can now save a copy of your program, including the PROCGPR within it. This may be done in the normal way.



Printout from BREAKOUT (issue 8)

PARAMETERS OF PROCGPR
The procedure must be called with 3 parameters. These are;

1. The upper Y coordinate of the strip to be printed,
2. The lower Y coordinate of the strip to be printed,
3. The colours to be printed.

The first two parameters are self explanatory, being the upper and lower limits of the strip to be printed. The coordinates must be given from the GRAPHICS origin, hence to print the whole screen the parameters would be 1023 and 0 respectively, and to print a narrow strip at the bottom of the screen would be 300 and 0. (see New User Guide for graphics details).

The third parameter determines which of the colours are to be printed. This number must be between 1 and 15. It will be ANDed with the logical colour of each point, and that point will only be printed if the result is non zero. To understand this very useful function, you must have an understanding of how AND is used as a bit operator. See New User Guide page 205 and previous BEEBUG articles. Also see NUG page 223 for details of logical colours. As an example consider a screen containing ALL of the following colours;

RED	Logical colour 1
GREEN	Logical colour 2
YELLOW	Logical colour 3
BLUE	Logical colour 4

Using a parameter 3 of 1 would draw only Red and Yellow points.
Using a parameter 3 of 2 would draw only Green and Yellow points.
Using a parameter 3 of 3 would draw only Red, Green and Yellow points.
Using a parameter 3 of 4 would draw only Blue points.
The most normal requirement of this third parameter will be to print all points on the screen, in this case use a value of 15. So to print all points on the whole screen use PROCGPR(1023,0,15).

```
29000 CLS:INPUT''"MODE (0,1,2,4,5) ",M
29010 MODE M: IF M=0 M=2 ELSE M=4
29020 VDU29,0;896;
29030 FOR X=0 TO 1276 STEP M
29040 S=SIN(X/26)*128*EXP(-X/300)
29050 DRAWX,S
29060 NEXT
29070 PRINTTAB(3,2)"EXPONENTIAL DECAY"
29080 PROCGPR(1023,768,15)
29090 PRINT TAB(0,10);:END
29100 DEF PROCGPR(YTOP,YBOTTOM,COLR)
29110 LOCAL A%,P%,Y%
29120 LOCAL B%,C%,D%: B%=!&84: C%=!&88: D%=!&8C:
      REM Save z-page &84-&8F
29130 REM Delete from here after first use
29140 DATA 3,&15
29150 DATA &84,&84,&85,&85,&A2,&0,&86,&88
29160 DATA &86,&89,&86,&8E,&86,&8F,&E8,&D0
29170 DATA &24,&8A,&20,&EE,&FF,&A5,&86,&20
29180 DATA &EE,&FF,&A5,&88,&18,&69,&4,&85
29190 DATA &88,&D0,&12,&E6,&89,&A5,&89,&C9
29200 DATA &5,&D0,&A,&8A,&20,&EE,&FF,&A9
29210 DATA &D,&20,&EE,&FF,&60,&A5,&84,&85
29220 DATA &8A,&A5,&85,&85,&8B,&86,&86,&A9
29230 DATA &7,&85,&8D,&A9,&9,&A2,&88,&A0
29240 DATA &0,&20,&F1,&FF,&A5,&9C,&18,&30
29250 DATA &7,&25,&87,&F0,&3,&EA,&EA,&38
29260 DATA &26,&86,&A5,&8A,&69,&4,&85,&8A
29270 DATA &90,&2,&E6,&88,&C6,&8D,&D0,&D8
29280 DATA &A2,&0,&E8,&A0,&80,&C4,&86,&F0
29290 DATA &7,&A5,&8E,&F0,&9C,&D0,&9,&EA
29300 DATA &E6,&8E,&D0,&9E,&E5,&8F,&D0,&9A
29310 DATA &8A,&20,&EE,&FF,&A9,&1C,&20,&EE
29320 DATA &FF,&8A,&20,&EE,&FF,&A5,&8E,&20
29330 DATA &EE,&FF,&8A,&20,&EE,&FF,&98,&20
29340 DATA &EE,&FF,&A9,&0,&85,&8E,&C6,&8F
29350 DATA &10,&DE,&E6,&8F,&F0,&CD,6
29360 RESTORE
29370 FOR P%=TOP-176 TO TOP-8
29380 READ A%
29390 ?P%=A%
29400 NEXT P%
29410 REM Delete down to here ***
29420 P%=TOP-174
29430 VDU26,2,1,8,1,13: REM restore gphc orig, pr g
phcs, newline
29440 ?&87=COLR AND &F
29450 FOR Y%=YTOP-24 TO YBOTTOM-24 STEP-28: A%=(Y%+2
56)DIV256-1: CALL P%: NEXT
29460 !&84=B%: !&88=C%: !&8C=D%: REM Restore Z-page
 bytes used
29470 VDU1,15,3: ENDPROC: REM prtr back to chars &
off
29480 MACHINE XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX CODE
```

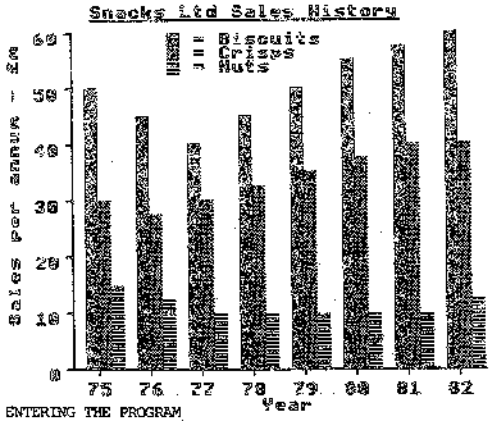
EPSON MX80 III MACHINE CODE SCREEN DUMP
by Timothy Powys-Lybbe

The following program was designed after reviewing a number of screen dumps kindly submitted by members. It combines the best features of these (I hope). It will print all graphics screens (i.e. modes 0,1,2,4 and 5) taking two minutes in MODE 0 and eighty seconds in other modes. The program automatically senses what mode is in use. The listing also includes a simple demonstration which

illustrates how to use the assembler program from within a Basic program. Alter line 20 for other modes.

There are two versions of the program - one for cassette (tested on O.S 0.1 and 1.0), and one for disc (tested on O.S 1.0). In both cases the machine code routine is stored in an area of memory below PAGE, and therefore requires no adjustments of PAGE or HIMEM. The cassette version is stored in the user area at &D00, and the disc version is stored at &A00. The user area at &D00 is corrupted by disc use, so we have used the cassette buffer and RS423 input buffer (at &A00) for storage. This means that the program cannot be used when RS423 input is in progress, but this is not likely to be a problem to most users.

Both versions can be saved and loaded transparently in a similar manner to the technique described by Gwyneth Pattit in BEEBUG No.7; and the cassette version is just short enough so that the 0.1 Operating System Bug Fix program can be included in the same area of memory, as long as the Bug Fix starts at the address hex &DD0. The program by Brian Carroll in BEEBUG No.7. works in the same way. Finally the dump should work with all operating systems and should also work across the Tube to a 6502 second processor.



ENTERING THE PROGRAM

Obviously entering an assembler program is difficult for those who are not familiar with assembler. The listing below uses the BBC Micro long variable name facility to make it easier to recognise any errors. Spaces are one problem; in the assembler program from lines 10040 to 10250 spaces are only needed after the names that start with a full stop, "LineGap" for example.

Note that for machines with a disc interface, alter line 10030 to:

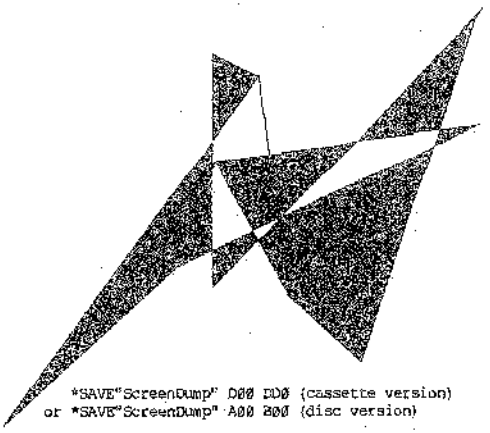
```
10030 P% = &A00:[OPT PASS
```

Do not RUN the program unless you have first listed it to your printer and saved it on tape (or disc for the lucky few). This will save hours of re-entering if you have made fatal errors.

When you run the program, you should get a random picture drawn on the screen, and then this should be automatically copied onto the printer.

Once a successful printout has been obtained, you can destroy the Basic program and assembler code (though it would be prudent to keep a copy) and just use the assembler code stored at &D00 or &A00.

To save the machine code only, use the command



*SAVE"ScreenDump" D00 DD0 (cassette version)
or *SAVE"ScreenDump" A00 B00 (disc version)

To load the machine code back into memory at any time (transparently), use this command

*LOAD"ScreenDump" (cassette or disc version)

To print the screen from Basic after loading the machine code use the command CALL &D00 (or CALL &A00 for a disc system); see line 1050 for the listing. Remember though that on series one operating systems (ie 1.0, 1.2 etc) you must turn on the printer with a CTRL B (or VDU 2) before calling &D00 (or &A00), and turn off the printer at the end with a CTRL C (or VDU 3). Thus the command:

VDU 2: CALL &D00 (or &A00 for discs) : VDU 3

will be needed to print a screen. If you are using cassette and 0.1 operating system, you can save and load all of the function keys, the user characters, this screen dump program and Brian Carroll's bug fix. Execute:

*SAVE"Utilities" B00 E00 0D0 <ret>

You can then load this back in, and run the screen dump with:

*RUN"Utilities" <ret>

Thanks to Andrew Burke for the idea of putting all graphics modes on one short program, to J.H. Williams for his neat demonstration designs, to K.E. Hussey for his idea of fitting the program into page D hex for cassette use, and to all those others who have written to us on this topic.

```
  10 PROCScreenDumpAssemble
  20 MODE0
  30 PROCDraw
  40 END
  50 :
1000 DEF PROCDraw:REPEAT
1010 CLS
1020 FOR I=1 TO 10
1030 PLOT 86,RND(1280),RND(1024)
1040 NEXTI
1050 VDU2:CALL ScreenDump:VDU3:REM Use CALL &0D00 (
&0A00 for disc) after assembler deleted
1060 CLS
1070 PRINTTAB(0,2)"Press SPACE to exit."
1080 PRINTTAB(0,6)"Press any other key to repeat."
1090 A=GET
1100 UNTIL A=32
1110 ENDPROC
1120 :
10000 DEF PROCScreenDumpAssemble
10010 xpointlo=&70:xpointhi=&71:ypointlo=&72:ypoint
hi=&73:pixelvalue=&74:printerbyte=&75:bitcount=&76:m
ode0=&77:step=&78:OSWRCH=&FFEE:OSWORD=&FFF1:OSBYTE=&
FFF4
10020 FOR PASS=0TO3 STEP3
```

```
10030 P%=&D00:[OPT PASS
10040 .ScreenDump LDA&FF:STAypointlo:LDA#&3:STAypoi
nthi
10050  LDA#0:STAmode0:LDA#4:STAstep:LDA#135:JSROSBYT
E:TYA:BNELineGap:INCmode0:LSRstep
10060 .LineGap LDA#27:JSRPrinter:LDA#65:JSRPrinter:L
DA#&6:JSRPrinter
10070 .NewLine LDA#0:STAxpointlo:STAxpointhi
10080 LDA#27:JSRPrinter:LDAmode0:BEQModeAbove0:LDA#7
6:JSRPrinter:LDA#128:JSRPrinter:LDA#2:JSRPrinter:BNE
NewColumn
10090 .ModeAbove0 LDA#75:JSRPrinter:LDA#64:JSRPrinte
r:LDA#1:JSRPrinter
10100 .NewColumn LDA#8:STAbitcount
10110 .ReadPixel LDX%xpointlo:LDY#0:LDA#9:JSROSWORD
10120 CLC:LDApixelvalue:BEQSetPrinterByte:SEC
10130 .SetPrinterByte ROLprinterbyte
10140 LDAypointlo:SEC:SBC#4:STAypointlo:BCSCheckColu
mnEnd:DECypointhi
10150 .CheckColumnEnd DECbitcount:LDAbitcount:BNERea
dPixel
10160 .Print LDAprinterbyte:JSRPrinter
10190 .NextColumn CLC:LDAstep:ADCxpointlo:STAxpointl
o:BCCCheckLineEnd:INCxpointhi
10200 .CheckLineEnd LDAxpointhi:CMP#5:BEQEndLine
10210 .ColumnTop LDA#32:CLC:ADCypointlo:STAypointlo:
BCCNewColumn:INCypointhi:BCSNewColumn
10220 .EndLine LDA#10:JSRPrinter
10230 .CheckEnd LDAypointhi:BMIEnd:JMPNewLine
10240 .End LDA#12:JSRPrinter:LDA#27:JSRPrinter:LDA#6
4:JSRPrinter:RTS
10250 .Printer PHA:LDA#1:JSR&FFEE:PLA:JSR&FFEE:RTS
10260 ]NEXT PASS
10270 ENDPROC
```

Program tested on 0·1 and 1·1 O.S.

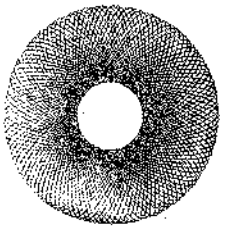
SPIROPLOT

by R.Sterry

This is a beautiful little program which generates pictures similar to those produced by a 'Spirograph'. You define the size of the 'disc' that you want to use and the position of a 'pen' within that disc (known as the 'locus' position). Many an hour can be spent playing with this program in trying to produce the magnus opus of computer generated spiroplots.

Some examples for you to try in order to get the idea of things are:

Disc Size	Locus
50	1.4
89	1.0
202	0.5
212	1.5
298	1.5



These examples should be enough to trigger your imagination. Happy designing!

```
  10  REM* SPIROPLOT BY R. STERRY *
  20  ON ERROR OFF
  30  MODE7
  40  PRINT TAB(10)"S P I R O P L O
T"TAB(10)STRING$(17,"=")
  50  INPUTTAB(0,12)"DISC SIZE      (
0 TO 360) ",B
  60  IF B>360 OR B<0.01 GOTO 50
  70  ON ERROR GOTO 20
  80  INPUTTAB(0,14)"LOCUS POSITION (
0 TO 1.5) ",M
  90  IF M>1.5 OR M<0.01 GOTO 80
 100  MODE4:VDU29,639;511;5
 110  VDU19,1,0,0,0,0,19,0,7,0,0,0
 120  A=-360:C=A+B:Q=0:REPEAT
 130  X=C*COSQ-M*B*COS(C*Q/B)
 140  Y=C*SINQ-M*B*SIN(C*Q/B)
 150  IF Q=0 MOVE X,Y ELSE DRAW X,Y
 160   Q=Q+B/800:UNTIL FALSE
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

*FX254 CALL

M.E.Horton sent in this hint. It provides a simple way to turn your 32k computer into a 16k computer.

Try: *FX 254,&40

Then press CONTROL and BREAK together. After this, you should get 'BBC Computer 16k'. It provides an easy way of making sure that your programs will run on both models. The call seems to work on a series 1 operating systems and is reset by *FX254, &80.

[Ed: It seems that any value between 0 and 127 will have the same effect as using &40, and any value between 128 and 255 will do instead of &80. For example you could just use 0 and 255].

USING FILES

by Sheridan Williams

Sheridan Williams begins a series of articles on the use of files on the BBC machine. It is hoped that this will be of interest to all those whose programs involve data handling, both on cassette and disc. The article begins at an introductory level.

All programs that perform data processing functions require some form of file storage, otherwise we cannot continue to process data from one day to the next. There are TWO main reasons why files are used:
1. To hold data over long periods.
2. To hold more data than can be held in the computer's main store.

There are three ways in which files can be held and these are:

1. In DATA statement. 2. On cassette 3. On disc

Choice number one (DATA statements) unfortunately does not rid us of the problem of limited memory space inside the computer. Choice number two (cassette) gets over that problem but introduces the problem of long and tedious reading and writing of the data to and from the file. Cassettes only read/write at about 100 characters per second, so to process a 25k file (not particularly large in realistic data processing terms) it would take 4 minutes. Also cassettes are severely limited because they have no capabilities for positioning the tape accurately anywhere in the file.

Thus if you wanted to change one field in just one record on a 25k file, it would take 8 minutes to do - 4 minutes to read the whole file and 4 minutes to rewrite it. You would also need a 32k machine for the file to fit.

Discs provide the best solution, however they are expensive compared to cassettes. There is no avoiding the fact that many data processing applications require the use of a disc, in this case the costs must be borne in mind. (See the December issue for a brief comparison of 5 disc drives).

FILES, RECORDS, FIELDS

Before we look at the statements in BASIC and some routines for file processing, we must understand some of the jargon.

File - A collection of related records. An ordinary metal filing cabinet may contain several files, or just one file, or even none! Similarly a cassette or disc may hold several files. Examples of files would be: Payroll file, Library book file, Criminal file, Stock file, Customer accounts file.

Record - A set of information relating to an individual item in the file. A card index file contains individual cards, each card being a record. Examples are: Criminal record, Library book record, stock record.

Field - A subdivision of a record. For example, a library book record would probably contain some (or all) of the following fields: Book Title, Author, Publisher, ISBN.

For the purpose of demonstrating files we will work with the following file throughout this series of articles:

Filename: PEOPLE RECORD DESCRIPTION

FIELD	CONTENTS
1	Surname
2	Title
3	Sex (M or F)
4	Date of Birth

Suppose that we wish to write a program to find all the males in the file that is stored in DATA statements. The following program will do this:

```
10 PRINT"List of all males in the file"
20 REPEAT
30  READ surname$,title$,sex$,dob$
40  IF sex$="M" then PRINT title$;"
";surname$
50 UNTIL surname$="*"
60 DATA Bloggs,Mrs,F,240472
70 DATA Jones, Mr, M,171066
80 DATA Smith, Mr, M,020155
90 DATA     *,  *,  *,     *
```

It is rather tedious, and a severe limitation to hold a file in DATA statements. For example to add, delete, or amend a record held in DATA statements you need some knowledge of programming, or at least be familiar with the BBC machine because you are altering the program itself when you are changing the data statements.

So we turn to cassette and disc files. Commands associated with files are: OPENIN, OPENOUT, PRINT#, INPUT#, BPUT#, BGET#, CLOSE#, PTR#, EOF#.

For the rest of this article, all programs given will work on a disc system or a cassette system. [Beware if you have O.S. 0.1 because you will have to run the "Bug Fix" program (given in July 1982 BEEBUG) to cure the cassette filing bug]. The term for the type of file that we will be using in this article is a SERIAL or in some cases a SEQUENTIAL file.

A file must be opened before data can be written to it, and similarly it must be closed when you have finished.

On a disc system several files may be in use at once, for example a sophisticated Payroll package may require all these files to be open at once:-

Employee tax file
Tax table reference file
Employee name & address file
Tax authority payments file

For this reason, when a file is opened it is allocated a channel number.

For example, you open a file for output (ie to be written to) as follows:-

```
channel=OPENOUT"PEOPLE"
```

if you then PRINT channel you will find out what channel the computer has allocated when this file was opened (probably 17). You needn't ever know the channel number because you will always use a variable to represent it. Note that the same channel number isn't always associated with the same file, in another program you may find that the computer has allocated a different channel number.

If you try this in immediate mode you will hear the cassette relay click on, or the disc drive start. It is writing the 'header' containing the file name to the cassette or disc. If you now type CLOSE#channel then the cassette/disc would switch off, and you will have created a file, albeit an empty one. This is analogous to a filing cabinet with a label saying 'Employee file' on the drawer, but with nothing inside.

To write data to a file is quite simple - you use the PRINT# statement. (Make sure you have a cassette or disc in situ). Proceed as follows:-

```
10 ch=OPENOUT"FRUIT"     20 PRINT#ch,"Apple","Pear","Orange"     30 CLOSE#ch
```

If you want to prove that the data really has been put on the file then we must establish how to read the data back from the file. To do this you must open the file for input rather than output as we did before. We use the statement OPENIN. We are now going to input data from the file, but stop for a minute; how will we know when there is no more data on the file to read? The function EOF# is provided for just this purpose. EOF# returns either true (-1) or false (0) depending on whether the End-Of-File has been detected.

So we can use OPENIN and then a REPEAT UNTIL EOF# loop to control our input from

the file. This is done in the program below (lines 50-100). It reads back the data written to the file in lines 10-30 above. (You can join both of these programs to make one before you run them).

```
 40 INPUT"Press 'return' when ready
to read back file",Q$
 50 chan=OPENIN"FRUIT"
 60 REPEAT
 70 INPUT#chan,fruit$
 80 PRINT fruit$
 90 UNTIL EOF#chan
100 CLOSE#chan
```

[If you are using cassette, rewind the cassette before you read back the file].

Once the file is open you can write (print) as many items of data as you want to the file, obviously at some point the cassette/disc will fill up so this is the upper limit to the size of the file. Files cannot realistically be extended across to different discs or cassettes. It is wrong to open and close a file between each write operation.

Each time a file is open for writing, the data is written to the BEGINNING of the file, so that any data previously there is overwritten. For example this program will only read back 'Apple' even though you have written two fruits to the file. Lines 10-30 write "Tangerine" to the file, then lines 40-60 write "Apple" to the same file. However, lines 70-110 read (input) the data from the file, and yet you will see that there is only one fruit (Apple) on the file.

```
 10 c=OPENOUT"FRUIT"
 20   PRINT#c,"Tangerine"
 30 CLOSE#c
 35
 40 c=OPENOUT"FRUIT"
 50   PRINT#c,"Apple"
 60 CLOSE#c
 65
 70 c=OPENIN"FRUIT"
 80   REPEAT
 90     INPUT#c,fruit$: PRINT fruit$
100   UNTIL EOF#c
110 CLOSE#c
```

If you think about it, you will probably say that this is ridiculous because you can never add anything to the end of the file, let alone modify one record in the middle of the file. Each time the file is opened for reading or writing the record pointer associated with that particular channel is set to the first record. There are theoretically three solutions:

1. Load the whole file into memory, make the modifications, then write them all back to the file. This is only feasible with a small file that will fit into memory.

2. Use two tape recorders, read one record at a time from the file on one recorder, if it doesn't need modifying then write it to the file on the second recorder, if it does need modifying then modify it before writing to the second recorder. Continue this process until the whole file is updated. However, the BBC micro doesn't readily support 2 cassette decks.

3. As in 2 above, but using two disc files, one for input and one for output. (Note that a disc can hold and process more than one file at once).

4. Use a "random access" disc file. (We will cover this topic in a later part in this series).

Next month we will look at the first of the three options above, taking examples from the cassette-based version of Masterfile.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SIDEWAYS SCROLLING

The Beeb has a ready-made command to perform a sideways scroll: VDU23;13,A;0;0;0; where A represents the new position on the screen. See review of Creative Graphics in this issue for further details but try:

```
 10 FORA=1TO200
 20 TIME=0:REPEAT UNTIL TIME=100
 30 VDU23;13,A;0;0;0;
 40 NEXT
```

For the reverse direction, alter line 10 to: FOR A=200 TO 1 STEP -1

BOOK REVIEWS

BBC MICRO BOOK SURVEY with Star Ratings and References to BEEBUG Reviews.
By David Graham.

Assembly Language Programming for the BBC Microcomputer.		
Ian Birnbaum	325 pages.	
MacMillan	£8.95	
BEEBUG no 8	*****	
Creative Graphics on the BBC Microcomputer.		
John Cownie	110 pages.	
Acornsoft	£7.50	
BEEBUG no 9	*****	
Basic Programming on the BBC Microcomputer.		
N.M.B. Cryer	195 pages.	
Prentice Hall	£5.50	
BEEBUG no 2	***	
Let Your BBC Micro Teach You To Program.		
Tim Hartnell	193 pages.	
Interface	£6.45	
BEEBUG no 8	*	
Practical Programs for the BBC Computer and Acorn Atom.		
D. Johnson-Davis	119 pages.	
Sigma	£5.95	
BEEBUG no 8	***	
30 Hour Basic.		
Clive Prigmore	254 pages.	
NEC	£5.95	
BEEBUG no 9	*	
The BBC Micro Revealed.		
Jeremy Ruston	141 pages.	
Interface	£7.95	
BEEBUG no 9	*	

Key to star ratings:
***** Highly recommended.
*** A useful book.
* Lowest rating.

Creative Graphics on the BBC Microcomputer
Acornsoft 110 pages
By John Cownie price £7.50
Reviewer David Graham

Order from Acornsoft Ltd., Vector Marketing, Denington Industrial Estate, Wellingborough, Northants NN8 2RL.

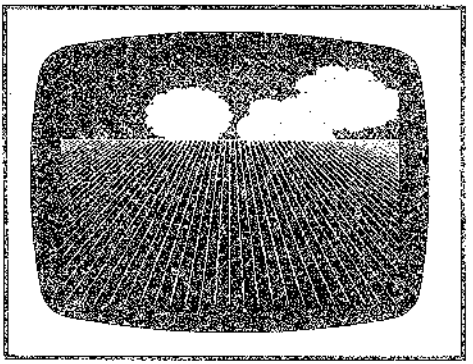
There have been a number of rather mediocre books published about the BBC micro. This one stands head and shoulders above the crowd, and reinforces Acornsoft's reputation for excellence.

Its relatively small number of pages are packed with a wealth of ideas on the use of graphics on the BBC micro. The book is logically laid out with the following chapter headings:

1. Graphics Commands
2. Functions and Symmetry
3. The Third Dimension
4. Animation
5. Recursion
6. Pictures

It takes the reader who has a grounding in BASIC from an introduction to the graphics commands on the BBC machine, right through to the generation of a number of animated sequences which make use of the techniques developed during the course of the book.

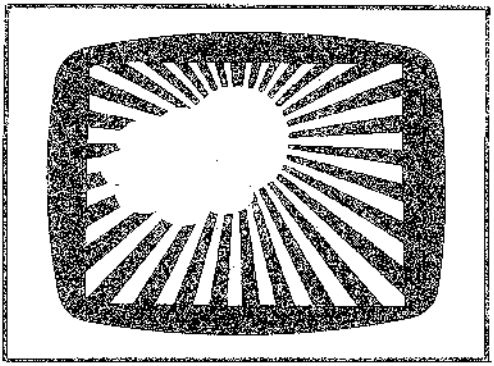
The whole is skilfully managed, and the book contains fully documented listings of 36 illustrative programs, many of which produce very nice visual displays of their own, and most of which will run with slight modifications on a model A. Moreover full use has been made of the advanced features of the BBC micro, including procedures, which play an important part in the structuring of the final animation sequences.



The one area of screen graphics that is excluded from the book is the user defined characters. The book concentrates on the kind of graphics that can be produced using the PLOT command. This intentionally omitted area might well provide the material for a sequal: but in the meantime Acornsoft are not giving away ALL the secrets of games such as Planetoid !

There is an accompanying cassette (£10.00 extra) containing the 36 programs, but since most

are very efficiently written - and therefore short - it is not too much trouble to enter them by hand.



We publish below with permission from Acornsoft the listing of one of the final animated sequences, together with screen photos, to illustrate some of the techniques used, and the quality of the programs listed. This program - Windy Field - firstly generates a series of billowy clouds which float across a blue sky. A furrowed field is then rapidly generated. The clouds then sink below the horizon, and the horizon itself scrolls down. A new picture is then generated with a sun partly obscured by cloud. This picture then ripples through a series of colour sequences (which may be speeded up by pressing any key). As you will gather from the length of the program, this is all achieved very efficiently.

One of the most interesting points to note is the use of the command VDU 23;13,X;0;0;0; to scroll the screen sideways. X is simply the degree of scroll leftwards. To scroll right, set this high, then reduce it by stages. This beautifully simple undocumented command is typical of the kind of things that may be learned from this book.

```
   0 REM Windy field
  10 REM Copyright (C) Acornsoft 1982
  20 MODE1
  30 VDU23;1,0,32;0;0;0;
  40 VDU19,0,4;0;19,2,2;0;
  50 WIND%=0
  60 FORY%=860 TO 700 STEP -40
  70 PROCCLOUD(200+RND(870),Y%,100+RND(100),80+RND
(20),TRUE,3,0)
  80 NEXT
  90 REPEAT
 100 WIND%=WIND%-1
 110 VDU23;13,WIND%;0;0;0;
 120 A=INKEY(12)
 130 UNTIL WIND%=0
 140 PROCGROUND(16)
 150 VDU28,0,9,39,0
 160 COLOUR128
 170 A=INKEY(300)
 180 PROCDOWN(10,80)
 190 VDU28,0,31,39,10
 200 PROCDOWN(22,0)
 210 VDU19,2,3;0;19,3,7;0;
 220 PROCSUN(600,700,1000,TRUE,3)
 230 PROCSUN(600,730,200,FALSE,1)
 240 PROCCLOUD(400,600,300,200,FALSE,2,3)
 250 A=INKEY(300)
 260 REPEATVDU19,RND(4)-1,RND(8)-1;0;:A=INKEY(200)
:UNTIL FALSE
 270 END
 280 DEFPROCDOWN(N%,D%)
 290 FORI%=1 TO N%
 300 VDU11
 310 A=INKEY(D%)
 320 NEXT
 330 ENDPROC
 340 DEF PROCGROUND(S%)
 350 VDU28,0,31,39,10
 360 COLOUR129
 370 CLS
 380 VDU26
 390 GCOL0,2
 400 VDU29,640;0;
 410 MOVE-640,700:DRAW640,700
 420 FORX%=-640 TO 640 STEP S%
 430 MOVEX%,700:DRAW4*X%,0
 440 NEXT
 450 ENDPROC
 460 DEFPROCCLOUD(X%,Y%,SX%,SY%,SCROL%,C1%,C2%)
 470 VDU29,X%;Y%;
 480 L%=6+RND(8)
 490 MOVE0,0:MOVESX%+SX%/L%,0
 500 X1%=SX%+SX%/10:Y1%=0
 510 IF WIND%>120 THEN S%=-1 ELSE S%=1
 520 FORI=0TO 6.3 STEP 0.1
 530 IF SCROL% THEN VDU23;13,WIND%,0;0;0;:WIND%=WI
ND%+S%
 540 X%=SX%*COS(I)+SX%/L%*COS(I*L%)
 550 Y%=SY%*SIN(I)+SY%/L%*SIN(I*L%)
 560 GCOL0,C1%
 570 MOVE32,12:PLOT85,X%,Y%
 580 MOVEX1%,Y1%
 590 GCOL3,C2%:DRAWX%,Y%
 600 X1%=X%:Y1%=Y%
 610 NEXT
 620 VDU29,0;0;
 630 ENDPROC
 640 DEF PROCSUN(X%,Y%,S%,RAYS%,C%)
 650 VDU29,X%;Y%;
 660 GCOL0,C%
 670 MOVE4,12
 680 T%=TRUE
 690 FORA=0 TO 6.3 STEP 0.1
 700 MOVE0,0
 710 X%=S%*SIN(A)
 720 Y%=S%*COS(A)
 730 IF RAYS% AND T% THEN MOVE X%,Y% ELSE PLOT 85,
X%,Y%
 740 T%=NOT T%
 750 NEXT
 760 ENDPROC
```

Title: 30 Hour Basic
By: Clive Prigmore
Reviewer: David Graham

254 pages
price £5.95

This book bears the BBC owl logo, and is promoted in connection with the Computer Literacy Project. With the word "BBC" on both the spine and the front page, would-be purchasers are likely to conclude that this is a course for users of the BBC micro. In this they would in my view be mistaken. The programs in this book were written for, and tested on a PET - since BBC micros were not available when the course was designed; and this text book ignores all the innovatory features of BBC Basic. For these reasons BBC micro users would I believe generally do well to steer clear of it.

Before backing up these allegations let me present the plus side of this book. It is quite well produced, gives you a lot of pages for your money, it has a spiral spine so that it will lie flat while you are keying in programs; and above

all it is a structured text (though unfortunately one that does not teach structured programming), and it has self assessment questions at the end of each unit.

But let us see just what a BBC Micro user would miss if he followed the course.

There is nothing on procedures.
There are no REPEAT..UNTIL loops.
There are no IF..THEN..ELSE statements.
There is almost no use of long variable names.
There is nothing on integer variables.
There is nothing on sound or graphics.
All modes but Mode 7 appear to be ignored.
The TAB function has no Y dimension.
There is nothing on error trapping.
The RND function is not properly used.
There is nothing on *FX calls, the function keys, or the use of TIME, PAGE, HIMEM etc.

This is a quite unbelievable list of omissions, but the BBC micro user has more to contend with. The listed programs are of a general nature, and contain many additional notes with advice for the BBC micro owner. For example in the dice throwing program listed on p. 170 it says "BBC: 60 omit line ". Line 60 is the RANDOMIZE command available on a number of extended Microsoft BASICs, but not on BBC Basic. Its function is to shuffle the random number generator called by the command RND. Deleting the line will remove the randomness of the results. BBC owners should have been told to replace line 60 with 60 A% = RND(-TIME), which has the same effect as RANDOMIZE.

There are two points to be made here. Firstly the BBC owner should not have to follow a number of corrections in order to make a program run on his machine, and secondly the corrections are not always correct, and in some cases are left out altogether. For example the program on p. 174 will not run because the RANDOMIZE has been left in.

If you are a relative beginner to BASIC, and are using a more primitive computer than the BBC micro, then this book could certainly be of use, though its lack of machine-specific treatment and absence of graphics will limit this use. But if you are using a BBC micro then you should not buy this book. I say this partly because of what the book lacks; and the list above makes very depressing reading; but perhaps even more so for a second reason: the relatively unstructured programming that this book fosters.

If you were learning to fly a jumbo jet, you would get only relatively little use from a Gipsy Moth manual - and you might even learn some bad habits. This is presumably why the NEC and BBC are getting together to bring out a real jumbo jet version in the near future - we will keep you informed.

Title: The BBC Micro Revealed — 144 pages
By: Jeremy Ruston — price £7.95
Reviewer: David Graham

This book bears the same form of title wording as the first rate book on the PET - "The PET Revealed" by Nick Hampshire - and the companion volume - "The Vic Revealed" by the same author. But Jeremy Ruston has not in my view produced the same calibre of magnum opus.

In the first case the Ruston book carries no index OR contents page, so that it is difficult to find your way to its various sections without reading from start to finish, and even then subsequent reference will not be easy.

In fact the book contains four sections
Section One (p5): The 6845 CRTC
Section Two (p36): Memory Locations
Section Three (p102): Basic Program Storage
Section four (p109): Basic Variables Storage

Section one looks, as its title suggests, at the video controller chip, and its registers, and shows a number of tricks that can be achieved by addressing them directly.

Section two is by far the longest section of the book, and is the most problematic. Essentially it gives the functions of a whole series of locations in RAM that are used by the operating system to store such things as cursor coordinates, or current palette, and also buffer areas. Generally speaking the BBC machine's FX and OSBYTE calls make such details somewhat redundant for normal programming purposes, added to which, if you do PEEK the locations given by Ruston in a program of your own, then it will not work across the Tube. Having said that, there is a more serious problem. Ruston has done his detective work on a machine fitted with a 0.1 operating system; and when I tested a sample of the locations specified, none of them worked with the series one (1.0) operating system that I was using.

The largest section of the book is therefore unfortunately rendered out of date with the advent of the new operating system; and although a note at the start of the section makes this clear, there is no warning on the cover to alert the reader to this fact before he purchases the book.

Section three usefully explains the way in which BASIC programs are stored in memory and contains a neat program for listing all of BASICs reserved words with their tokens, direct from the machines' look-up table. Section four discusses the storage of variables, and like sections one and three makes interesting reading; but even without the O.S. 0.1 problem the book can never be "destined to become the bible of the BBC user" as Personal Computing Today is reported to have claimed: the ground that the book covers is far too limited. The BBC micro is a very complex machine, and the Bible is a very LARGE book.

Title: Assembly Language Programming for the BBC Microcomputer — 305 pages
By: Ian Birnbaum — price £8.95
Reviewer: Colin Opie

Published by Macmillan Press, this book helps enormously to fill a gap in readily available publications concerning different aspects of home computers. It is also a pleasure to see such a book printed and published in England, rather than yet another import from across the water, though one would expect this as the book is concerned with the BBC computer.

In essence I believe the author has done a first rate job. Many books exist on 6502 Assembly Language programming but there is a definite need to apply low-level techniques to a particular machine. The operating system and general environment of different machines means that varying facilities are available to individual users. The author has exploited and discussed well the art of 6502

programming on the BBC microcomputer. He has resisted the temptation to spend too much space on detailed explanations of each machine instruction, and has supplied vast quantities of examples linked to adequate instruction descriptions.

The book is split into 10 chapters and 9 appendices. All but one of the chapters relates to programming, the remaining chapter (10) being devoted to some utility programs. Appendices cover the 6502 instruction set, useful information about using the BBC computer, and a few odd discussions on 'floating point representation' and 'indexed indirect' addressing. One exceptionally useful facet is the number of exercises supplied, with a full solution to each one given at the end of the book.

In terms of layout I think it a pity that so many different type faces are used, and that their use is not consistent throughout the book. The appendix on the 6502 instruction set is hand-written although very readable. More spaces could have been used in the majority of program listings to make them more readable. As it is there are often pages which have a listing squashed up on the left hand side with the rest of the page completely blank.

Two tapes can accompany the book if desired and cost £9.00 each or £16.00 if bought together. The book does not require the tapes to make it useful. Despite the sometimes poor layout as mentioned above, the author has produced an extremely useful aid to anyone with a BBC microcomputer wishing to learn or improve programming at assembly level. I would heartily recommend the book as a self teaching text and would not hesitate to use it as a text book with students.

Globe Book Services are offering a 10% discount on this book to BEEBUG members. See the advert on page 50 of the supplement.

STOP PRESS

Last minute addition to book discounts. The following shop stocks computer books and will offer free postage to BEEBUG members. Please telephone to check stocks.
Appleby, Myers & Clarke: 60 Market St., WATFORD. Tel:31960.

BEEBUG SOFTWARE COMPETITION RESULTS

We had twice as many entries for our second software competition as for our first; and were pleased to see some quite novel ideas amongst the entries. We have used the same judging criteria as for the last competition, and these were outlined in BEEBUG no 5. This time we have given a total of over £800 worth of prizes.

PRIZE-WINNERS

£200	M Tilley	Debugger	£20	A Baker	Tune
£50	J Banks	Space City	£20	D T Blyth	Compacter
£50	J Hardie	Life	£20	E D Carter	Ohms-Law
£50	R R Hull	Beebmaze	£20	N Mallinson	Dive Bomber
£50	B J Kilby	Five Dice	£20	K Penton	T.Bug.
£50	S Wilkinson	Racer	£20	G Pomfret	Subsearch
			£20	Mrs J E Sey	Elements

£15 Prizes: L P Baxter, G K Blackwell, B Carroll, M Case, S J Curtis, P Giblett, C J Hall, Lynton Jones-Ng, I Pomeroy, D Seaton, T Segall, K Stephenson, C Walton, J Webb.

£10 Prizes: M Harman, M Humphries, R Moseley, A Paskins, A Schild, Mrs. E Shone.

Once again, we would like to thank all those who entered, and encourage entries for our next competition. Full details appear in this issue.

DISC ROUNDUP

MEMORY SHIFT ROUTINE

Object: To run large cassette programs from disc.
Method: Load software from disc and then shift it down.

Many existing cassette programs are too large to run on a disc machine. The reason for this is that a cassette machine is able to start storing BASIC programs from location &E00. In a disc system this start location is &1900, the area between these two locations being used for various disc operations - a loss of about 2.5K of program memory.

When the need arises it would be useful to retrieve this space so that a large cassette program can then run on a disc system. To do this automatically we need to load the program from the discs into the normal program area at &1900, shift it down to &E00, set up a few variables and then run it.

Given below is a one line program which will define one of the function keys to perform this 'shift+run'. Type the program in and save it on disc with the name "SHIFT". Each time you need to set up the function key type:

CHAIN "SHIFT" <return>.

Assuming the function key is set, to shift a program simply LOAD it into memory and press function key 9. This will automatically run the program. If you ESCAPE from the program you need only type RUN to restart it. Pressing BREAK will re-instate the disc system but the program will now be lost from memory.

Program:

```
10*KEY9FORI%=0TOTOP-PAGE STEP4:I%!&E00=I%!&1900:
NEXT|MPAGE=&E00|MEND=|MLOMEM=TOP|MRUN|M
```

Note that the SPACE between the words PAGE and STEP is important.

In 'Practical Computing' (FEB'83 p.71 - listing 2) there is a machine code program which you can append to the beginning of your programs to perform this shift operation - though we have not tested it).

CNO/RP

AUTO START

Object: To run a program automatically when shift and Break are pressed simultaneously.
Method:Use *OPT 4,3, *BUILD and !BOOT.

When using a BBC micro with discs, an option is available to automatically perform a *RUN or CHAIN or *EXEC function just by pressing the shift and break keys simultaneously. The following shows the steps necessary to define the user keys automatically in this way.

The file of key definitions is created with the *BUILD utility. The special file MUST be named !BOOT (the reasons for which are of no consequence).
Execute the following:

```
*BUILD !BOOT <return>
  1 *KEY0 |ORUN|M <return>
  2 *KEY1 |N|LLIST|M|O  <return>
    .
    .
    .
  9 *KEY9 |O|B|LLIST|M|O|C <return>
 10 *KEY9 DIM%=-1!MP.HIMEM-P%, TOP=PAGE|M <return>
 11 <press ESCAPE>
```

Follow this with: *OPT 4,3 <return>

Line numbers are given automatically by the *BUILD command and are always in steps of one. At the end of each line press the RETURN key, and at the end press the ESCAPE key (just as the AUTO command is used in BASIC).

The command at the end of the above sequence, *OPT 4,3 sets a marker on the current disc to indicate that the file should be brought into memory with the *EXEC command. This takes place when you hold down the shift key, press and release break, wait a second or two and release the shift key. If you release the shift key too soon then an ordinary BREAK will occur. Different values of n for *OPT 4,n will indicate:

(n=0):NO ACTION;
(n=1):CHAIN THE FILE;
(n=2):*RUN THE FILE.

In all cases the file must be named !BOOT.

In some respects you can achieve greater flexibility by using !BOOT to chain in another program. This simply involves making line one of !BOOT take the following form: CHAIN "D.name", where name is the name of the program that you want to auto-boot, and D is the directory that it is in. The program could be a keyset routine, as above, or it could be a disc menu, or just a favourite program.

RP

TEXT FROM DISC

Object: To read in text directly from disc during the running of a program. This is extremely useful for providing program instructions etc, without taking up valuable memory.
Method: Use *TYPE (fsp) and *BUILD (fsp).

THE COMMAND *TYPE (fsp) in a program will cause a text file with numbered lines to be sent directly to to the screen from disc. If the file contains more than a screenful of text, VDU14 (Control N) may be used for paging purposes. The text file can be easily generated using the *BUILD command.

For example, suppose you wish to provide the instructions for a game in this way. You could begin by typing *BUILD $.TEXT!
then type in the text (the line numbers appear automatically), and the text is automatically saved to disc under the title $.TEXT!. To escape from *BUILD, press ESCAPE.

In the games program, the following can be used:

```
  80 PRINT"INSTRUCTIONS?"
  90 ON INSTR("YyNn",GET$) GOTO
 100,100,130,130 ELSE 90
 100 VDU14
 110 *TYPE $.TEXT!
 120 VDU15
 130 REM ** MAIN PROGRAM **
```

DEG

MERGE

Object: To merge 2 programs.
Method: Use *SPOOL and *EXEC.

To add program A to program B, where A could be a procedure or second program.

1. Load program A into the machine.
2. Renumber program A so that the two sets of program lines do not clash. It is usual to make the line numbers of A greater than those of B.
3. *SPOOL "name" <return> where name is any name to be applied to program A.
4. LIST <return>
5. *SPOOL <return>

This process creates an ASCII file of program A on disc - i.e. it is not stored using tokens for Basic commands. To append this to program B, continue as follows:

6. Load program B
7. *EXEC "name" <return>

This completes the operation. The *EXEC command reads in data from disc as if it came from the keyboard (hence the need to de-tokenise the Basic with *SPOOL).
Note that where line numbers clash, those of A will obliterate those of B.

DEG

ACCIDENTAL PROGRAM LOSS

Object: To avoid accidental program loss.
Method: By keeping backup copies, and 'locking' files on disc.

To cover yourself against accidental program loss always keep a backup copy. These can be conveniently saved in directory B. Thus use:

SAVE "TEST1"

and also

SAVE "B.TEST1"

To be doubly sure, you can use *ACCESS (fsp) L to "Lock" a file, and prevent it from accidental erasure. Also it is best to keep the backups on a different disc in case of disc failure.

BEWARE OF *COMPACT

When you use *COMPACT to compact a disc it will wipe out programs in memory. This creates an easy trap for the unwary. Suppose you have just finished developing a program, and decide to save it to disc. You try, but are informed that the disc is full. You are sure there is plenty of room on the disc so you *COMPACT. You may have been right, but you will not need the space now, since *COMPACT will have destroyed the program that you were about to save.

DEG

MICROWARE DISCS

We reported in the Disc System Review in our last issue that we had a drive fault on one of Microware's 100k drive units. We have since tested two similar units of theirs for a period of several weeks, without the recurrence of the fault. This suggests that the reported fault was not typical of the batch. At BEEBUG we have purchased one of Microware's larger drives, and have had no problems with this one either.

DEG

NO DOS MANUAL & NO UTILITIES DISC

If you order a disc interface from Acorn, you will not receive an operating manual for the system. Acorn's policy has been to supply these, and a utilities disc (essential for formatting discs before use) only when you buy an Acorn drive. As we said in our last issue, it was not acceptable in our view to sell a product (i.e. a disc interface and operating system) and then refuse to provide a manual for its use. We have been in discussion with Acorn about this, and they have decided to sell the utilities disc and manual together at a price of around £30, obtainable from Vector Marketing. This is clearly pretty expensive, but it does provide one way out for those stranded with non-Acorn drives and no way to use them.

In the next issue, if all goes well, we shall be publishing a disc formatting program together with details of the DFS (Disc Filing System) commands. (We have already given brief notes on these DFS commands in BEEBUG no.8 p.9).

DEG

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

*FX18 CALL

Those of you lucky enough to be running an OS from 1.0 onwards may like to be reminded (or informed!) of the *FX18 instruction. This will reset the user defined keys so that they no longer contain character strings. Thanks to David Hoffman for this one.

DEFINING KEYS

Roger Wilson has noted that under certain circumstances a colon can not be used in place of !M when defining strings for keys through the use of *KEY. The important point is that colons can only replace the newline code when the resulting string might be interpreted as a sensible BASIC line. For example, OLD can be followed by the newline character and other text, but it may not be followed by a colon because it cannot be part of a BASIC program.

LAST LINE

Douglas Nunn has found a way of quickly determining the last line number in your program. Press the (CTRL) and (SHIFT) keys down and then hit the (ESCAPE) key twice. On releasing the keys the message "Escape at line XXXX" will appear, where XXXX is the last line number used. This is especially handy after renumbering, or when appending programs to each other. The operation works on any OS.

BBC BASICS – PROCEDURES –

by Rob Pickering

This month our column for the less experienced user covers procedures and their use in structured programs.

WHAT IS A PROCEDURE ?

Although it is perfectly feasible to write programs for the BBC Micro without any knowledge of procedures, there is no doubt that their correct usage leads to a far superior program. But beware; using procedures wrongly can be much worse than not using them at all!

Think of a short part of a whole program, which carries out one minor task within the program. A few lines which, say, wait for the user to press a key before continuing. Assume that this minor task will be performed several times within the program, which is fairly typical. We could enter the same lines at each stage of the program where it is required, but this is tedious and takes up a lot of memory. What we do is make the relevant group of lines into a collective unit called a procedure. So you can think of a procedure as being a section of program which performs a given task as many times as required. It is similar in some ways to a subroutine (ie. using GOSUB...RETURN) but has certain advantages over the subroutine.

Each procedure is given a name to distinguish it from other procedures. A procedure name is very similar to a variable name, but with two real differences:

(a) they must always start with the keyword "PROC" in order to distinguish them from variable names,

(b) they may not end with a "$" or "%" symbol.

Choosing an appropriate name for a procedure is very important, not only to make the program understandable to other people, but also to help yourself to remember what all the procedures do. Since the purpose of our example procedure is to wait for the user to press a key, it seems sensible to call it "PROCwait". Note also that using lower case letters for the name helps to make it clearer to read. Here then is our example procedure in full:

```
100 DEF PROCwait      120  key=GET
110  *FX15,1          130 ENDPROC
```

As you can see this is only a simple short example and as such it is not typical of the usual length of procedures. You will notice that line 100 starts with the letters "DEF", short for DEFINITION. It simply informs the computer that this is the start of a procedure that we are 'defining'. We only have to define a procedure once within a program, then to use it we simply state its unique name. For instance, here is an example of how we might use PROCwait within a program. Try it out and see what happens. Line 40 is particularly important here since it stops the computer trying to execute the procedure an extra time.

```
10 PRINT"Press any key to continue..."
20 PROCwait
30 PRINT"Thankyou."
40 END
```

Notice that we do not need to know at what program line the procedure starts in order to use it. The BBC Basic holds a list of where the procedures are in the program, each location being stored when the procedure is defined. This means that when you use a procedure the Basic does not have to search all the way through the program in order to find the right one, hence obtaining fast operation. You should also realise that there is nothing declaring which instruction to go to at the end of a procedure. BASIC takes care of this for us as soon as it reaches the ENDPROC statement, passing execution back to the instruction following the one which called the procedure.

PARAMETERS

You will inevitably have heard of parameters, but may not know what they are, or what their purpose is. To explain how they are used in the context of procedures, we will consider a variation upon our previous example of a procedure, in which we would like to wait for a number of seconds before continuing, rather than waiting for the user to press a key. We will initially assume that the delay time is fixed at five seconds. This time the procedure is to cause a delay rather than waiting, so a sensible name would be "PROCdelay", as given below:

```
100 DEF PROCdelay       130   REM Do nothing
110  TIME=0             140  UNTIL TIME>100*5
120  REPEAT             150 ENDPROC
```

Now to cause a five second delay we need only to use the statement "PROCdelay". But uses for a fixed delay are comparatively few. It would be far more useful to have a procedure that would wait for a specified length of time, but be able to specify that time when we use the procedure. One way to do this is to replace the number 5 with a variable in line 140 of the definition, then we just store the required number of seconds in that variable before using the procedure. However, this is a very bad way to use a procedure (which will be explained in due course). What we should do is use a PARAMETER. Look at the procedure below:

```
100 DEF PROCdelay(secs) 130   REM Do nothing
110  TIME=0             140  UNTIL TIME>100*secs
120  REPEAT             150 ENDPROC
```

A parameter is simply a value which will be used by the procedure in order to carry out its task. We use the expression "parameter passing" for the way in which values are passed from the main program into the procedure. Here, the procedure definition includes a Parameter, (a variable named "secs") which will determine the number of seconds delay. The variable name "secs" in the definition applies to a variable which will exist in the locality of the procedure only. This concept of 'LOCAL' and 'GLOBAL' variables requires careful consideration.

LOCAL AND GLOBAL VARIABLES

Something which exists inside a procedure definition is said to be "LOCAL" to the procedure. A reference to something outside the procedure is said to be a "GLOBAL" reference. When a variable name is given as a parameter in the procedure definition it does not then use a variable of the same name from the main program. What happens is that when we use the procedure we must give a parameter there too. The value given in the call to the procedure is automatically copied into the procedure's own local variable. An example will help to make this clear.

```
 10 secs=100              70 END
 20 FOR s=1 TO 4         100 DEF PROCdelay(secs)
 30  PRINT"Value of      110  TIME=0
secs in main=";secs      115  PRINT"Value of secs in PROC=";secs
 40  PRINT"waiting ";s;" 120  REPEAT
seconds."                130   REM Do nothing
 50  PROCdelay(s)        140  UNTIL TIME>100*secs
 60 NEXT s               150 ENDPROC
```

Enter and Run the complete program above which demonstrates that the variable called "secs" in the procedure is actually a different one from the variable "secs" in the main program. In the main program "secs" is simply set to the value 100 and not used, but the fact that the variable of the same name within the procedure varies demonstrates that they are separate. Also notice that the value of "s" in the main program is copied (passed) into the procedure parameter "secs" automatically as the procedure is called with "PROCdelay(s)".

It is not necessary to give a variable name in the procedure call, e.g. the command "PROCdelay(5)" would be perfectly acceptable, causing a five second delay. The early user guides stated that it was not possible to pass strings to procedures. This is not true, you may pass strings to procedures, but remember that the type of parameter MUST match the type of variable in the definition. A fact which limits the

usefulness of procedures in BBC Basic is the anomaly that parameter passing techniques cannot be applied so as to pass a value back to the main program through the use of local variables, (see article in BEEBUG vol.1 no.3, p26).

Local variables are not limited to those which we use as parameters, in fact we can make all the variables within a procedure local to itself. To achieve this we are provided with the BASIC command "LOCAL" which is followed by a list of all the local variables. The effect of doing this may not seem very important, but it is. Using all procedure variables as local means that you are not relying upon a specific variable existing within a program, and consequently the procedures become of common value to more than one program. It means that you can take a procedure from someone else and need only know what task it performs, its name, and what parameters it needs. You don't have to worry about it upsetting any variables in your own main program. The whole idea of having a collection or library of common procedures is very important, preventing unnecessary duplication of effort and allowing programs to be written much more quickly. Looking back through past issues of BEEBUG you should come across some useful procedures in our "Procedure/Function library" column - BEEBUG vol.1 no.5 p.28 carries an index.

STRUCTURED PROGRAMS

An important topic related to the use of procedures is that of structured programming. To explain what this is to newcomers who may not have encountered it before, we could say that it is a way of writing programs so we can actually see what they do. The first section of the program is called the MAIN section and contains generally very 'readable' code, calling upon each procedure to carry out its task when it is needed. The other main section of the program is merely a collection of all the procedure definitions. This 'standard' layout is exemplified by the following typical program. See if you can tell what it does without having any idea of the actual programming involved in the definition.

```
REM  ****  MAIN PROGRAM  ****
PROCtitles
PROCinitialise
REPEAT
  PROCoption_menu
  choice=FNoption
  IF choice="C" PROCcreate_file
  IF choice="E" PROCedit_file
  IF choice="U" PROCupdate_file
UNTIL choice="STOP"
PROCtidy_up
END
REM *** DEFINE ALL PROCEDURES
DEF PROCxxxxxxxx
  ...
  ...
ENDPROC
DEF PROCxxxxxxxx
  ...
  ...
ENDPROC
DEF PROCxxxxxxxx
  ...
  ...
```

Without much difficulty we can see what this program might do. This is shamefully un-typical of most programs we see in magazines, even BEEBUG. It is a reflection of the way in which few people ever sit down to write a program knowing what they want it to do. Just as they get started they suddenly think of something else it could do and squeeze it in where it shouldn't be. Then there are the other problems that they find something doesn't work and decide to remove it. Of course, they can't find all the little bits concerned, and confusing pieces are left where they shouldn't be. The GOTO statement is another 'nasty' leading to over complex programs. The ideal is to avoid GOTO statements as far as possible because they tend to foster bad program structure.

So if you are just starting out in programming, beware of the messy badly designed programs that your forerunners have written. None the least of which is me!!

SOFTWARE UPDATE

In this column we will be bringing you updates and ideas for use with programs in the software library - this will include the word processor package Wordwise.

DISC CONFIGURATIONS

At present all BEEBUGSOFT programs are cassette-based. To load them on a disc machine execute TAB/BREAK (or *TAPE) before loading. Note however that some programs (eg. 'MAGIC EEL') will run out of memory unless you also type PAGE =&E00 <return> before loading. This is necessary because machines with a disc interface fitted use an extra 3K of memory for the disc work-space, unless you reset PAGE in this way.

MASTERFILE

Instructions are given in the documentation of Masterfile to configure it for printer output. As the documentation says, the lines to look for (2000-2999) are in the FIRST program (ie. the header) - A number of members could not find them because they were looking in the main program.

MOON LANDER

Keyboard responses on MOONLANDER can be improved by employing the negative INKEY function. To do this make the following alterations:

```
450 IFINKEY(-82)GOTO610                    470 IFINKEY(-103) YS=YS-0.04:FUEL=FUEL-1
460 IFINKEY(-104) YS=YS+0.04:FUEL=FUEL-1   480 IFINKEY(-68) K=K-0.01:FUEL=FUEL-2
```

WORDWISE IDEAS 1 - (More next month)

When editing text, try using option 5 - Search and Replace (Selective). This will accept very long strings, and spaces are permitted, allowing whole phrases to be corrected. This is much quicker than working through the text with the cursor, and performing manual corrections. You can also press Return on an empty replace string, and then press Escape when Wordwise asks whether to replace it or not. This will leave the cursor at the point required, ready for manual editing. Note that for all operations under option 5, the search is only carried out from the current cursor position. This enables you to work your way through a text very efficiently.

INFO INFO INFO INFO INFO INFO INFO INFO INFO INFO INFO

ON ERROR BUG

Thanks to Eddie Atherton for highlighting this one. There appears to be a small bug associated with the ON ERROR GOTO statement. Try typing in the following mini program exactly as listed, including the REM and comments. Then run it, hit escape to exit and relist it. The result is a corrupted program.

This only happens when the REM and comments immediately follow the GOTO, and actually include a keyword. The computer has attempted to tokenise the keyword ON ERROR, even though it is only a comment.

```
10  ON ERROR GOTO 40: REM ON ERROR EXIT    30  PRINT"this routine"
20  PPRINT"Testing"                        40  END
```

MORE JOYSTICKS

We received a review model of another joystick from CompUtopia Ltd of Leighton Buzzard. They are 'T' shaped joysticks with a 2 dimensional paddle at the top and a small push-button on the stem. Their size and shape make them easy to hold and there is no doubt as to which way round you should use them. The push-button is in a convenient position for 'thumb' operation, though its small size could give rise to a slight 'imprint' on your thumb if you play for an hour or so. Value for money: they are as good as others in the same price bracket, and certainly as sturdy.

Sold as a pair for £14.50 (inc. VAT) plus p & p, they are available from CompUtopia Ltd., 30 Lake St., Leighton Buzzard, Beds. LU7 8RX. Tel: (0525) 376600.

Program tested on 0.1 and 1.10.S.

PROGRAM COMPACTER (Revisited) (16k)

Text by Colin Opie Program by David Tall

In the DECEMBER (BEEBUG vol.1,no.8) issue we had an article on compacting Basic programs. The concept of being able to squeeze large programs into a small space seems to have stirred the imagination of a number of members. Perhaps a craze has been started which will be second only to space invader programs?

David Tall wrote to us a number of times with suggestions on how to improve the initial program. The problem with the published one was that because of the way in which destination line numbers are stored for GOTO, GOSUB, and RESTORE, the program will occasionally corrupt the program to be compacted, because it mistakes a destination line number code for a REM token; though this never happened in our tests. The problem with modifying the compacter was that it would become too long for the area of memory allocated to it.

The 'super-compacter' presented here, written by David Tall, gets around all these problems. As it is in machine code it is fast, will fit below the cassette PAGE boundary, and work regardless of whether or not a disc system is fitted. It will also remove comments from assembler listings. All in all we think that you will like this update on what is an extremely useful utility.

OPERATION

The program copes with three independent choices. When used it offers the choices as follows:

SPACES?	To remove spaces
REMs?	To remove Basic REMs
COMs?	To remove Assembler comments.

If you don't have any assembler code then it will not matter what reply you give to the third choice. Note also that whereas REM deletions will remove the whole REM, deleting assembler comments only affects a comment from the \ symbol up to, but not including, the next separator (ie. a colon). Therefore:

```
10 A=B:REM this is a remark:X=Y
```

reduces to:

```
10A=B
```

but:

```
200 [LDA&70\this is a comment:STA&72]
```

reduces to:

```
200[LDA&70:STA&72]
```

The program is located from &85F to &CFF, using all of the user defined character space (though this is of no consequence since there is no need to RUN the program to be compacted), and part of the user-key buffer space. There is still room to define some keys however, and to make use of this fact the version listed in this article uses 7 of these keys for a purpose:

f0	PAGED MODE 6	f4	*RUN
f1	LIST	f5	RUN
f2	COMPACT	f10	OLD
f3	FREE SPACE		

SETTING UP PACK2

The procedure for setting up this version is simpler than the previous one and is identical for disc or cassette use.

1) Type in the Basic program and SAVE a copy for the sake of safety. A lot of the program is machine code so be very careful as you type it in. Anything after (and including) the '\' character in a line of machine code, up to the end of that line, may be discarded as this is a comment only. It is vitally important that you do type it in correctly because if you do not the program will not run, and no sensible error messages will be displayed.

2) RUN the program.

3) Perform a *SAVE "PACK2" B00 D00 B73

After the above operations you should have two things on tape. One is the assembler listing (or source code), the second is the actual machine code which you will execute each time you want to perform a compaction. It is this second copy which will now be referred to as 'PACK2'.

USING PACK2

There are two ways of running this version. First Load in the program to be compacted. Then you can perform a *LOAD "PACK2" in order to load the compacter into memory, and then use key f2 to compact your program. Alternatively after loading the program to be compacted you could simply do a *RUN "PACK2" which will automatically load (from tape or disc) the machine code, and then run it from the execution address &B73. Incidentally with a disc system (and therefore O.S 1.0) you can in fact just type *PACK2 and the disc operating system will search for PACK2, load it and run it automatically, providing that it is in the designated library area and is not a reserved word (eg. we cannot use the title 'COMPACT'). In this way one can create a series of new commands that can even be used within a Basic program (though PACK2 should NOT be called in this way, for obvious reasons).

Note that the program automatically uses the current value of PAGE to find your program, in order to compact it.

We are most grateful to David Tall for the work which he has done on this program.

```
10 REM PACK2 (c) David Tall 1982
20 REM version 1.20
30 REM (inspired by Graham Taylor's PACK)
40 REM Machine code version
50 REM to reside in pages &B00, &C00
60 REM RUN the program & then
70 REM *SAVE "PACK2" B00 D00 B73
80 REM The *SAVEd program may be *RUN
90 REM to compact BASIC programs.
100 REM Reply Y to SPACEs? to remove
110 REM redundant spaces. <Necessary
120 REM ones such as the one in
130 REM IFA=B C=D  will be kept.>
140 REM Reply Y to REMs? to remove REMs.
150 REM Reply Y to COMMENTS? to delete
160 REM comments in assembler coding.
170 *KEY0
180 *KEY1
190 *KEY2
200 *KEY3
210 *KEY4
220 *KEY5
230 *KEY6
240 *KEY7
250 *KEY8
260 *KEY9
270 *KEY10
280 *KEY0|ML.00|MMO.6:W.19;4;0;|M|N
```

```
  290 *KEY1|M|M|M|ML.|M
  300 *KEY2|MCA.&B73|G|M
  310 *KEY3|MP.;H.-!2A.&FFFF" free"|L|M
  320 *KEY4|M*RUN|M
  330 *KEY5RUN|M
  340 *KEY10OLD|M
  350 $&85F="SPACEs?REMs?COMs?"
  360 FORN%=0TO1:P%=&B73
  370 [OPT3*N%:CLD:LDA#&5E:STA&70:LDA#&B:STA&71:LDX#2:LDY#0
  380 .a LDA(&70),Y:JSR&FFE3:INY:CMP#&3F:BNEa\print options
  390 .x JSR&FFE0:AND#223:CMP#89:BEQy:CMP#78:BNEx\input response Y or N
  400 .y STA&74,X:JSR&FFE3:JSR&FFE7:DEX:BPLa\store response (& print it)
  410     INX:STX&70:STX&72:STX&7C:STX&7D:STX&80:LDX&10:STX&71:STX&73\     store line-start pointers & set ASSEMBLER flag = 0
  420 .b CLC:LDA&72:ADC&7D:STA&72:BCCc:INC&73\start of next input line
  430 .c CLC:LDA&70:ADC&7C:STA&70:BCCd:INC&71\start of next output line
  440 .d LDA#0:LDY#4
  450 .e STA&77,Y:DEY:BPLe\set all flags to zero (except ASSEMBLER)
  460 .f INY:LDA(&70),Y:STA(&72),Y:CPY#3:BEQg:CPY#1:BNEf:CMP#&FF:BNEf:RTS\transfer initial bytes & check if last line
  470 .g STA&7C:STA&7D:INY:STY&7E\store line lengths & output pointer
  480 .h STY&81:LDA(&70),Y:STA&7F\(START TRANSFER LOOP)     store input pointer & current byte
  490     LDX&7B:BNEt:CMP#&8D:BNEt:LDX#4\check for coded numbers (outside quotes)
  500 .G INC&81:JSRP:DEX:BPLG:BMih\if coded, transfer
  510 .t LDX&80:BEQi\if outside ASSEMBLER, go to i
  520     LDX&74:CPX#&59:BNEu:CMP#&5C:BNEu:STX&79\(in ASSEMBLER) check \ & set COMMENT flag accordingly
  530 .u CMP#58:BNEv:LDX#0:STX&79\(in ASSEMBLER) seek colon and turn off COMMENT flag if found
  540 .v LDX&79:BNEj:CMP#93:BNEj:LDX#0:STX&80\(in ASSEMBLER)  if outside COMMENT, seek ] and turn off ASSEMBLER flag as appropriate; in all cases go to j
  550 .i LDX&77:BNEo\(outside ASSEMBLER) if in REM delete, move on
  560     CMP#&22:BNEj:LDA&78:EOR#1:STA&7B:LDA&7F\look for ", change QUOTE flag as necessary
  570 .j LDX&78:BNEp\if inside QUOTES move on to transfer
  580     CMP#91:BNEk:LDX#1:STX&80\(outside QUOTES from here on) if [, set ASSEMBLER flag
  590 .k CMP#&DC:BEQn\DATA?
  600 .m CMP#&F4:BNEo:LDX&75:CPX#&59:BNEn:LDX#1:STX&77\REM? - if found & deletion required, set REM flag
  610 .n LDX#1:STX&7A\set DATA flag (for DATA or REM)
  620 .o LDA&77:ORA&79:BNEq\if REM or COMMENT don't transfer
  630     LDA&7A:ORA&80:BNEp:LDA&7F:CMP#32:BNEp\if DATA  or ASSEMBLER  or not a SPACE, do transfer
  640     LDX&76:CPX#&59:BNEp:LDX#0:STX&82:INY:LDA(&70),Y:JSRsearch:BEQq:DEX:TXA:BCR#1:STA&82\if SPACES are to be deleted, consider following byte
  650     LDY&7E:DEY:LDA(&72),Y:JSRsearch:BEQg:LDX#1:INY\look at previous byte transferred
  660 .A DEX:TXA:ORA&82:STA&82:DEY:LDA(&72),Y:JSRsearch:BNEA:LDX&82:CPX#0:BEQq\search earlier bytes
  670 .p LDY&81:JSRP:BNEz\transfer byte
  680 .q LDY&81:DEC&7D\don't transfer
  690 .z CPY&7C:BCCs\check for end of line and repeat as necessary
  700     LDA&79:BEQw:INC&7D\adjust for COMMENT (1 deletion too many!)
  710 .w LDA&7D:CMP#5:BCCr\if line length less than 5, abort current line transfer
  720     LDY#3:STA(&72),Y:JMPb\else transfer adjusted line length & move to next line
  730 .r JMPc\(abort)
  740 .s INY:JMPh\(next byte)
  750 .search LDX#0:CPY#5:BCCF:CMP#&30:BCCF:CMP#&3A:BCCN:CMP#&40:BCCF:CMP#&5B:BCCL:CMP#&5F:BCCF:CMP#&7B:BCSF
  760 .L INX
  770 .N INX
  780 .F LDA#&20:CPX#0:RTS\consider byte, X = 1 (number), = 2 (letter),  = 0 (otherwise)
  790 .P LDA(&70),Y:LDY&7E:STA(&72),Y:INC&7E:LDY&81:RTS\(transfer):]
  800 NEXT:END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SIMPLE MUSICAL KEYBOARD

J C Fenton writes in with a novel single line program which turns your keyboard into a piano style keyboard. The program line supplied makes the QWERTY and ZXCVBN rows into the white keys, and the respective row above these into the black keys:

```
CLS:REPEAT:G$=GET$:PRINTG$:SOUND&11,-15,21+4*INSTR("Q2W3ER5T6Y7UI9O0PZSXDCFVBHNJM,L.;/",G$),10:UNTIL G$="1":SOUND&11,0,0,0
```

Program tested on 0·1 and 1·1 O.S.

MORE CASSETTE LOAD PROBLEMS

David Coups, like many other members has had problems with the Ferguson 3T07 recorder. He wrote to Ferguson who carried out the following modification to his recorder.

Disconnect C105 (.02.uF) and add a 100k ohm 1/4 watt resistor in series with the instruments input circuit. This resistor is best added in the connecting lead going into the tape recorder.

Many cassette loading problems are caused by tape oxide deposited on the heads. This problem will be particularly bad when poor quality tapes are used. The solution is simple; use good quality tapes and clean your heads and tape path regularly with a cassette head cleaning kit available from any Hi-Fi shop. Heading cleaning cassettes are NOT recommended.

If you still have problems after this then this tip from Mr. C.C. Evans may be what you need. He found that use of a 'Tape head demagnetiser' solved his loading problems. Those members who have reported their Welcome tape seeming to 'erase' after a few plays may find this to be the answer.

Program tested on 0.1 and 1.1 O.S.

SINGLE KEY MEMORY DISPLAY (16k)

by Alan Cocker

Thanks to Alan Cocker for sending in this well written program. This is a very small program which will quickly and clearly display in hex and character (ASCII) format any desired area of memory. The program is so small that Alan has strung it together as a single line program which can be entered within a function key. This makes it extremely handy as it can be loaded and stored totally independently of any other program.

INSTRUCTIONS FOR USE

1) Type the program in exactly as listed below, and run it.

2) To save a copy of the program, which is now stored within the function key buffer, use *SAVE "DISPLAY" BØØ CØØ

3) To reload the program at any time use *LOAD "DISPLAY". This will reload the program into the function key buffer again, without disturbing any other program you may have in the computer.

4) To run the program hit the red function key 8. The screen will clear and you will be prompted for a "Start" address. This may be entered in decimal, hex or as any valid expression. (To enter a hex number, precede it with &). Then hit return and a ? will appear on the screen. This is the prompt for the number of bytes that you want displayed. Once again this may be decimal, hex or an expression.

The display will then commence. Each line is preceded with the address in hex, of the first byte on the line. This is followed by 8 bytes displayed in hex, and then in ASCII. If the ASCII display is an unprintable character, it is displayed as a full stop. This default unprintable character may be altered by inserting the value of the required character in place of the "46" in the program. The program does not use paged mode, so if more than a full screenfull of display is created, use CTRL and SHIFT keys, held down together to allow scrolling through the display. If you require a printout of the display, simply activate the printer in the normal way before hitting function key 8.

NOTE A tip for typing the program in. The character string in the last line is (R<32 OR R> etc ..ie the character after 32 is not zero.

Program listing:

```
10*KEY8 MO.7:I."START, >"S$,N$:S=EV.S$:N=EV.N$:F.I=S TOS+N S.8:@%=4:P.~I;:B$="  "
:F.J=ØTO7:R=I?J:P.~R;:@%=3:R=R+(R<32 ORR>126)*(R-46):B$=B$+CHR$R:N.:P.B$:N.:@%=10|M
```

PROGRAM EXPLANATION

```
Mo.7:                          Set Mode for display
Input "Start, >" S$,N$         Get Start Address and Number
S=Eval (S$):                   Evaluate Start (in case hex)
N=Eval (N$):                   Evaluate Start (in case hex)
For I=S To S+N Step 8:         Loop round memory requested
@%=4:                          Set for correct printing
Print ~I;:                     Print address of 1st byte hex
B$="":                         Initialise ASCII store
For J=Ø To 7:                  Loop round each byte on line
R=I?J:                         Get contents of byte
```

```
Print ~R;:                          Print contents in hex
@%=3:                               Set for correct printing
R=R+(R<32 OR R>126)*(R-46):         Use "." (46) if out of range
B$=B$+Chr$(R):                      Add into ASCII store
Next J:
Print B$:                           Print ASCII at end of line
next I%
@%=10                               Reset printing to normal
!M
```

Example output:

The output shown below is a display of the start of the function key buffer. As the program in this article was the first function key program to run after a cold start, you can see it stored at the beginning of the buffer:

```
 START, >&B00
?100
 B00  AB AB AB AB AB AB AB AB  ........
 B08  10 AB AB AB AB AB AB AB  ........
 B10  AB 4D 4F 2E 37 3A 49 2E  .MO.7:I.
 B18  22 53 54 41 52 54 2C 20  "START,
 B20  3E 22 53 24 2C 4E 24 3A  >"S$,N$:
 B28  53 3D 45 56 2E 53 24 3A  S=EV.S$:
 B30  4E 3D 45 56 2E 4E 24 3A  N=EV.N$:
 B38  46 2E 49 3D 53 20 54 4F  F.I=S TO
 B40  53 2B 4E 20 53 2E 38 3A  S+N S.8:
 B48  40 25 3D 34 3A 50 2E 7E  @%=4:P.~
 B50  49 3B 3A 42 24 3D 22 20  I;:B$="
 B58  20 22 3A 46 2E 4A 3D 30   ":F.J=0
 B60  54 4F 37 3A 52 3D 49 3F  TO7:R=I?
```

POSTBAG

re:GAMES PROGRAMS

Dear BEEBUG,

I am writing to congratulate you on your program "ASTRO-TRACKER". Myself and my friends have had BBC micros for about a year now and have been playing a lot of the Acornsoft/BEEBUG games, our high scores so far are as follows:

SNAPPER about 128,000
ARCADIANS 8670
DEFENDER (PLANETOID) around the 8 million mark. (Yes, 8,000,000).
ASTRO-TRACKER about 13,000

John Penfield.

[Ed: can anyone bear 8 million at PLANETOID? We have already exceeded the ARCADIANS score with 15,000 and ASTRO-TRACKER with 22,0001.

re:MAGIC EEL

Dear Sirs,

I recently bought a copy of GAMES 4 'MAGIC EEL'. It does not run on my machine. I have a Model B which appears to be fitted with the 1.0 O.S., and the switch-on message includes the message 'DFS'. After some days of worry and experimentation I can now load cassettes by using the *TAPE command. Could this affect the 'MAGIC EEL' program? The program appears to load perfectly and presents the page of instructions. However, when the SPACE bar is pressed the machine reverts to command mode. Any suggestions ?

R.A. Walker.

[Ed: The loading of 'MAGIC EEL' using *TAPE on a disc system does NOT affect it adversely. However, the program requires graphics MODE 1 to run, and the above effect occurs because you have not reset PAGE to &E00. Please see BEEBUG vol.1 no.7 p.27 (PAGE change hint) and BEEBUG vol.1 no.8 p.38 (Points Arising)].

re:USER PORT

Dear Sirs,

I was very interested in your articles on the user port in BEEBUG nos.3 and 4. In the article in no.4 you mentioned the IC 74LS17, in the accompanying diagram it was given as 7416. Whichever it is (and perhaps it does not matter ?), I cannot find a supplier of it. Can you help me?

Torben Hesselbo.

[Either device will do the job but you can use a non-LS type which may be more readily available in shops].

Program tested on 0.1 and 1.1 O.S.

FIVE-DICE (16k)

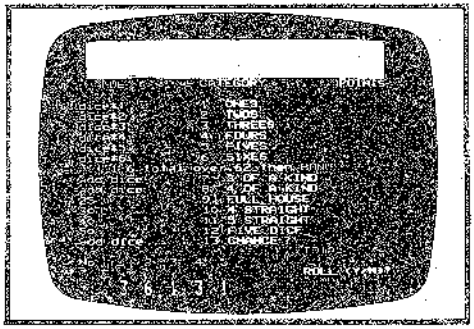
by Brian Kilby

This program simulates the dice game "YAHTZEE" (r), in which five dice are thrown and a score is given for their value, depending on the category in which the player puts it. The game requires a fair amount of skill and judgement, although luck does play its part. The program uses coloured teletext graphics in an exemplary way.

You are allowed a maximum of three re-throws on any number of dice, and then you must place the result in one of the categories listed.

Full instructions can be listed when the program is first run (the instructions appear in a novel way), and a high score is kept. If you are unfamiliar with the game, do not be daunted by the rules, these are soon picked up. We found this game good fun to play.

[NOTE: when playing, if you get 5 numbers of the same value eg 66666 this CANNOT be put down as a full house.]



PROCEDURES

init	Initialises variables, clears screen.
screen	Prints the title and categories.
roll	Throws dice and allows two more goes, calls FNchoice, calls test subroutine, and awards points.
FNtest1(number%)	Checks if dice (1-5)=number% and returns value of quantity*number%, tests for categories 1-6.
FNtest11	Sorts dice into numerical order and checks if either 12345 or 23456 exist.
FNtest10	Checks if dice equal 1234, 2345 or 3456, other dice can be any value.
FNtest9	Checks for a 'Full House' (doesn't allow five of a kind).
FNtest7(number%)	Tests for 3,4, or 5 of a kind.
FNadd	Returns the total value of the dice.
sort	'Bubble' sorts the dice into numerical order.
throw(dice$)	Assigns new value to dice chosen.
clear	Clears the screen.
bonus	If categories 1-6 add up to more than 62, then a bonus of 30 is added to the scoresheet.
choice	Main part. Asks if another roll is required, which category to score, and returns value of category.
dice	Prints dice in double height characters.
end	Prints total score and best score.
wait	Waits for the 'space' bar to be pressed.
read(N)	Reads N lots of data, and prints it at slow speed for the instructions.

```
   10 REM"FIVE DICE  by Brian Kilby
   20 MODE7
   30 ON ERROR GOTO 1700
   40 B%=RND(-TIME):B%=0
   50 IF FNyes("DO YOU WANT THE RULES ?",152,5,12)TH
EN PROCread(5):PROCread(10)
   60 PROCinit
   70 PROCscreen
   80 REPEAT
   90 PROCroll
  100 UNTIL cat%=13
  110 PROCend
  120 IF FNyes("New game ?",136,27,22) THEN CLEAR:GO
TO 60
  130 CLS:END
  140 REM"INITIALISE
  150 DEF PROCinit
  160 DIM dice%(5),cat%(13)
  170 VDU23;8202;0;0;0;
  180 VDU15,12
  190 cat%=0:call%=0:bonus%=0
  200 ENDPROC
  210 REM"SCREEN
  220 DEF PROCscreen
  230 VDU12,130,157,10,13,136,157,131,141,10,13,130,
157,131,141,10,13,130,157,132,31,38,0,156,8,10,156,8
,10,156,8,10,156
  240 FORI%=1TO2:PRINTTAB(35,I%)"FIVE DICE":NEXT
  250 IF B%>0 THEN PRINTTAB(15,3);"Best  ";B%
  260 PRINTTAB(0,4)CHR$134;"VALUE"TAB(15)"CATEGORY"T
AB(32)"POINTS"
  270 FORI%=1TO5:PRINT"dice";I%;CHR$133;TAB(15);I%;
CHR$135:NEXT
  280 PRINTTAB(18,6)"ONES"
  290 PRINTTAB(18,7)"TWOS"
  300 PRINTTAB(18,8)"THREES"
  310 PRINTTAB(18,9)"FOURS"
  320 PRINTTAB(18,10)"FIVES"
  330 PRINTTAB(18,11)"SIXES"
  340 PRINTCHR$134;"If 1-6 total over 62 then"CHR$12
9;"BONUS"
  350 FORI%=3TO4:PRINT"add dice"TAB(18);I%;" OF A KI
ND":NEXT
  360 PRINT;25TAB(18)"FULL HOUSE"
  370 PRINT;30TAB(18)"4 STRAIGHT"
  380 PRINT;40TAB(18)"5 STRAIGHT"
  390 PRINT;50TAB(18)"FIVE DICE"
  400 PRINT"add dice"TAB(18)"CHANCE"
  410 PRINTTAB(26);CHR$129;"TOTAL"
  420 FORI%=13TO19:PRINTTAB(14,I%)CHR$133;I%-6;CHR$1
35:NEXT
  430 PRINTTAB(0,21);"No. ";CHR$130;"1  2  3  4  5";
  440 FORI%=5TO20:VDU31,32,I%,131:NEXT
  450 FORI%=23TO24:VDU31,0,I%,141:NEXT
  460 ENDPROC
  470 REM"MAIN SECTION
  480 DEF PROCroll
  490 go%=0
  500 PROCthrow("12345")
  510 Q=FNchoice
  520 ON Q GOSUB 560,560,560,560,560,560,600,600,610
,620,630,640,650
  530 gt%=gt%+points%
  540 PRINTTAB(35,5+Q DIV7+Q);points%
  550 ENDPROC
  560 points%=FNtest1(Q):call%=call%+1
  570 bonus%=bonus%+points%
  580 IF call%=6 THEN PROCbonus
  590 RETURN
  600 points%=ABS(FNtest7(Q-4)*FNadd):RETURN
  610 points%=FNtest9:RETURN
  620 points%=FNtest10:RETURN
  630 points%=FNtest11:RETURN
```

```
  640 points%=ABS(FNtest7(Q-7)*50):RETURN
  650 points%=FNadd:RETURN
  660 REM"CAT. TESTS
  670 REM"ONES to SIXES
  680 DEF FNtest1(number%)
  690 LOCAL total%,I%
  700 total%=0
  710 FORI%=1TO5:total%=total%+(dice%(I%)=number%):N
EXT
  720 =ABS(total%)*number%
  730 REM"5 STRAIGHT
  740 DEF FNtest3
  750 LOCAL test$,I%
  760 PROCsort
  770 test$=""
  780 FORI%=1TO5:test$=test$+STR$(dice%(I%)):NEXT
  790 IF test$="12345"OR test$="23456"THEN =40 ELSE
=0
  800 REM"4 STRAIGHT
  810 DEF FNtest4
  820 IF (FNtest1(3)>=3 AND FNtest1(4)>=4)AND((FNtes
t1(2)>=2 AND(FNtest1(1)>=1 OR FNtest1(5)>=5))OR(FNte
st1(5)>=5 AND FNtest1(6)>=6)) THEN =30 ELSE =0
  830 REM"FULL HOUSE
  840 DEF FNtest5
  850 PROCsort
  860 LOCAL flag%
  870 flag%=((dice%(1)=dice%(2))AND(dice%(2)=dice%(3
)))AND(dice%(4)=dice%(5))OR((dice%(1)=dice%(2))AND(d
ice%(3)=dice%(4))AND(dice%(4)=dice%(5)))
  880 IF flag%<>FNtest7(5) THEN =25 ELSE =0
  890 REM"3-4-5 OF A KIND
  900 DEF FNtest7(number%)
  910 LOCAL I%,X%,total%
  920 FORI%=1TO6
  930 total%=0
  940 FORX%=1TO5
  950 IF dice%(X%)=I% THEN total%=total%+1
  960 NEXT
  970 IF total%>=number% THEN I%=7:=TRUE
  980 NEXT:=FALSE
  990 REM"ADD DICE
 1000 DEF FNadd
 1010 LOCAL I%,total%
 1020 FORI%=1TO5:total%=total%+dice%(I%):NEXT:=total%
 1030 REM"SORT
 1040 DEF PROCsort
 1050 LOCAL Y%,Z%,T%
 1060 FORY%=1TO4
 1070 FORZ%=Y%+1TO5
 1080 IF dice%(Y%)<=dice%(Z%)THEN 1090 ELSE T%=dice%
(Y%):dice%(Y%)=dice%(Z%):dice%(Z%)=T%
 1090 NEXT:NEXT:ENDPROC
 1100 REM"RE-THROW DICE
 1110 DEF PROCthrow(dice$)
 1120 LOCAL I%
 1130 FORI%=1TO LEN(dice$)
 1140 flag%=VAL(MID$(dice$,I%,1)):IF flag%>5 THEN fl
ag%=0:I%=6:ENDPROC
 1150 dice%(flag%)=RND(6)
 1160 NEXT
 1170 PROCdice:ENDPROC
 1180 REM"CLEAR MESSAGE
 1190 DEF PROCclear
 1200 PRINTTAB(27,22);CHR$133;SPC(11)
 1210 PRINTTAB(28,23)SPC(10)
 1220 ENDPROC
 1230 REM"BONUS
 1240 DEF PROCbonus
 1250 IF bonus%>62 THEN bonus%=30:gt%=gt%+30 ELSE bo
nus%=0
 1260 PRINTTAB(35,12);bonus%
 1270 ENDPROC
 1280 REM"YES OR NO
 1290 DEF FNyes(Q$,col%,X%,Y%)
 1300 LOCAL reply$,col%
 1310 PRINTTAB(X%,Y%)CHR$(col%);Q$;
 1320 REPEAT:reply$=CHR$(GET AND &DF):UNTIL reply$="
Y" OR reply$="N"
 1330 PROCclear
 1340 =(reply$="Y")
 1350 REM"CHOICES
 1360 DEF FNchoice
 1370 REPEAT:go%=go%+1
 1380 SOUND1,-12,200,5
 1390 IF NOT FNyes("ROLL (Y/N)?",135,27,22) THEN go%
=2:GOTO 1490
 1400 REPEAT
 1410 REPEAT
 1420 SOUND1,-10,150,5
 1430 PRINTTAB(27,22)CHR$130;"WHICH DICE"
 1440 INPUTTAB(28,23)Q$
 1450 PROCclear
 1460 UNTIL Q$>CHR$47 AND Q$<CHR$54
 1470 PROCthrow(Q$)
 1480 UNTIL flag%
 1490 UNTIL go%=2
 1500 REPEAT
 1510 REPEAT
 1520 SOUND1,-10,100,5
 1530 PRINTTAB(27,22)CHR$133;"CATEGORY?"
 1540 INPUTTAB(28,23)"Q
 1550 PROCclear
 1560 UNTIL Q>0 AND Q<14
 1570 UNTIL cat%(Q)=0
 1580 cat%(Q)=1:cat%=cat%+1:=Q
 1590 REM"PRINT DICE
 1600 DEF PROCdice
 1610 FORflash%=152TO135STEP-17
 1620 FORI%=1TO2:PRINTTAB(4,22+I%);CHR$(flash%);:FOR
N%=1TO5:PRINT;dice%(N%)SPC2;:NEXT:PRINTCHR$146;:NEXT
 1630 IF flash%=152 THEN TIME=0:REPEATUNTIL TIME>50
 1640 NEXT:ENDPROC
 1650 REM"END RESULT
 1660 DEF PROCend
 1670 PRINTTAB(34,20);CHR$136;gt%;
 1680 IF gt%>B% THEN B%=gt%:PRINTTAB(15,3);"Best  ";
B%
 1690 ENDPROC
 1700 REM"ERROR TRAP
 1710 CLS
 1720 IF ERR<>17 THEN REPORT:PRINT" in line ";ERL
 1730 REM"CURSOR ON ! SIZE
 1740 VDU23,0,10,112;0;0;0;
 1750 END
 1760 REM"RULES
 1770 REM"Implanted control chr.'s
 1780 DATA"In the following game the object is to  s
core as high as possible,by skillfully placing the r
esults obtained from the  dice, into the various ca
tegories."
 1790 DATA"There areFIVE DICE."
 1800 DATA"You are allowed a maximum of three goes f
or each category (although you can stopafter the fir
st or second roll)"
 1810 DATA"You may re-throw ANY or ALL the dice as y
ou see fit,by typing out the relevant  dice numbers
(1 to 5 in any order)"
 1820 DATA"When you are satisfied with the results (
or after the third throw which ever is first) you MU
ST place them in one of the13 categories."
 1830 DATA"CATEGORIES:"
 1840 DATA"ONES to SIXES :every dice that has thes
ame value as the cat. chosen is added  together eg.1
1223=4(2+2) in TWOS"
 1850 DATA"3 OF A KIND   :any 3 dice the same    s
core value of all 5 dice"
 1860 DATA"4 OF A KIND   :any 4 dice the same    s
core value of all 5 dice"
 1870 DATA"FULL HOUSE    :3 of one number AND 2  o
f another scores 25 points"
 1880 DATA"4 STRAIGHT    :either 1234 or 2345 or 3
456 score 30 points (any order)"
 1890 DATA"5 STRAIGHT    :either 12345 or 23456  s
cores 40 points (any order)"
 1900 DATA"FIVE DICE     :all the same ? score 50p
oints"
 1910 DATA"CHANCE        :anything goes here so  s
```

```
core total value of dice."
 1920 DATA"When asked'CATEGORY?'enter the NUMBER o
f the category (colour coded magenta)"
 1930 REM"WAIT
 1940 DEF PROCwait
 1950 PRINT"press space bar."
 1960 REPEAT UNTIL GET=32
 1970 CLS:ENDPROC
 1980 REM"SLOW PRINTOUT
 1990 DEF PROCread(N)
 2000 LOCAL T,letter,word,word$
 2010 VDU12,10
 2020 FORword=1 TO N
 2030 READ word$
 2040 FORletter=1 TO LEN(word$)
 2050 PRINTMID$(word$,letter,1);
 2060 TIME=0:REPEAT UNTIL TIME>8
 2070 NEXT
 2080 TIME=0:REPEAT UNTIL TIME>250
 2090 PRINT:NEXT
 2100 PROCwait
 2110 ENDPROC
```

**NOTE - In the game of FIVE-DICE some of the instructions (held in DATA statements) are in fact colour coded teletext characters. The lines concerned are 1790, and 1830-1920. In the original program they are mostly GREEN with the odd one in MAGENTA (see listing). If you would like to keep this feature or use different coloured teletext characters, then do one of the following: If you have 0.1 OS see the article in BEEBUG v1 no4 p7; if you have a series 1 OS then see the hint in this issue.

POINTS ARISING

PACK article (BEEBUG v.1 no.8 p.12)

There is a problem with the Compacter Program published in the last issue. See the article 'Compacter revisited' in this issue, for an update on this program.

OSBYTE hint (BEEBUG v.1 no.8 p.41)

In the last issue we gave details of the useful OSBYTE 135 to find out what screen mode you were in. Contrary to what we said, this DOES work on O.S 0.1.

BAD PROGRAM article (BEEBUG v.1 no.8 p.4)

Under the heading 'THE PROGRAM' all references to line numbers in the SECOND paragraph must be incremented by 10, (ie. 'line 110' should read 'line 120').

BREAKOUT

For improved colour sequences on Breakout change line 20 to 20 VDU 19,1,1,0,0,0 When using the 16k version of the game do NOT delete line 1330 (contrary to instructions). Note this only affects error handling.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

COLOURED LISTINGS (TELETEXT)

In issue 4 of BEEBUG (p.7) there is an article on teletext characters for the (then) 0.1 OS. On a 1.0 OS coloured or flashing characters can be obtained in mode 7 merely by using the SHIFT key in conjunction with the red user keys (ie. by pressing SHIFT and a user key simultaneously). This can be used to good effect in the documenting of programs.

MOVING THE GRAPHICS ORIGIN

There is a built in command to shift the graphics origin on the Beeb. After executing VDU 29,X;Y; all PLOTs and DRAWs will be shifted by the amount X,Y. This can be extremely useful, and we are grateful to Paul Livingstone for this one. [We used this command in the very first issue of BEEBUG in the "3-D Noughts & Crosses" program in line 1770, but never included it as a 'hint & tip'. Ed.]

DISC-CASSETTE BREAK

If you have a disc filing system (DFS) ROM fitted, then it is well known that pressing CTRL/BREAK (ie. pressing CTRL and BREAK simultaneously) will cause a cold system restart to the DFS. Did you know that pressing T/BREAK or TAB/BREAK will cause a restart to the cassette filing system (CFS)? Thanks to K.Simpson for this hint.

It is true that pressing TAB/BREAK has the same effect as a CFS restart, but the value of PAGE remains set for the DFS, ie. at &1900.

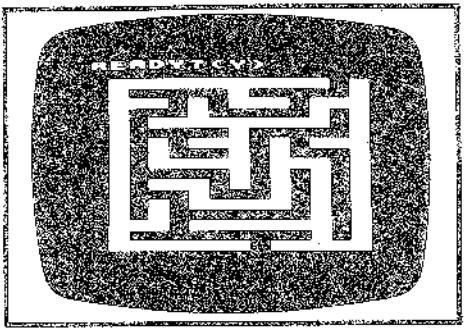
Program tested on 0·1 and 1·1 O.S.

BEEBMAZE (32k)

by R R Hull

It makes a change to have a program that doesn't involve death and destruction. This game retains some of the excitement of arcade type games, but gives you time for thought between each move, and can be quite addictive.

The program creates a random maze, and you have to find your way through to the exit. You are awarded a score depending on several factors. The interest comes because the screen displays the view that you would see if you were actually within the maze itself. The graphics are good, but you need both a good memory and sense of direction to get through the maze in record time.



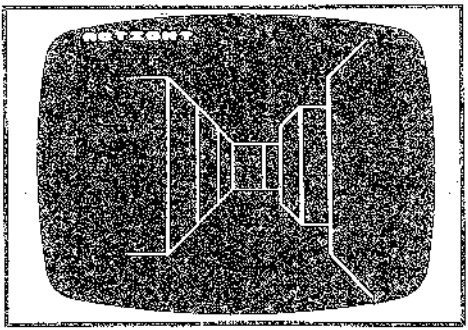
When you get hopelessly lost, which you will to begin with, you can ask for a plan view of the maze (you lose points though) so that you can find out precisely where you are.

You simply control your movements with L R F A M which are turn left, right, move forwards, about turn, and request map.

There are two versions of the game, one timed and the other untimed. In the timed version you start with a certain amount of points, and lose them as follows:-

1 point per elapsed second
2 points per move
10 points for hitting a wall
50 points for leaving via the entrance
100 points for requesting a map

You GAIN 2?% points for passing through the exit.



Finally if you want to change the range of sizes of the generated mazes study line:-

1090 line=20:width=RND(4)+4:height=RND(6)+7

You need to alter 'width' and 'height'. For example as it stands it will generate random widths from 5 to 8. Alter it to width=RND(2)+8 and it will generate widths from 9 to 10 only. The same applies to height. (Keep the number in brackets between 2 and 8). You can, of course, remove the random function altogether.

Please note that while this program works very well, and is good graphically, it does employ some rather unusual techniques - such as the 'pokes' and use of HIMEM at line 1140 etc. We would not normally advise the use of these in such a program because they obscure understanding of the program operation, and prevent it from working across the Tube.

Please note, if you are using a disc system that you will sometimes run out of memory. You could either remove the instruction lines, or use the 'Move down' routine published elsewhere in this issue.

```
  10 REM***   BEEBMAZE   ***
  20 REM*** BY R.R. HULL ***
  30 REM***     1/9/82     ***
  40 MODE 7:?&FE00=10:?&FE01=32
  50 PROCtitle
  60 REM*******INSTRUCTIONS**
  70 D$=CHR$134:PRINT'D$"    The computer draws a 3
_ D maze of"
  80 PRINTD$"random size.  You start just inside it"
  90 PRINTD$"with the entrance directly behind you."
 100 PRINTD$"     Your objective is to find your way"
 110 PRINTD$"through the maze and out via the exit."
 120 PRINTD$"  You move in the direction in which"
 130 PRINTD$"you are facing.  You have a choice of"
 140 PRINTD$"two  games, timed(TI) with a score(SC)"
 150 PRINTD$"and, untimed & unscored.  In the timed"
 160 PRINTD$"version you start with a number of"
 170 PRINTD$"points, which depend on the maze size."
 180 PRINTD$"Points are lost or won as shown:-"
 190 D$=CHR$131
 200 PRINTD$"-1 point per elapsed second"
 210 PRINTD$"-2 points per move (or turn)"
 220 PRINTD$"-10 points for hitting a wall"
 230 PRINTD$"-50 for trying to leave via the way in"
 240 PRINTD$"-100 for asking for a map."
 250 PRINTCHR$129"+200 for escaping via the exit."
 260 PRINT'CHR$132CHR$136"DO YOU WANT TIMED(T) OR UN
TIMED(U)?"
 270 A$=GET$:IF A$<>"T" AND A$<>"t" AND A$<>"U" AND
A$<>"u" THEN 270
 280 CLS:IF A$="T" THEN time%=1 ELSE time%=0
 290 PROCtitle
 300 PRINT''CHR$134"  You move around the maze using
 :-"
 310 PRINT''TAB(7)"Key - F to move forward"
 320 PRINTTAB(7)"     - A to turn about"
 330 PRINTTAB(7)"     - L to turn left"
 340 PRINTTAB(7)"     - R to turn right"
 350 PRINT'TAB(3)CHR$129"In emergencies only"CHR$135
"- M for map."
 360 PRINT'TAB(6)CHR$133CHR$136"PRESS SPACE BAR TO S
TART"
```

```
  370 A$=GET$:IF A$<>" " THEN 370
  380 REM*MODE CHANGE/MAP GRAPHICS
  390 MODE 5:?&FE00=10:?&FE01=32
  400 VDU 23,224,0,24,60,126,24,24,24,0
  410 VDU 23,225,0,8,12,126,126,12,8,0
  420 VDU 23,226,0,24,24,24,126,60,24,0
  430 VDU 23,227,0,16,48,126,126,48,16,0
  440 VDU 23,228,255,255,255,255,255,255,255,255
  450 VDU 19,1,6;0;19,3,2;0;
  460 ON ERROR GOTO 2250
  470 COLOUR 1:PRINTTAB(4,14)"PLEASE WAIT"
  480 PROCinit
  490 REM*MAIN PROG/MOVE/TURN*
  500 DEF PROCkey
  510 *FX11,0
  520 *FX15,0
  530 PRINTTAB(0,0)"                    "
  540 COLOUR 1:PRINTTAB(0,0)"ACTION?"
  550 A$=INKEY$(0):PROCtime:IF A$=""THEN 550
  560 N%=P%:Q%=?(P%):IF A$="F" OR A$="f" THEN N%=P%+d
ir(M%)
  570 COLOUR 3:AD%=AD%+2
  580 IF N%=P% THEN 630
  590 IF P%=out% AND M%=3 PRINTTAB(0,0)"HURRAH - YOU
ESCAPE!":AD%=AD%-200:PROCtune:PROClongdelay:
PROCagain
  600 IF P%=in% AND M%=1 PRINTTAB(0,0)"WAY IN - NO EX
IT":AD%=AD%+50:SOUND1,-15,20,20:GOTO 630
  610 IF Q%-D%*INT(Q%/D%)<>0 THEN P%=N%:PRINTTAB(0,0)
"YOU MOVE":GOTO 710
  620 PRINTTAB(0,0)"YOU HIT A WALL":AD%=AD%+10:SOUND1
,-15,200,3
  630 R%=M%
  640 IF A$="R" OR A$="r" THEN R%=M%+1:C$="RIGHT"
  650 IF A$="A" OR A$="a" THEN R%=M%+2:C$="ABOUT"
  660 IF A$="L" OR A$="l" THEN R%=M%+3:C$="LEFT"
  670 IF A$="M" OR A$="m" THEN AD%=AD%+100:PROCmap
  680 IF R%=M% THEN 710
  690 IF R%>3 THEN R%=R%-4*INT(R%/4)
  700 M%=R%:PRINTTAB(0,0)"YOU TURN "C$
  710 D%=wall(M%):Q%=?(P%):Y%=P%:left%=M%-1:right%=M%
+1
  720 IF left%<0 THEN left%=3
  730 IF right%>3 THEN right%=0
  740 PROCcell
  750 PROCdelay
  760 PROCthree_d
  770 PROCkey
  780 ENDPROC
  790 REM***INSPECT CELL WALLS***
  800 DEF PROCcell
  810 LOCAL L,R
  820 L=wall(left%):R=wall(right%)
  830 lwall=Q%-L*INT(Q%/L)
  840 rwall=Q%-R*INT(Q%/R)
  850 cwall=Q%-D%*INT(Q%/D%)
  860 ENDPROC
  870 REM**MOVE/LOOK AT NEXT CELL
  880 DEF PROCnextcell
  890 Y%=Y%+dir(M%):Q%=?(Y%)
  900 ENDPROC
  910 REM**PLAN MAZE-IN & OUT***
  920 DEF PROCmakemaze
  930 LOCAL A,C,D,E,I,M,J,R,T,V
  940 C=0
  950 P%=P%+1:IF P%>E% THEN P%=S%
  960 A=RND(4)-1:D=0
  970 A=A+1:D=D+1:IF D>3 THEN 950
  980 IF A>3 THEN A=0
  990 M%=P%+dir(A):IF M%<S% OR M%>E% THEN 970
 1000 V=?(P%):M=?(M%):IF C>0 AND V=210 THEN P%=M%:GOT
O 960
 1010 T=M%-S%:IF(V=M OR M<210)AND C>0 THEN 970
 1020 E=T-width*INT(T/width):IF(E=0 AND dir(A)=1)OR(E
=G% AND dir(A)=-1):GOTO 970
 1030 R=INT(15/wall(A)):V=V/wall(A):?P%=V:M=M/R:?M%=M
 1040 P%=M%:C=C+1:IF C<H% THEN 960
 1050 J=?(out%):J=J/wall(3):?out%=J:I=?(in%):I=I/wall
(1):?in%=I:M%=in%:P%=M%:M%=3:M%=P%:TIME=0:GOTO 710
 1060 ENDPROC
 1070 REM*VARIABLES/MAZE SIZE**
 1080 DEF PROCinit
 1090 line=28:width=RND(4)+4:height=RND(6)+7
 1100 DIM wall(3),dir(3),X(10),Y(10)
 1110 wall(0)=5:wall(1)=7:wall(2)=3:wall(3)=2
 1120 dir(0)=1:dir(1)=width:dir(2)=-1:dir(3)=-width
 1130 H%=width*height-1:G%=width-1
 1140 HIMEM=22420:S%=HIMEM:E%=S%+H%:FOR A=S% TO E%:?A
=210:NEXT:P%=S%+RND(H%+1)
 1150 out%=S%+(RND(width)-1):in%=E%-(RND(width)-1):AD
%=0
 1160 PROCmakemaze
 1170 ENDPROC
 1180 REM***MAP ROUTINE*******
 1190 DEF PROCmap
 1200 CLS:LOCAL A,B,C,D,F,G,H,I,L,M,S,map,cell
 1210 A=31864:B=A:S=S%:F=S%+G%
 1220 FOR map=S TO F:FOR cell=0 TO 3:C=?(map):C=C-wal
l(cell)*INT(C/wall(cell))
 1230 L=1:IF cell=1 OR cell=3 THEN L=line
 1240 IF cell>1 THEN L=-L
 1250 M=line/L:H=A+L:G=H+M:I=H-M
 1260 IF map=P% PROCarrow(A)
 1270 IF C=0 PROCwall(G):G=H:PROCwall(G):G=I:PROCwall
(G)
 1280 NEXT:A=A+2:NEXT:S=S+width:F=F+width:A=B+(2*line
):B=A
 1290 IF F<=E% THEN 1220
 1300 PROCdelay
 1310 COLOUR 1:PRINTTAB(0,0)"READY?(Y)":A$=INKEY$(0):
PROCtime:IF A$<>"Y" AND A$<>"y" THEN 1310
 1320 ENDPROC
 1330 REM***DRAW MAP WALLS****
 1340 DEF PROCwall(G)
 1350 COLOUR 2
 1360 LOCAL X,Y
 1370 Y=INT((G-31800)/line)
 1380 X=G-31802-line*Y
 1390 PRINTTAB(X,Y);CHR$228
 1400 SOUND1,-10,X*Y,1
 1410 PROCtime
 1420 ENDPROC
 1430 REM*CHOOSE & DRAW ARROW*
 1440 DEF PROCarrow(A)
 1450 COLOUR 3
 1460 LOCAL D,X,Y
 1470 IF R%=0 D=225
 1480 IF R%=1 D=226
 1490 IF R%=2 D=227
 1500 IF R%=3 D=224
 1510 Y=INT((A-31800)/line)
 1520 X=A-31802-line*Y
 1530 PRINTTAB(X,Y);CHR$D
 1540 ENDPROC
 1550 REM***CHANGE CELL DEPTH*
 1560 DEF PROCthree_d
 1570 CLS
 1580 X=180:XX=1120:Y=1000:YY=10
 1590 CW=1.4:CH=1.4:X1=158:Y1=158
 1600 depth=0
 1610 depth=depth+1
 1620 X=X+X1:XX=XX-X1:Y=Y-Y1:YY=YY+Y1
 1630 X1=X1/CW:Y1=Y1/CH
 1640 PROCdraw_maze
 1650 IF Y%=in% AND M%=1 OR Y%=out% AND M%=3 PROCkey
 1660 IF cwall=0 PROCkey
 1670 PROCnextcell
 1680 PROCcell
 1690 IF depth<5 GOTO 1610
 1700 PROCkey
 1710 ENDPROC
 1720 REM***DRAW 3-D MAZE*****
 1730 DEF PROCdraw_maze
 1740 PROCtime
 1750 GCOL0,2
 1760 REM***DRAW VERTICALS****
 1770 MOVE X,Y:DRAW X,YY:MOVE XX,Y:DRAW XX,YY
 1780 REM*CHANGE X,Y FOR CELL*
 1790 X2=X-X1*CW:X3=XX+X1*CW
```

```
1800 Y2=Y+YI*CH:Y3=YY-YI*CH
1810 REM***DRAW HORIZONTALS**
1820 IF ewall=0 MOVE X,Y:DRAW XX,Y:MOVE X,YY:DRAW XX
,YY
1830 REM*DRAW LEFT WALL/WING*
1840 MOVE X,Y:IF lwall<>0 DRAW X2,Y ELSE DRAW X2,Y2
1850 MOVE X,YY:IF lwall<>0 DRAW X2,YY ELSE DRAW X2,Y
3
1860 REM*DRAW RIGHT WALL/WING
1870 MOVE XX,Y:IF rwall<>0 DRAW X3,Y ELSE DRAW X3,Y2
1880 MOVE XX,YY:IF rwall<>0 DRAW X3,YY ELSE DRAW X3,
Y3
1890 ENDPROC
1900 REM*******ANOTHER GAME**
1910 DEF PROCagain
1920 CLS:COLOUR 3:PRINTTAB(0,13)"LIKE TO SEE MAP NOW
?"
1930 A$=GET$:IF A$<>"Y" AND A$<>"y" AND A$<>"N" AND
A$<>"n" THEN 1930
1940 IF A$="Y" OR A$="y" time%=0:PROCmap
1950 CLS:COLOUR 1:PRINTTAB(1,13)"WOULD YOU LIKE TO"T
AB(1,15)"TRY ANOTHER MAZE?":A$=GET$:IF A$<>"Y" A$<>"y
" AND A$<>"N" AND A$<>"n" THEN 1950
1960 IF A$="Y" OR A$="y" THEN RUN
1970 GOTO 2250
1980 ENDPROC
1990 REM********END TUNE*****
2000 DEF PROCtune
2010 FOR I=0 TO 1:SOUND1,-15,97,10:SOUND1,-15,105,10
:SOUND1,-15,89,10:SOUND1,-15,41,10:SOUND1,-15,69,20:N
EXT
2020 ENDPROC
2030 REM*******SHORT DELAY***
2040 DEF PROCdelay
2050 FOR I=1 TO 1000:NEXT
2060 ENDPROC
2070 REM*******LONG DELAY****
2080 DEF PROClongdelay
2090 FOR I=1 TO 20000:NEXT
2100 ENDPROC
2110 REM*******TIME-SCORE****
2120 DEF PROCtime
2130 IF time%=0 ENDPROC
2140 LOCAL TI,SC
2150 TI=TIME:TI=INT(TI/100)-(200+H%):TI=ABS(TI)
2160 SC=INT(200+(1.5*H%)-AD%*TI)
2170 COLOUR 1:IF TI<=0 COLOUR 3:PRINTTAB(6,15)"TIME
UP":SOUND0,-15,185,30:SOUND1,-15,185,30:PROClongdela
y:PROCagain
2180 PRINTTAB(0,31)"TI=";TI-1;"  ";TAB(11,31)"SC=";S
C;"  ";
2190 ENDPROC
2200 REM********TITLE********
2210 DEF PROCtitle
2220 PRINT:FOR I=0 TO 1:PRINTTAB(8)CHR$141CHR$157CHR
$132"B E E B M A Z E  "CHR$156:NEXT
2230 ENDPROC
2240 REM**MODE CHANGE/ENDING*
2250 *FX11,70
2260 MODE 7
2270 PROCtitle
2280 FOR I=0 TO 1:PRINTTAB(7,11+I)CHR$141CHR$157CHR$
136CHR$129"GOODBYE  FOR  NOW   "CHR$156:NEXT:END
```

WORDWISE Word Processor

BEEBUG Discount 13% SAVE £5

This is a highly sophisticated word processing package for the BBC Micro, and compares very favourably with those currently available on other microcomputers. It makes full use of the BBC micro's advanced facilities, and text is typed and edited in the 40 column Teletext mode, saving memory, thus allowing it to be used with more or less any TV. See the software review in this issue for further details.

Wordwise is supplied in EPROM with simple fitting instructions, a full manual, and a sample data cassette. Wordwise must be used in conjunction with a series 1 operating system.

The normal price of Wordwise is £39+VAT=£44.85(plus p&p)
To BEEBUG members it is £34+VAT=£39.10 plus 90p post & packing=£40.
Make cheques payable to BEEBUG and send to:
Wordwise Offer, PO Box 50, St Albans, Herts, AL1 2AR.

EXTRA-SPECIAL OFFER
WORDWISE PLUS 1.2 ROM

WORDWISE PACKAGE PLUS NEW 1.2 ROM IS OFFERED AT £45.00 INCLUDING P&P & VAT. AVAILABLE TO MEMBERS ONLY.

SEND CHEQUE TO WORDWISE OFFER, BEEBUG, PO BOX 50, ST ALBANS. IT IS ESSENTIAL TO QUOTE YOUR MEMBERSHIP NUMBER WITH ORDER, AND PLEASE ALLOW UP TO 28 DAYS FOR DELIVERY ON THIS DOUBLE OFFER.

BEEBUG NEW ROM OFFER

ACORN PRESS RELEASE TO BEEBUG MEMBERS

A special arrangement has been agreed between Acorn and BEEBUG whereby BEEBUG members may obtain the Series One Machine Operating System in ROM at the price of £5.95, including VAT and post and packing.

The ROM will be supplied with fitting instructions to enable members to install it in their machine.

In the unlikely event of subsequently operating incorrectly, members may take their machines to any Acorn dealer for the upgrade to be tested, which will be done at a charge of £5.00 plus VAT. This charge will be waived if the ROM is found to have been defective. If the computer has been damaged during the installation process, the dealer will make a repair charge.

NOTES ON ORDERING

1. To get a ROM, BEEBUG members should send a cheque for £5.95 to ROM Offer, BEEBUG, PO Box 109, High Wycombe, Bucks, HP11 2TD. It is essential to include a cheque with order, and to give your membership number.
2. ROM orders must not be combined with any other order (eg for software etc) and because of constraints on supply, multiple orders cannot be accepted until further notice.
3. Because of uncertainties in supply, please allow 4 weeks for delivery. We undertake not to cash cheques until the week prior to despatch, and we will provide a monthly account of the supply situation in BEEBUG. Please keep a note of the date on which you posted your order so that you can relate this to future announcements.
4. Please note that we cannot accept EPROM-based O.S. operating systems in lieu of payment. The exchange of EPROMs for the new operating system can only be performed by Acorn dealers or by Acorn's service centre at Fulbourn.

ADDRESS: ROM offer, BEEBUG, PO box 109, High Wycombe, Bucks, HP11 2TD.

NOTICE BOARD

SOFTWARE LIBRARY

Please note new address for members software library.
BEEBUG Software, PO BOX 109, High Wycombe, Bucks, HP11 2TD.

PAYMENT FOR CONTRIBUTIONS

New rates of pay for BEEBUG contributions covering hints, programs and articles can be found on page 5 of this issue.

MEMBERS' PERSONAL ADS.

We are starting a regular personal ad page in the magazine supplement. Rates are 15p per word for members' personal ads. Members' business ads cost 25p per word – and if you are selling any item in quantities rather than one off, then that counts as business. In either case send cheque with order, post to Personal Ads, BEEBUG, PO Box 50, St Albans, Herts, AL1 2AR. It is ESSENTIAL to include your membership number. Please note that we cannot arrange for box numbers.

IF YOU WRITE TO US

BACK ISSUES (Members only)

All back issues are kept in print from April 1982. Send 95p 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

Send all applications for membership, subscriptions renewals, and subscription queries to the subscriptions address.

Membership rates: £5.40 for 6 months (5 issues)
£9.90 for 1 year (10 issues)

European Membership £16 for 1 year

Elsewhere - Postal Zone A £19, Zone B £22, Zone C £23

Subscriptions Address
BEEBUG
PO BOX 109
High Wycombe
Bucks
HP11 2TD

SOFTWARE AND ROM OFFER (members only)

These are available from the address opposite, which is our NEW software address. Note that this does not relate to hardware - in this instance please see magazine for details).

Software Address
BEEBUG
PO BOX 109
High Wycombe
Bucks
HP11 2TD

TYPES OF ARTICLES AND PAYMENT

Useful articles are particularly welcome and we will pay around £30 per page for these, but in this case please bear in mind that you intend to write in the case of material longer than a page, we would prefer this to be presented on cassette or disc or machine readable form (using "Wordwise", "Wordtwat Editor" or other forms if you are unsure then please include a back-up copy as well).

Editorial Address
BEEBUG
PO Box 50
St Albans
Herts
AL1 2AR

We will also pay £10 for the best tip or the best re-published item each issue. Please send all editorial material to the editorial address opposite. If you require a reply or is essential to quote your membership number and address on it.

BEEBUG NEWSLETTER is edited and produced by Dr David Graham and Sheridan Williams.
Technical Editors: Colin Opie. Production Editors: Phyllida Vanstone.
Technical Assistants: Alan Webster.
Thanks are due to P.R.O. Pickering, John Yale, Adrian Calcraft, and Tim Powys-Lybbe for assistance with this issue.

All reasonable precautions are taken by BEEBUG to ensure that the advice and data given to readers are reliable. We cannot however guarantee it, and we cannot accept legal responsibility for it. Neither can we guarantee the products reviewed or advertised.

Printed & Published by Beebug Printers, St Albans & Unwin Brothers, Old Woking, Surrey.

© BEEBUG February 1983