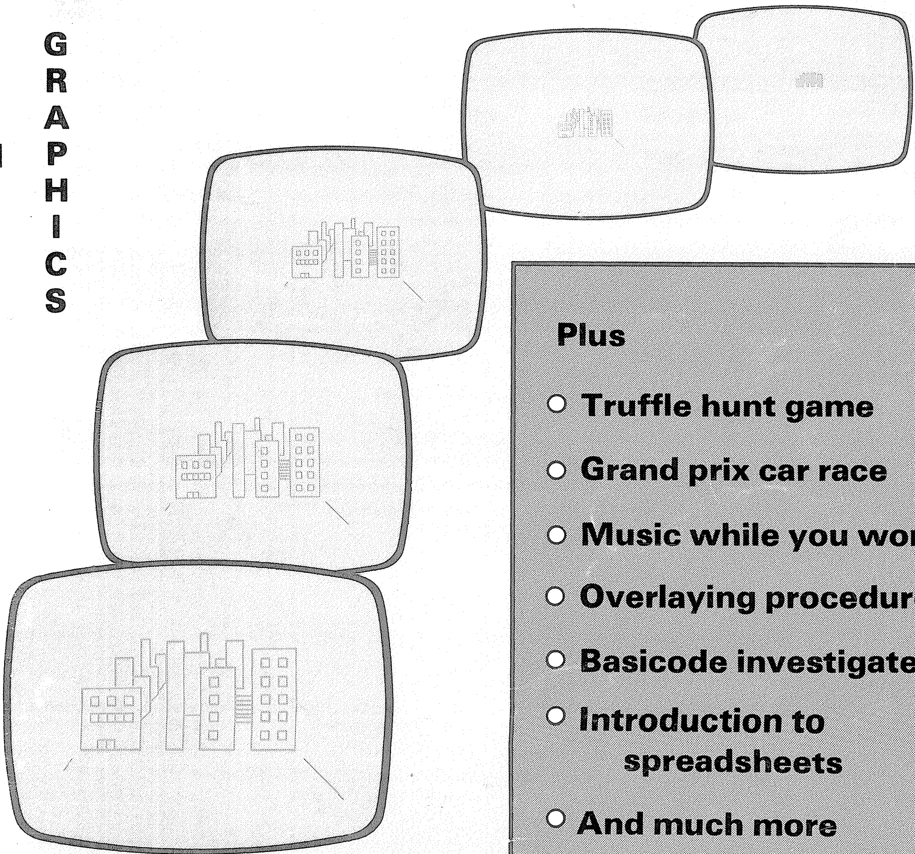
£1.00

BEEBUG FOR THE BBC MICRO

Vol 3 No 2 JUNE 1984

ZOOM GRAPHICS



Plus

- Truffle hunt game
- Grand prix car race
- Music while you work
- Overlaying procedures
- Basicode investigated
- Introduction to spreadsheets
- And much more

BRITAIN'S LARGEST COMPUTER USER GROUP
MEMBERSHIP EXCEEDS 25,000

EDITORIAL

SECOND PROCESSORS

Last month we were able to report on the launch of Acorn's 65Ø2 Second Processor, claimed by Acorn to be the most important event since the launch of the BBC micro itself. You may wonder, in looking at the contents of this issue, why we are not reviewing this major enhancement. The reason is quite simple: in common with most of the other computing magazines, our review system arrived nearly eight weeks after the launch date, not quite in keeping with Acorn's claim at the launch that Second Processors were in stock in the warehouse. Despite this, we are very excited by the 65Ø2 Second Processor and the Bitstik graphics system. Our review will appear next month, and we shall be telling you of the problems as well as the advantages of using the 65Ø2 Second Processor.

May also sees the launch of the Acorn Z8Ø Second Processor, clearly aimed at the business market with the extensive applications software that is bundled in as part of the package. We shall be reviewing this as well, as soon as we receive one!

THIS MONTH'S MAGAZINE

Amongst the many interesting articles and programs in this month's issue, I would particularly draw your attention to two. The first is another graphics program, allowing you to zoom in on any drawing you create at a tremendous magnification. We found this so fascinating that we have also created a competition (the Zoom Treasure Hunt) to go with this article, though, for technical reasons this is only feasible on the magazine cassette, and not printable in the magazine itself.

The other program is a major utility, primarily for disc users, that allows you to create Basic programs whose length is not at all limited by the physical memory available in the Beeb. This 'Virtual Memory' system will be a real boon to disc users, who have less memory to work with than cassette users. Mike Williams

TICE BOARD NOTICE BOARD NOTICE BOARD NOTICE BOA

BEEBUG'S ROM RULE

We have published a number of hints and tips in recent issues arising from clashes between the command sets used by different pieces of ROM software. BEEBUG has proposed a standard to resolve this problem. This allocates a unique letter (A to Z) to each software house, and this letter in turn can be used to optionally precede any command and give it a unique identity. BEEBUG has allocated 'B' for its own use, and in all future issues of ROM software, such as Toolkit, any command can be preceded by this letter to avoid confusion with any other ROM installed at the same time. So far the letter 'C' has been agreed with Computer Concepts, who have also adopted this standard, 'A' has been reserved for Acorn Computers, and 'W' for Watford Electronics.

HINT WINNERS

This month the hint winners are G.Middleton who wins the £1Ø prize and R.M.Blackall who wins the £5 prize. Keep sending those hints in please.

MAGAZINE CASSETTE

This month's magazine cassette contains, in addition to all the programs printed in the magazine, the data file for our Zoom Treasure Hunt competition and the winning entry by Bill Wilkinson in our 'Niagara Falls' Brainteaser Competition. The results of this, and a new Brainteaser, appear in this month's supplement.

MAGAZINE DISC

We have been unable to finalise the arrangements for producing the magazine 'cassette' on disc. We expect to conclude the details in time for inclusion in next month's issue.

BEEBUG MAGAZINE

GENERAL CONTENTS

Tested on Basic I & II and O.S. 1.2

ZOOM (32K)

by Pete and Derek Chown

For such a short program this one is remarkably entertaining. It allows you to zoom in and out of any picture you create with magnifications of up to 10 million! It is based on a perfectly serious application in Computer-Aided Design (CAD), but can be used to construct line drawings of high precision, or light-hearted sketches containing microscopic detail for a game of "Hide and Seek".

INTRODUCTION

You will remember from the BBC TV series, "The Computer Programme", the demonstration in which a house was shown with a globe in the window. The picture was magnified to show the British Isles on the globe. On the map was a house. In the window of the house was a globe... All that is possible on the BBC Microcomputer and with this little program. If you have a printer with graphics capability, you can add your own routine to print the results.

The 'ZOOM' program allows you to produce line drawings, and at any stage, enlarge or reduce the size of the picture. This allows details to be drawn much larger than their final size for accurate results, and also allows part of a picture to be hidden, by reducing it so small that it is no longer visible on the screen. The program is entirely in Basic, and should be quite straightforward to type in and save on cassette or disc.

USING THE ZOOM PROGRAM

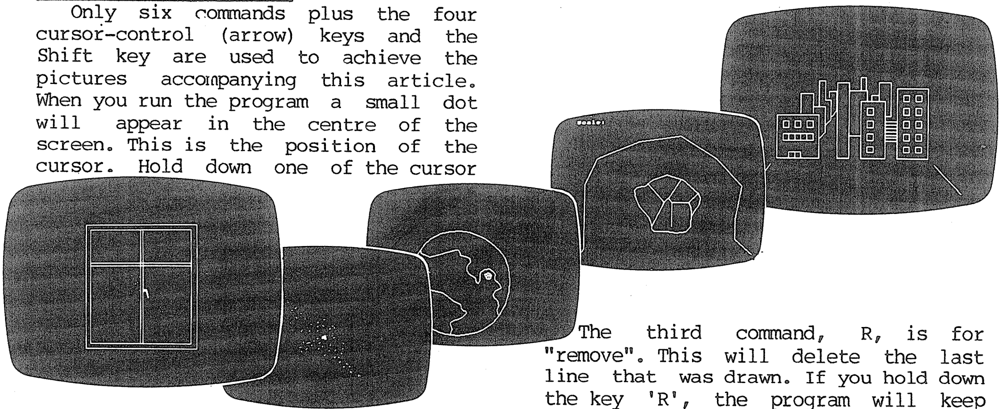
Only six commands plus the four cursor-control (arrow) keys and the Shift key are used to achieve the pictures accompanying this article. When you run the program a small dot will appear in the centre of the screen. This is the position of the cursor. Hold down one of the cursor keys and a line will be drawn (holding down Shift at the same time moves the cursor much faster). Use different cursor keys and the line can be moved as if connected to the starting point by a piece of elastic.

When the line is where you want it, press D for "draw" and the line becomes fixed. If you merely wish to move the starting point of the line without drawing it, press M for "move". So the procedure is: stretch the elastic to a new position, and then press either D or M. This is vastly superior to the "Sketch" method on the BBC "Welcome" cassette.

ZOOM Key Functions

M - Move	
D - Draw	f0 - Save screen
R - Remove	f1 - Load screen
S - Scale	
P - Print	

Cursor keys move cursor



The third command, R, is for "remove". This will delete the last line that was drawn. If you hold down the key 'R', the program will keep deleting lines, until all the lines have been removed. If removal of a line leaves a 'hole' in any remaining line, then the drawing can be 'repaired' by re-drawing with a scale of 1 (see later).

The fourth command is S for "scale". Type S and the word "Scale:" appears in the top left corner of the screen. Then type in a positive number for the magnification you require, e.g. 2 = twice as big, Ø.5 = half as big, 1 = same size. When Return is pressed the picture is re-drawn centred on the latest position of the cursor.

The command, P for "print", is only useful if you have a graphics dump routine and a printer. Press 'P' at any time to print the current screen display. The program listing shows where a printer dump routine (or command to one of the new printer dump ROMs like Printmaster) should be inserted.

We have also incorporated a simple facility to save and load screen displays, using the two function keys fØ for save and f1 for load. In each case the program will ask you for the filename to save or load. Be careful when saving and loading (especially on disc), as pressing f1 instead of fØ will delete the current picture in memory, and pressing fØ instead of f1 will wipe out a file on disc (but not cassette).

ZOOM IN PRACTICE

Some useful features of this program are the ability to create precise drawings, by building your picture up on a large scale, and then shrinking it down to size. You can also translate your shape in any of four directions. For example, if you have a square in the middle of the screen, then move the cursor to the top right-hand corner of this square. The press 'S' and choose a scale of 1 (same size). The square will now be re-drawn with the top right-hand corner in the middle of the screen.

LIMITATIONS

The program at present will allow drawings to be created with up to 6ØØ points, set in the program at line 11Ø. This could be increased and the ultimate limit is like to be determined by the amount of memory available for storing the lengthy arrays declared at line 3ØØ.

Using the scaling feature actually changes the stored data values. If you magnify any part of your drawing by more than about 1Ø million, you will be exceeding the accuracy of the micro and the results may be unrecogniseable. You may also find that because the stored data has been changed, you cannot reduce the picture correctly back to the original version. A screen will always be saved at the current magnification.

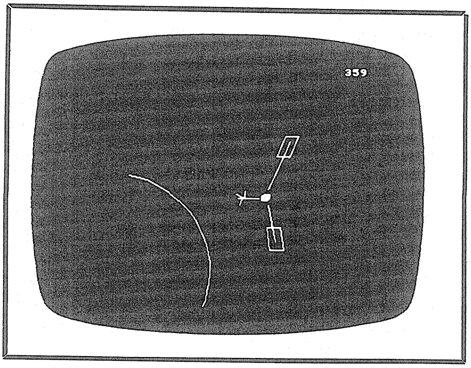
ZOOM FOR FUN

The program really is great fun to use, and you can puzzle your friends by hiding some minute detail somewhere in a picture and challenging them to find what it is. If you try this, we have found that it is better to start at the highest level and gradually magnify parts of the picture as you get more and more detailed. If you start with the smallest details and gradually reduce these to hide them, you are like to find a very obvious 'blob' being drawn as part of the initial picture, a real give away to the location of the hidden detail.

The pictures with this article show one such sequence. The first shows a window with a view of the stars in the sky (actually the constellation of Orion). If you zoom in on one of the stars in Orion you will see a planet surrouded by asteroids. Magnifying the central planet will reveal the outline of some continents and an island, and further magnification of the island will show first the road system on the island, and finally a city of tall buildings. One point you soon realise is that you only have to be as little as a millemetre out initially to be a 1ØØ kilometres adrift at a magnification of 1Ø million.

ZOOM TREASURE HUNT

Because ZOOM is such an interesting program we have provided a data file, DZOOM, on the magazine cassette for you to explore. Unfortunately the data file is much too large and in the wrong format to be included in the magazine itself. If you search hard enough you will find the name 'BEEBUG' hidden



somewhere in the picture you see here. Tell us where 'BEEBUG' is located and we will award a prize of a Computer Concepts Graphics ROM to the first two correct answers opened. Send all entries to the editorial address, and mark the envelope 'ZOOM'. Good zooming!

```
   10 REM PROGRAM ZOOM
   20 REM AUTHORS P.CHOWN A.WEBSTER
   30 REM VERSION BØ.6
   40 REM BEEBUG  JUNE 1984
   50 REM PROGRAM SUBJECT TO COPYRIGHT
   60 :
  100 ON ERROR GOTO 390
  110 np%=600
  120 *FX4,2
  130 *K.Ø z
  140 *K.1 y
  150 *K.10|UO.|M
  160 *K.12|H
  170 *K.13|I
  180 *K.14|J
  190 *K.15|K
  200 *FX12,3
  210 :
  220 MODE 4
  230 VDU19,1,2,Ø,Ø,Ø
  240 X=639.5:Y=511.5
  250 oldX=X:oldY=Y
  260 pnt%=Ø
  270 DIM endx(np%),endy(np%),startx(np
%),starty(np%)
  280 GCOL 4,Ø
  290 PROCline
  300 REPEAT
  310 VDU4:PRINTTAB(3Ø,Ø);pnt%;SPC(5):V
DU5
  320 COM$=GET$:*FX15,Ø
  330 IF COM$="P" PROCdump
  340 PROCline
  350 PROCcom
  360 PROCline
  370 UNTIL Ø
  380 :
  390 ON ERROR OFF
  400 *FX4
  410 *FX12
  420 MODE 7
  430 IF ERR=17 END
  440 REPORT:PRINT" at line ";ERL
  450 END
  460 :
 1000 DEF PROCline
 1010 MOVE oldX,oldY
 1020 *FX19
 1030 PLOT21,X,Y
 1040 ENDPROC
 1050 :
 1060 DEF PROCcom
 1070 IF INKEY-1 C=32 ELSE C=4
 1080 IF COM$=CHR$(8) OR COM$=CHR$(14Ø)
X=X-C
 1090 IF COM$=CHR$(9) OR COM$=CHR$(141)
X=X+C
 1100 IF COM$=CHR$(1Ø) OR COM$=CHR$(142
) Y=Y-C
 1110 IF COM$=CHR$(11) OR COM$=CHR$(143
) Y=Y+C
 1120 IF COM$="M" oldX=X:oldY=Y
 1130 IF COM$="D" GCOL Ø,1:MOVE X,Y:DRA
W oldX,oldY:GCOL 4,Ø:startx(pnt%)=oldX:
starty(pnt%)=oldY:endx(pnt%)=X:endy(pnt
%)=Y:pnt%=pnt%+1:oldX=X:oldY=Y
 1140 IF COM$="R" AND pnt%<>Ø pnt%=pnt%
-1:GCOLØ,Ø:MOVE startx(pnt%),starty(pnt
%):DRAW endx(pnt%),endy(pnt%):GCOL4,Ø
 1150 IF COM$="S" AND pnt%>Ø PROCscale(1)
 1160 IF COM$="z" OR COM$="y" PROCsavel
oad
 1170 ENDPROC
 1180 :
 1190 DEF PROCscale(sc)
 1200 IF sc=Ø GOTO 124Ø ELSE MOVE Ø,1Ø23
```

```
 1150 IF COM$="S" AND pnt%>Ø PROCscale(1)
 1160 IF COM$="z" OR COM$="y" PROCsavel
oad
 1170 ENDPROC
 1180 :
 1190 DEF PROCscale(sc)
 1200 IF sc=Ø GOTO 124Ø ELSE MOVE Ø,1Ø23
```

```
 1210 VDU5:GCOL 4,Ø:INPUT "Scale:"SCALE$
 1220 SCALE=VAL(SCALE$)
 1230 IF SCALE<=Ø VDU 7:ENDPROC
 1240 CLS:GCOL Ø,1
 1250 FOR I%=Ø TO pnt%-1
 1260 startx(I%)=(startx(I%)-X)*SCALE+6
39.5:starty(I%)=(starty(I%)-Y)*SCALE+51
1.5
```

```
 1270 endx(I%)=(endx(I%)-X)*SCALE+639.5
:endy(I%)=(endy(I%)-Y)*SCALE+511.5
 1280 IF ABS(startx(I%))<32768 AND ABS(
starty(I%))<32768 AND ABS(endx(I%))<327
68 AND ABS(endy(I%))<32768 MOVE startx(
I%),starty(I%):DRAW endx(I%),endy(I%)
 1290 NEXT
 1300 X=639.5:Y=511.5:oldX=X:oldY=Y
 1310 GCOL 4,0:VDU 7
 1320 ENDPROC
 1330 :
 1340 DEFPROCdump
 1350 REM VDU5:*GDUMP1 1
 1360 ENDPROC
 1370 :
 1380 DEFPROCsaveload
 1390 VDU4:VDU31,0,0
 1400 IF COM$="y" PRINT " Load "; ELSE
PRINT " Save ";
 1410 INPUT"Filename :"fn$:CLS:VDU5
 1420 IF COM$="y" GOTO 1500
 1430 X=OPENOUT fn$
 1440 PRINT #X,pnt%
 1450 FORp=0TOpnt%
 1460 PRINT #X,startx(p),starty(p),endx
(p),endy(p)
 1470 NEXT
 1480 CLOSE#0
 1490 GOTO 1570
 1500 :
 1510 X=OPENIN fn$
 1520 INPUT #X,pnt%
 1530 FORp=0TOpnt%
 1540 INPUT #X,startx(p),starty(p),endx
(p),endy(p)
 1550 NEXT
 1560 CLOSE#0
 1570 SCALE=1:X=640:Y=512:PROCscale(0):
*FX15
 1580 ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

WHICH DISC DRIVE ? - R.M.Blackall

Location &10CB, contains the number of the current disc drive (Acorn DFS 0.90).

MORE PERMANENT *TV EFFECT - G.Middleton

For those who would like to maintain a *TV type shift for the duration of a session on their Beeb; try the following short routine.

```
   10 MODE7
   20 *FX247,0
   30 *FX248,0
   40 *FX249,0
   50 INPUT"Screen shift",X%
   60 INPUT"Interlace (ON/OFF)",Y$
   70 IF Y$="ON" THEN Y%=0 ELSE Y%=1
   80 P%=&A01:[OPT1:.SS LDA#&90:LDX#X%:LDY#Y%:JSR&FFF4:RTS
   90 .start LDA#&F7:LDX#&4C:JSR&FFF4:LDA#&F8:LDX#SS MOD 256:JSR&FFF4:LDA#&F9:LDX#S
   S DIV 256:JSR &FFF4:RTS]
   100 CALL start
```

ASCII VALUE AT THE CURSOR - T. Powys-Lybbe

Is this the shortest way to find the ASCII value of the character at the text cursor, and the current screen mode ? (A%,M% return ASCII value, current mode respectively.)

```
   DIMQ%3:A%=135:!Q%=USR&FFF4:A%=Q%?1:M%=Q%?2
```

THE SHORTEST GAME ? - ROGER JANE

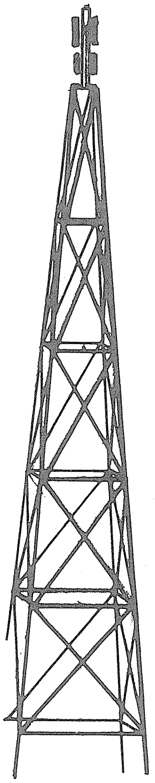
(See the previous shortest Vol.2, No.6, p.19 - after the REM statements have been removed).

```
   0H%=0:REPEATMODE7:VDU28;24,39;:P%=&7C64:S%=0:REPEAT?P%=86:PRINT"|"TAB(RND(38))"*
   "TAB(39)"|"CHR$13;:P%=(P%+INKEY(-98)-INKEY(-67)-19)MOD38+&7C51:S%=S%+1:UNTIL?P%=
   42:H%=H%+(S%>H%)*(H%-S%):VDU7:PRINT''"Score:"S%'"Hi:   "H%'"Press
   space: ";:REPEATUNTILGET=32:UNTIL0
```

Unfortunately there are no instructions with this, but the Z and X keys control left and right directions of movement. Imagination dictates that you have to pilot a spaceship safely through a meteor storm! Now try it for yourself, or perhaps you have a better one !

A LOOK BACK AT BASICODE

by Steve Fox



BBC Radio 4 has recently completed a series of broadcasts of computer programs using an Esperanto form of Basic known as Basicode. Now that this 'Take Away' service, as it was called, has come to an end, at least for the moment, Steve Fox takes a look back, and assesses its value.

Basicode is the brain-child of a handful of Dutch computer enthusiasts. The idea is to package a Basic program in such a way that it will run on a variety of different machines, and to build up a stock of non-copyrighted software, written by amateurs for amateurs. The programs are to be made available to the widest possible group of users via the domestic radio services of the countries participating. Radio Hilversum broadcasts them at 1840 GMT each Sunday on 747 kHz and West Germany also participates. The BBC, perhaps fearing adverse listener reaction to the unmusical tones, has so far confined transmissions to the unsocial hour of 0023 hrs GMT. The first series of broadcasts, associated with the Radio 4 "Chip Shop" programme, is now at an end but doubtless a further series will follow.

The Basicode system has to overcome two problems in order to make a program readable by a range of machines. Firstly, the tones transmitted (or recorded on cassette) must be comprehensible to the operating system, and secondly the Basic statements must be capable of interpretation by the computer's Basic ROM. To make these things possible, different computers must be provided with a different software interface. The BBC supplies, for a modest £3.95, all the software necessary to load a Basicode program into a range of popular machines - Apple, Sinclair, Commodore, Tandy, etc, and of course the BBC micro. Also included is a corresponding range of "Basicode Save" programs to enable the user to produce a Basicode version of his own software. Some micros also need additional hardware to be built, but this is fortunately not necessary for the Beeb.

The Basicode signal is an ASCII listing of the program made up of bits which are serially transmitted at 1200 Baud (bits per second). The obvious method of coding would be to send one cycle of 1200 Hz tone for a '1' and 1/1200th of a second's silence for a zero. However, this would mean that system noise might generate a '1' for a zero, so Basicode defines a '1' as two cycles of 2400 Hz tone and a zero as one cycle of 1200 Hz tone. To make the system even more robust, each byte is preceded by a startbit (logic 0), and is followed by two stopbits (logic 1). At the end of each program, a special 8-bit word called a checksum is included. By carrying out an EOR calculation on the preceding bytes it is possible to check that they have been successfully read. If not, a checksum error is declared by the loading program, and it is up to the user to scrutinize the listing to try and find where the program has been mis-read. Notice that at this stage the computer has not recognized the data as a Basic program, so until "RUN" is entered no error messages can be issued.

Because Basicode statements have to be acceptable by the full range of machines, each with its own Basic dialect, it follows that the programs have to be written using only those statements which are common to all. This is a very severe limitation indeed, especially for BBC Micro users. Gone are procedures, graphics statements, sound statements, assembler language, and *FX commands, to name but a few! With Basicode, we enter the colourless world of basic Basic. Is it worth it? Well, read on...

The original Basicode was so crude that, as there exists no universal statement for "Clear the screen", it couldn't be cleared! But then it was realized that what was needed was a mechanism for interpreting a number of universal commands into the "local" lingo. Basicode 2 issues these commands in the form "GOSUB (line no.)" and the Loading program appends to the front of the main program, lines which will interpret such commands. The main program might, for example, begin "1ØØØ GOSUB 1ØØ", and if yours is a BBC machine the Loader will supply the subroutine at line 1ØØ in the form: "CLS:RETURN". Altogether there are a dozen such subroutines for a carrying out such highly desirable tasks as positioning the cursor, inputting from the keyboard, giving random numbers, beeping and print-formatting. To jump over these sub-routines the first line has to be "1Ø GOTO 1ØØØ". In addition to these customized sub-routines, the Loaders may carry out some special modifications to the listing. BBC Basic is unusual in that it doesn't require a space to be left after the line number and in the use of LN rather than LOG for logs to the base 'e'.

These changes are taken care of by the Basicode Load program, which consists of an instruction screen at Page &EØØ (or &19ØØ for disk users), the subroutine lines which are initially stored at Location &77FE, and a machine code program starting at location &7979, and running up to the Mode 7 screen at &7CØØ. The normal load consists of shifting down to PAGE the sub-routines (over-writing the instructions) and then appending the decoded listing as it is read off tape. Once in memory, the program can be RUN and SAVEd like any other straightforward Basic program.

Writers of Basicode Basic have to observe a few more restrictions. Lower-case letters are not allowed, nor are integer variables. Real variables are single-precision only. A screen size of 24 by 4Ø is standard, but larger sizes (including Modes 6 and 7) are OK. The names of variables must not be more than two characters long and must not begin with "O".

With such depressing restrictions, I expected little of the sample programs on the "Chip Shop" cassette, kindly supplied by the BBC, and indeed the presentation of them all does leave very much to be desired. Four or even three years ago, this tape would have been thought a goldmine, but in 1984 such packages, as they stand, cannot be acceptable to the BBC micro user. However, if one looks below the surface at the program ideas themselves, several are indeed good, and well worth the effort of rewriting using the full range of facilities on the Beeb.

Items which particularly appeal to me are: "Chords", guitar fingering to order, "Yahtzee" (similar to 'Five Dice'; BEEBUG Vol.1 No.9), a game of chance (of which the rules are fairly easy to guess if you don't know them but which aren't properly explained), and Mezirac's Squares. The Germanic humour in "Titration" is a bit heavy, but it makes one of the duller processes of chemistry seem like fun, which is surely the measure of any good educational program.

I also made one or two off-air recordings and, there were no technical problems, but the programs I "took away" are in fact the same as those on the cassette.

The ultimate value of the Basicode experiment lies in the quality (and quantity) of the software which people are prepared to donate or sponsor. Some of the present software carries words of encouragement aimed at getting the reader to "do his bit", but without a few carrots being offered by the BBC, I'm not too optimistic about the future of the project. I feel too that BBC micro users would be better served if the free software which appears on CEEFAX were to be broadcast as tones for direct recording on cassette, instead of being available only to those who are willing to invest in a Teletext Adaptor. The BBC's Telesoftware may not be too wonderful, but most of it is written for the BBC micro, and so doesn't suffer from the over-simplification demanded by Basicode.
The Chip Shop cassette and Basicode manual are available at £3.95 including postage and packing from Broadcasting Support Services, PO Box 7, London W3 6XJ. This must be the bargain of the year!

Tested on Basic I & II and O.S. 1.2

MUSIC WHILE YOU WORK (16K)

by David A. Fell

Some people find that they can concentrate much better if there is some background noise, preferably music, continually playing whilst they are working. In this article we show you how your Beeb can be made to play the music of your choice while you continue using the machine for programming, word processing or whatever else takes your fancy (apart from commercial machine code games).

Sometimes, even using your Beeb can become a little boring, such as waiting for a cassette file to load, or listing a long program out to a printer. The programs described here can liven up these moments (and others) by playing music at the same time using interrupts. These are used continually by the operaing system, but in the packaged form of 'events' (see BEEBUG Vol.2 No.8) they can also perform operations for the user. The programs listed here allow for you to generate a data file (containing a tune), and then play this tune using a special 'event driven' routine.

The main program 'MUSIC' assembles and saves a machine code routine that plays the tune. The program is well anotated, and has been kept as short as possible. To use it, you will also need a data file holding a sequence of bytes that correspond to the tune that you wish to play. The program 'CREATE' allows you to generate such a data file.

The first step is to type the two programs in, and to save them before running (this is especially so for the MUSIC program, as any mistake in entering this could result in your program being corrupted). Once this has been done, you should run the MUSIC program. This will pause briefly while it assembles, and then *SAVEs the resulting machine code program under the name TUNE.

The next stage is to run the CREATE program. This short Basic program is used to enter the tune to be played and to create a corresponding data file. When using CREATE, you will be prompted to enter a group of four numbers, the same as you would enter in a normal Basic SOUND statement. For example;

```
SOUND 17,-1Ø,2ØØ,3
```

would be entered simply as 17,-1Ø,2ØØ,3. To create a tune, just keep entering the notes until you reach the end. Once you have finished entering the data, just type Ø,Ø,Ø,Ø. This will cause the data to be saved as a file called DATA. Incidentally, you cannot have more than 255 notes, nor less than two. (The data supplied will play 'Jesu Joy of Man's Desiring' from Bach's Cantata No.147, as we listed in BEEBUG Vol.2 No.9.)

If you have a word processor such as Wordwise, you might find it easier to build up a text file that contains your data. The file should start with

```
OLD
RUN
```

followed by the data for the tune to be played. To use this proceed as follows. Type:

```
*BASIC
LOAD"CREATE"
*EXEC FILE
```

where FILE is the name of the text file containing the data. This causes the CREATE program to be run, and then the numbers following in the data file would be input as the data. You can now watch the computer enter the data for you. The advantage of this method is that if you decide that one note is wrong, then only that particular note need be altered in the text file.

Whenever you want a musical accompaniment, just type:

```
*RUN TUNE
```

This will run the machine code program TUNE. Note that it expects to *LOAD the file called DATA. If you wish, you could have different versions of the TUNE program that *LOAD different files, and thus play different tunes.

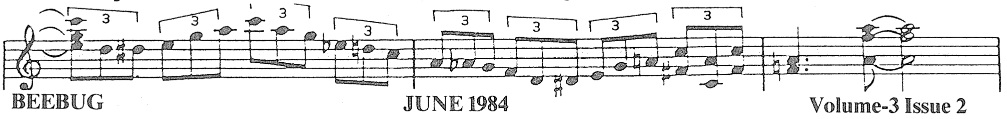
TAPE USERS
If you are using tape, then you will need to alter the locations at which the data and machine code program are to reside. The program TUNE could easiy reside at &DØØ onwards by altering line 12Ø of the MUSIC program to P%=&9ØØ. As the data can no longer reside at &AØØ, we also need to alter lines 1ØØ and 141Ø to refer to &BØ1 and &BØØ and not as listed.

```
 1,1,81,4    1,1,177,4    1,1,137,4    1,1,145,4    1,1,177,4    1,1,1Ø9,4
2,1,81,12    2,1,117,12   2,1,1Ø1,12   2,1,81,12    2,1,117,12   2,1,1Ø9,12
1,1,129,4    1,1,157,4    1,1,165,4    1,1,129,4    1,1,157,4    1,1,129,4
1,1,137,4    1,1,145,4    1,1,157,4    1,1,137,4    1,1,145,4    1,1,125,4
1,1,145,4    1,1,129,4    1,1,125,4    1,1,145,4    1,1,129,4    1,1,129,4
2,1,129,12   2,1,69,12    2,1,89,12    2,1,117,12   2,1,1Ø9,12   2,1,81,24
1,1,157,4    1,1,137,4    1,1,129,4    1,1,157,4    1,1,137,4    1,1,145,4
1,1,149,4    1,1,145,4    1,1,137,4    1,1,149,4    1,1,145,4    1,1,157,4
1,1,149,4    1,1,149,4    1,1,1Ø9,4    1,1,149,4    1,1,117,4    1,1,177,4
2,1,117,12   2,1,89,12    2,1,77,12    2,1,1Ø1,12   2,1,1Ø1,12   1,1,157,4
1,1,165,4    1,1,157,4    1,1,125,4    1,1,165,4    1,1,157,4    1,1,145,4
1,1,157,4    1,1,165,4    1,1,137,4    1,1,157,4    1,1,149,4    1,1,129,24
1,1,157,4    1,1,157,4    1,1,149,4    1,1,157,4    1,1,145,4    2,1,1Ø9,24
2,1,97,12    2,1,97,12    2,1,1Ø9,12   2,1,97,12    2,1,1Ø5,12   Ø,1,Ø,Ø
1,1,177,4    1,1,149,4    1,1,145,4    1,1,177,4    1,1,137,4    Ø,Ø,Ø,Ø
1,1,173,4    1,1,145,4    1,1,137,4    1,1,173,4    1,1,129,4
```
Data for 'Jesu Joy of Man's Desiring'

PROGRAM NOTES
The program initialises itself by copying the current event vector to a safe place for future reference, and then copies the entry address for itself into the event vector. This means that whenever an event occurs, our piece of code will be called first. Then, under the vertical sync event (which occurs regularly every fiftieth of a second, and so provides a frequent event for us) the program plays its notes. Once entered, the program repeatedly enters notes into the sound buffer until it finds that the buffer for sound queue number 1 is full. Once this has happened, the routine exits to the original event routine. The reason for doing this test is to ensure that the routine doesn't end up waiting for a note to finish playing.

The data for the tune is held in a data file which is *LOADed into memory when the machine code program is run. In this file, the first byte holds the total number of notes in the tune, and then groups of eight bytes, in the format as described on page 461 of the User Guide (i.e. channel, volume, pitch, duration as two byte low-high numbers). By storing the notes in this format, there is no need for extensive decoding of the data before the call to OSWORD is made. In fact, all that is done is that the current note number is multiplied by 8, and an offset (equal to the base of the data - &AØ1 in the example given) is added. This gives us the values to pass directly to OSWORD. It would be possible to take the sound statements and to compress them into four bytes per note, but this would require a considerable degree of decoding, and would drastically increase the length of the machine code.

Although the program itself can cope with up to 255 notes, there are some practical limitations. This is because the area of memory 'safe' from most applications is sorely limited. Thus, it is not very easy to have more than 64 notes with the current program design.



```
  10 REM PROGRAM MUSIC
  20 REM AUTHORS DAVID FELL
  30 REM AND ALAN WEBSTER
  40 REM VERSION B1.0
  50 REM BEEBUG JUNE 1984
  60 REM PROGRAM SUBJECT TO COPYRIGHT
  70 :
 100 data=&A01    :REM ALTER FOR DATA.
 110 FOR PASS=0 TO 1
 120 P%=&900      :REM ALTER FOR CODE.
 130 [OPT PASS*2
 140 \ FIRST DEFINE THE ENVELOPE
 150 \ TO BE USED
 160 LDA #8
 170 LDX #ENV AND 255
 180 LDY #ENV DIV 256
 190 JSR &FFF1
 200 \ THEN *LOAD IN THE DATA BLOCK
 210 LDX #(ENV+14) AND 255
 220 LDY #(ENV+14) DIV 256
 230 JSR &FFF7
 240 \ STOP ANY INTERRUPTS OCCURING
 250 \ WHILE WE SWITCH OVER THE EVENT
 260 \ VECTOR.
 270 SEI
 280 LDA &220
 290 STA &230
 300 LDA &221
 310 STA &231
 320 \ OLD VECTOR COPIED
 330 \ *** CAUTION ***
 340 \ DON'T RUN MORE THAN ONCE OR
 350 \ WE'LL FOUND OURSELVES LOOPING
 360 \ AROUND, AND NEVER GETTING
 370 \ ANY WORK DONE!
 380 LDA #EVENT AND 255
 390 STA &220
 400 LDA #EVENT DIV 256
 410 STA &221
 420 \ NEW VECTOR INSTALLED
 430 LDA #14
 440 LDX #4
 450 JSR &FFF4
 460 \ WE'LL USE THE VERTICAL SYNC
 470 \ EVENT 'CAUSE IT'S THE EASIEST
 480 \ MAKE SYNC EVENT HAPPEN
 490 LDA #0
 500 STA &90
 510 CLI
 520 \ ALLOW INTERRUPTS AGAIN,
 530 \ AND EXIT
 540 RTS
 550 \ THIS IS THE MAIN ENTRY POINT
 560 .EVENT
 570 CMP #4
 580 \ IS THE EVENT OURS?
 590 BEQ EVENT1    \ YES
 600 JMP (&230)    \ NO!
 610 .EVENT1
 620 \ WE'VE GOT THE EVENT
 630 PHA
 640 TXA
 650 PHA
 660 TYA
 670 PHA
 680 \ SAVE ANY REGISTERS, MAINLY
 690 \ FOR THE SAKE OF SAFETY
 700 .EVENT3
 710 LDA #128
 720 LDX #250
 730 LDY #255
 740 JSR &FFF4
 750 \ CHECK TO SEE IF CHANNEL ONE
 760 \ IS FULL. THIS ALLOWS FOR US
 770 \ TO KEEP PILING UP THE NOTES
 780 \ WITHOUT HAVING TO WAIT WITH
 790 \ A FULL BUFFER.  CHANNEL ONE
 800 \ SHOULD BE  USED FOR MOST OF
 810 \ THE PROCESSING THOUGH.
 820 CPX #0         \ FULL?
 830 BNE EVENT2     \ NO
 840 .EXIT          \ YES
 850 \ GENERAL EXIT POINT
 860 PLA
 870 TAY
 880 PLA
 890 TAX
 900 PLA
 910 JMP (&230)
 920 \ EXIT VIA OLD EVENT HANDLER
 930 \ ALLOWS OTHERS TO BE CHAINED IN
 940 .EVENT2
 950 INC &90
 960 \ UPDATE BLOCK POINTER
 970 LDA &90
 980 \ CHECK IF GOT TO END OF LIST
 990 CMP data-1
1000 BNE L1      \ NO
1010 LDA #0      \ YES
1020 STA &90
1030 .L1
1040 PHA
1050 LDA #0
1060 \ ZERO OUT WORKSPACE
1070 STA &92
1080 STA &93
1090 PLA
1100 STA &92
1110 LDX #2
1120 .L2
1130 ASL &92
1140 ROL &93
1150 DEX
1160 BPL L2
1170 \ MULTIPLY BLOCK # BY 8
1180 CLC
1190 LDA #data AND 255
1200 ADC &92
1210 TAX
1220 LDA #data DIV 256
1230 ADC &93
1240 TAY
```

»

```
1250 \ ADD IN OFFSET, AND COPY TO
1260 \ THE X AND Y REGISTERS, READY
1270 \ FOR THE OSWORD CALL TO MAKE
1280 \ THE SOUND
1290 LDA #7
1300 \ OSWORD # 7 IS MAKE SOUND
1310 JSR &FFF1
1320 JMP EVENT3
1330 \ KEEP LOOPING 'TIL BUFFER FULL.
1340 .ENV
1350  \ DATA TO DEFINE THE ENVELOPE
1360 ]
1370 RESTORE:FOR I%=0 TO 13
1380 READ ENV?I%
1390 NEXT
1400 DATA 1,1,-1,0,1,1,-1,127,-2,-1,-1
5,127,127,0
1410 $(ENV+14)="LOAD DATA A00"
1420 NEXT
1430 *SAVE TUNE 900 9FF
```

```
  10 REM PROGRAM CREATE (MUSIC)
  20 REM AUTHOR DAVID FELL
  30 REM VERSION B1.0
  40 REM BEEBUG JUNE 1984
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 MODE 7
 110 DIM N%(255,3)
 120 PROCbanner("Music")
 130 PROCbanner("Data file Creator")
 140 PROCbanner("By David Fell")
 150 PRINT'"Enter the desired four par
amters for thesound command like this:"
'"A,B,C,D"'"Use 0,0,0,0 to exit."
 160 VDU28,0,24,39,10
 170 N%=0
 180 REPEAT
 190 N%=N%+1
 200 CLS
 210 PRINT"ENTER NOTE NUMBER ";N%
 220 INPUT""N%(N%,0),N%(N%,1),N%(N%,2)
,N%(N%,3)
 230 IF (N%(N%,0) OR N%(N%,1) OR N%(N%
,2) OR N%(N%,3))=0 AND N%=1 N%=0
 240 UNTIL (N%(N%,0) OR N%(N%,1) OR N%
(N%,2) OR N%(N%,3)) =0 AND N%<>1 OR N%=
255
 250 CLS
 260 PRINT"NUMBER OF NOTES ENTERED = "
;N%
 270 F%=OPENOUT"DATA"
 280 IF F%=0 PRINT"CAN'T OPEN FILE":END
 290 BPUT#F%,N%
 300 FORI%=1TON%
 310 FORJ%=0TO3
 320 PROCbput(N%(I%,J%))
 330 NEXT
 340 NEXT
 350 CLOSE#F%
 360 PRINT"DATA FILE CREATED"
 370 END
 380 :
1000 DEF PROCbanner(A$)
1010 LOCAL I%
1020 FORI%=0TO1
1030 PRINTTAB(16-(LENA$DIV2));
1040 VDU141,129,157,131
1050 PRINTA$;
1060 VDU32,32,156
1070 NEXT
1080 ENDPROC
1090 :
1100 DEF PROCbput(N%)
1110 BPUT#F%,N% AND 255
1120 BPUT#F%,N% DIV 256
1130 ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

INTEGERS AND INTEGERS - Richard Sterry

Note that conversion of real numbers to integers can be done in two alternative but direct ways that give different answers (for negative numbers only):

```
    A%=INT(-1.3)  gives A%=-2
```

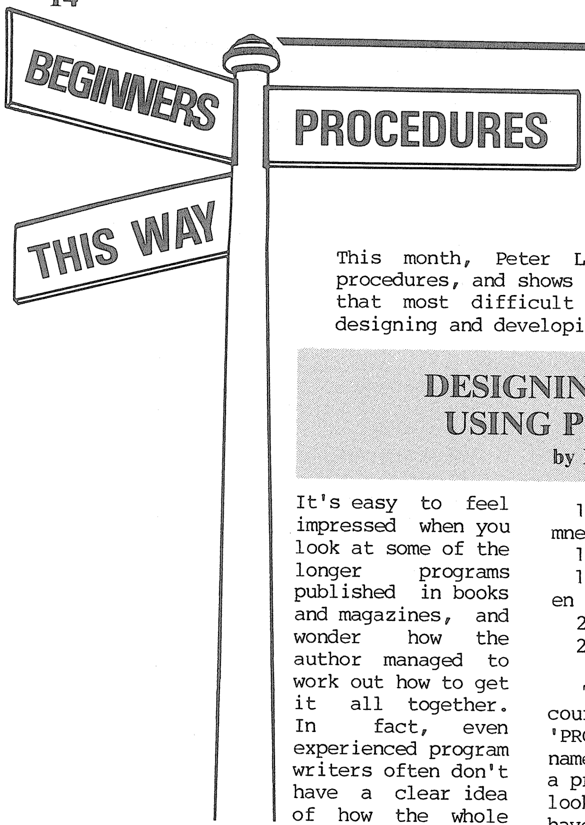
by rounding down to the next nearest integer but

```
    A%=(-1.3)     gives A%=-1
```

by truncating the number for the integer part.

FIFTH WAY TO USE FUNCTION KEYS - Paul Holgate

In addition to the four ways of obtaining input from any of the function keys listed in BEEBUG Vol.2, No.6, p.41 there is a fifth way (User Guide, p.275).Using INKEY with a negative argument (eg. -33 for f0), then depression of a particular function key can be detected. However, don't forget that if used in this way, a function key will enter it's soft key definition into the keyboard buffer unless disabled by the *FX18 command.



BEGINNERS START HERE

This month, Peter Lewis continues writing about procedures, and shows you how useful they are for that most difficult of tasks for the beginner, designing and developing your own program.

DESIGNING PROGRAMS USING PROCEDURES

by Peter Lewis

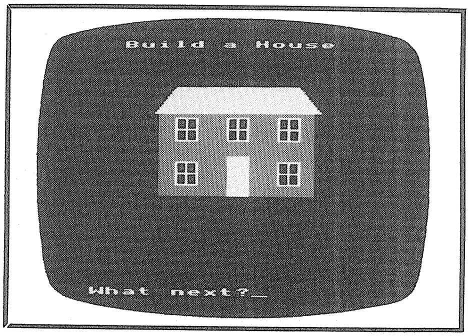
It's easy to feel impressed when you look at some of the longer programs published in books and magazines, and wonder how the author managed to work out how to get it all together. In fact, even experienced program writers often don't have a clear idea of how the whole program will finish up when they start writing. An excellent approach to this sometimes daunting task involves the use of procedures right from the word go, and this is what I'm going to tell you about this month.

Let's take a fairly simple example of a program that we want to write. We will construct something that will build up a picture of a house on the screen as we type in the names of the various parts of the house. We will use Mode 2, because that allows us to use lots of colours. Our basic program could look something like this:

```
  100 MODE 2
  110 PROCbegin
  120 REPEAT
  130 PROCask
  140 IF answer$="walls" THEN PROCwalls
  150 IF answer$="roof" THEN PROCroof
  160 IF answer$="windows" THEN PROCwin
dows
  170 IF answer$="chimney" THEN PROCchi
mney
  180 IF answer$="door"THEN PROCdoor
  190 IF answer$="garden" THEN PROCgard
en
  200 UNTIL answer$="finish"
  210 END
```

This isn't the complete program, of course. Every occurrence of the word 'PROC' in the program followed by a name like PROCgarden, is an example of a procedure call, just like those we looked at last month. At the moment, we have not described (or defined) any of the procedures that we are going to use.

Now this may sound odd at first, but in fact is one of the real beauties of using procedures anyway. At this stage in designing our program, we don't really know what our house is going to


```
Build a House



What next?_
```

look like. We don't know how the walls or roof will be drawn. But that doesn't matter, because by using procedures, we can put together the outline of our whole program as you see it above.

Many of the procedures will be obvious from their names, and just draw the relevant part of our house. The procedure called PROCask is where the program will ask for the name of the next part of the house to be drawn, and the program assumes that the answer is in fact stored using the character (or string) variable 'answer$'. The procedure called PROCbegin simply indicates that there may well be a need to set up a few things at the start of the program. All of this is very nice even if still rather vague at this stage. Of course we can think about how the program looks, and change it around if we want to before putting in any more detail. Nearly any program, large or small, that we want to write, can be started in this outline fashion.

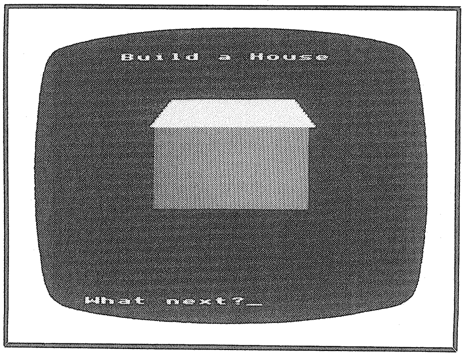
We will now start defining some of our procedures, adding more detail to our program. It is a good idea to leave a gap in the line numbers between the main program already written and the procedures. This allows the main program to grow if necessary. We will start at 1000, which is just a nice round number (in fact we do just this with most of the programs published in BEEBUG). The group of instructions that makes up each procedure is sandwiched between the two statements 'DEF PROC' and 'ENDPROC', that mark the start and finish of that procedure. For example, PROCbegin could be defined as follows to put a title at the head of the screen.

```
1000 DEF PROCbegin
1010 COLOUR 3
1020 PRINTTAB(3,1)"Build a House"
1030 ENDPROC
1040 :
```

Let's define some of the other procedures as well. PROCask is rather more complicated. The program not only needs to ask for the name of the next part of the house to be drawn, but also needs to delete the previous reply. Here is a procedure that will do that, and also the procedures to draw the walls, roof, and door of the house.

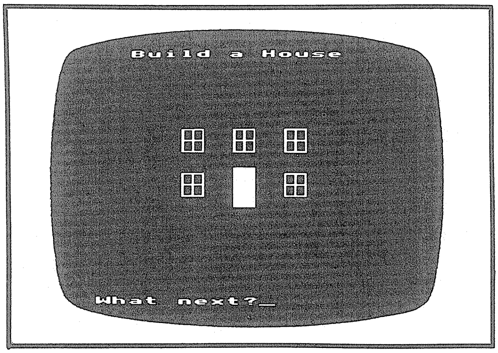
```
1100 DEF PROCask
1110 PRINTTAB(11,30)SPC9;
1120 INPUT TAB(1,30)"What next",answer
1130 ENDPROC
1140 :
1200 DEF PROCwalls
1210 GCOL0,1
1220 MOVE 340,400:DRAW 340,700
1230 PLOT 85,940,400:PLOT 85,940,700
1240 ENDPROC
1250 :
1300 DEF PROCroof
1310 GCOL0,5
1320 MOVE320,700:MOVE 400,800
1330 PLOT 85,960,700:PLOT 85,880,800
1340 ENDPROC
1350 :
1400 DEF PROCdoor
1410 GCOL0,6
1420 MOVE600,400:MOVE600,540
1430 PLOT85,680,400:PLOT85,680,540
1440 ENDPROC
1450 :
```

If you add these lines to the main program you can now test out what you have done so far. Run the program, and if you type in 'walls' or 'roof' in response to the question, then you should see the appropriate part of the house appear on the screen. Note that I have chosen to use lower case for the names in my version. You could change this if you wished.


Build a House
What next?_

If you type in a word for another part, like 'chimney' or 'garden' then you will get an error message because we haven't yet written these procedures. Anything else typed in will just cause the question to be asked again, except that typing 'finish' will terminate the program.

The procedures for drawing the walls, roof and door use a combination of MOVE and PLOT instructions to plot coloured triangles to form the shapes. If you are not too sure about the use of these instructions, then look them up in the User Guide (pages 56 and 162), or your favourite Basic book. Each procedure also specifies a colour using the GCOL instruction.



These procedures are all relatively straightforward, but the next procedure (PROCwindows) will involve the use of parameters if we are to write our program well (we also looked at parameters last month). We want this procedure to draw five identical looking windows in five different places. The best way of doing this is to write one procedure that will draw a single window in such a way that we can specify where the window is to be drawn, and then to 'call' that procedure five times from within PROCwindows. Here is the procedure for drawing a single window.

```
 1500 DEF PROCwindow(x,y)
 1510 size=80
 1520 GCOL0,0
 1530 MOVEx,y:MOVEx,y+size
 1540 PLOT 85,x+size,y:PLOT 85,x+size,y
+size
 1550 GCOL0,7:MOVE x,y
 1560 DRAWx,y+size:DRAWx+size,y+size
 1570 DRAWx+size,y:DRAWx,y
 1580 MOVEx,y+size/2:DRAWx+size,y+size/2
 1590 MOVEx+size/2,y:DRAWx+size/2,y+size
 1600 ENDPROC
 1610 :
```

The values of x and y represent the position of the bottom left hand corner of the window on the screen where x is in the range from 0 to 1239 and y is in the range from 0 to 1023 (the normal graphics dimensions). The size of the window is set quite arbitrarily in this example by trial and error. Now we can define the procedure to draw all five windows as follows, simply using the single window procedure that we have already written.

```
 1700 DEF PROCwindows
 1710 PROCwindow(400,440)
 1720 PROCwindow(800,440)
 1730 PROCwindow(400,600)
 1740 PROCwindow(800,600)
 1750 PROCwindow(600,600)
 1760 ENDPROC
```

If you add these two procedures to the program you can try them out for yourself by typing in 'windows' as the part of the house to be drawn. As the program is written, if you ask for the walls to be drawn after the windows, then the windows will disappear! Each window is drawn by first displaying a black square (made of two triangles) followed by drawing white lines over the top to form the frame.

The use of parameters with procedures is a very important concept and worth spending a little more time on. In our procedure PROCwindow(x,y) the 'x' and the 'y' are really only meaningful within the procedure, and are said to be 'local' to the procedure. On the other hand, variables like 'answer$' are meaningful both in the main program and in the procedures, and we say this is 'global' to the whole program.

Remember, as we saw last month, that any of these procedures can also be tried out in immediate mode if you select the right graphics mode first (Mode 2).

Sometimes we need to use more variables inside a procedure other than just the parameters or global variables. It is always a good idea to specify them as local to the procedure by using the Basic keyword 'LOCAL'. For example, we could specify the variable 'size' in PROCwindow as being 'local' to that procedure by adding the following line:

```
1505 LOCAL size
```

There is another very important reason for knowing about this feature in BBC Basic. If we say that 'size' is local to this particular procedure, then it will not make any difference at all if we happen to have used the same variable name ('size') anywhere else in the program. The micro is able to distinguish between the different uses, all of which helps to avoid mistakes or problems when writing programs.

I hope that you can now see how very useful procedures are in designing and writing even modest programs like the one we have developed here. I will not define the remaining procedures, but leave that for you to try. You might like to consider defining PROCgarden by calling some further procedures (PROCpath, PROCgrass, PROCtrees etc), and experiment further with parameters as well. In fact, experimenting is what it's all about, both when you are learning, and when you get more experienced. Few people can sit down and write a complete program just like that. A lot of trial and error is needed to get any program to do what you want. Procedures will go a long way in helping you design and write good computer programs.

Next month I hope to talk again about procedures and show you how you can build up a collection of your own procedures to assist in program development.

POINTS ARISING

PRINTER RIBBON REJUVINATION - Jonathan Jones

We are advised that the hint that we published in Vol.2, No.10, p.38, for recycling of old printer ribbons can be detremental to the longevity of your printer head, and further, has no lasting effect on the ribbon anyway. We can only therefore advise due caution and would be pleased to hear from anybody who has any experience of this.

MODE 7 CLOCK DISPLAY - Dick Orton

Mr. Orton points out a neater solution to a section of the assembler program used in this program (Vol.2, No.6). First of all, having identified what he calls a 'cluge' (a patch to correct for an unlocated bug), the DEX instruction at line 640 should be removed and a CLC inserted before the ADC of line 680.

As he says, this emphasises the necessity for always using a CLC before a new ADC operation when using 6502 machine code.

CHAINING ASTAAD - R.W.Smith.

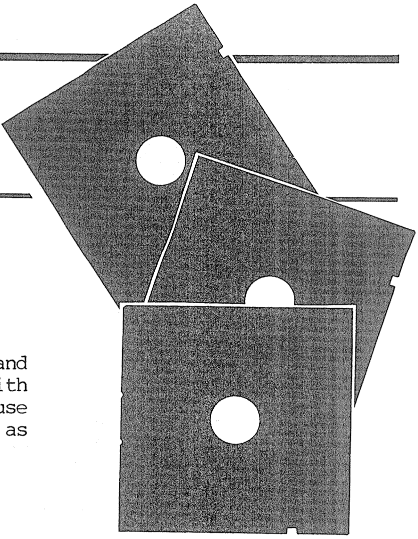
If entering the extended version of ASTAAD (Vol.2, No.9) from another program, the system variables E%, F%, U% should be reset to zero at line 370. Erroneous values could otherwise be passed on to the ASTAAD toggle switches and lead to confusion.

OUTDENTS IN WORDWISE

Further to our previous point about outdenting of text (Vol.2, No.9); a number of readers have required clarification of the point that the 'TI' command means temporary indent. Outdenting therefore, by using this with a negative number, first requires previous use of the indent command, 'IN', by a positive amount.

DYNAMIC LOADING OF FUNCTIONS AND PROCEDURES

by Hans Bakker



One of the most frustrating limitations for disc users of the BBC micro is the lack of memory. The utility described in this article overcomes these problems by dynamically loading functions and procedures into memory when required. With careful use, it could be adapted for tape use as well. Now your programs can be as long as you like.

The lack of sufficient memory on the Beeb is one of the most often quoted criticisms of this fine machine. Disc users, and users of other ROM based filing systems such as Econet and Teletext suffer an even greater loss of memory. In the higher resolution modes, a disc user can have as little as 5.5K of memory left for his own program and data.

Coping with the limitations of too little memory is a long standing problem in computing. Many mainframe computer users have had to contend with this problem, and one of the oldest solutions is that of overlays. With this technique, a large program is divided into a main module and a series of overlay modules. The main module sits permanently in memory, while each overlay module is loaded into a separately designated area of memory as and when required.

In principle, exactly the same technique is implemented by the OVERLAY utility presented here. This allows functions and procedures, which have previously been saved on disc, to be loaded into a special overlay area of memory when called by the main program. Once set up, the whole system runs quite automatically, allowing you to run programs which are effectively much larger than the available memory area. Indeed, with such a 'virtual memory system', program size is limited only by disc capacity rather than memory.

The OVERLAY utility reserves two pages of RAM (512 bytes) for a machine code routine which controls the loading and execution of the functions and procedures. It is placed in memory between &900 and &A00, which means that the system cannot be used if the RS423 system, cassette file system, the speech system or envelopes 5-16 are active at the same time. However, the user is quite free to assemble the code to an alternative location, if care is taken.

USE OF THE PROGRAM

The program OVERLAY, written in assembler, should be typed in and run. The machine code that it generates will automatically be saved to disc by the program, as a file called U.PROCFN. This is then available to *LOAD as needed. To speed up entry, comments (i.e. the parts on lines after, and including the back-slash character) may be omitted. As with all programs that use the Basic assembler, save the program before running it!

Programs that are to use this overlay technique should be structured as follows:

1. Set LOMEM to TOP+length, where 'length' is the length in bytes of the longest function or procedure that is to be loaded (found using *INFO for disc, or *CAT for cassette).

2. *LOAD U.PROCFN - this is the machine code used to control the loading of functions and procedures.

3. CALL &900 - this initialises the loader. If the code is assembled to a location other than &900 this value should be changed accordingly.

4. ON ERROR PROCfinish, followed by your own error trapping routine.

5. The normal program.

6. PROCfinish

7. END

8. Finally, the following lines should be appended to the end of your program (assuming that 10000 is higher than any line number used in the main program).

```
    10000 DEFPROCfinish
    10001 ?(TOP-1)=&FF
    10002 ENDPROC
```

Functions and procedures definitions are prepared as programs of their own, and can be SAVEd in the usual way. Their line numbers are not significant, and may clash with the calling program provided that the function or procedure doesn't reference line numbers directly (this is because any overlaid function or procedure is never seen as being part of the main program by Basic). Functions and procedures should be SAVEd under a filename within directory 'O', and with the first six letters of their name preceeded by F or P (for Function or Procedure). Thus a procedure called 'walk' would be saved as O.Pwalk, and a function called 'big' would be saved as O.Fbig. Note that while the disc system does not differentiate between lower and upper case characters, Basic does, so beware! The format can be seen in the example program, DEMO.

```
 10 REM Program DEMO
 20 REM Authors H.Bakker
 30 REM and A.France
 40 REM Version B1.1
 50 REM BEEBUG June 1984
 60 REM Program subject to copyright
 70 :
100 LOMEM=TOP+&40
110 *LOAD U.PROCFN
120 ON ERROR PROCfinish:REPORT:PRINT"
at line ";ERL:END
130 CALL &900
140 PROCastrix
150 PROCprint("This will test")
160 PROCprint("The automatic loading")
170 PROCprint("of FNs and PROCs")
180 PRINT FNlimit("---")
190 PROCastrix
200 PROCend
210 END
220 :
1000 DEFPROCfinish
1010 ?(TOP-1)=&FF
1020 ENDPROC

1000 DEF PROCastrix
1001 PRINT STRING$(39,"*")
1002 ENDPROC

1000 DEF PROCprint(A$)
1010 PRINT TAB((40-LEN(A$))/2)A$
1020 ENDPROC

1000 DEF FNlimit(A$)
1030 =A$+"Nesting of these is not poss
ible"+A$
```

LIMITATIONS

There are various limitations to the functions and procedures that the program calls. These are as follows:

1. A function or procedure that is overlaid cannot call for another one to be loaded - i.e. it can only use routines that are either in the main program (e.g. if other functions and procedures need to use them) or must be defined at the end of the function or procedure to be loaded, and saved with it. Note, however, that if the latter action is taken, the function or procedure called MUST have a unique name, otherwise Basic may get its pointers confused, with unpredictable results!

2. Avoid referencing line numbers in the function or procedure. If you must use them, make sure that your function or procedure uses high numbers, and that they are higher than any in your main program.

3. There is a limit of six characters for the name of any function or procedure that is to be loaded. This only applies to the function or procedure that calls the loading routine, and is due to the limit of seven characters as a disc filename. If a non-existent function or procedure with an illegal name is called, the routine faults it, and returns to Basic with a 'Name too long' error, and ERR=47. The limit is nine characters if using cassettes.

HOW IT WORKS

When a language detects an error in a program, it ceases execution of it by indirecting through the vector, BRKV, at &2Ø2. By redirecting this to point at user supplied code, it is possible to modify the way that program errors are dealt with. In this case, when a procedure or function that has not been defined is discovered, the error generated will cause an indirection to the machine code OVERLAY routine, which will then instigate a search of the current filing system for the missing procedures or functions. This is then *LOADed above TOP and below LOMEM. The original line of Basic calling the overlaid routine is then re-presented to Basic for execution in immediate mode before re-entering Basic.

When a new function or procedure is to be loaded, any existing overlaid routine is deleted from Basic's own catalogue of functions and procedures. This prevents Basic from jumping into 'mid-air' expecting to find an old routine.

Note that the OVERLAY utility checks for Basic I or Basic II, and adapts itself accordingly.

FOR CASSETTE USE

The OVERLAY utility can be used with cassette systems, all be it with less convenience, provided the procedures and functions are called in a known order. This limitation must be remembered, and the following modifications made to the program.

```
 15Ø space%=&CØØ
127Ø CPX #11
147Ø LDX #(comline-2) AND &FF
148Ø LDY #(comline-2) DIV &1ØØ
3Ø7Ø $P%="L."+STRING$(16,CHR$32)
3Ø8Ø comline=P%+2
315Ø *SAVE PROCFNCODE CØØ DFF
```

The main Basic program will then have to CALL &CØØ - not &9ØØ as for the disc version.

```
  1Ø REM Program OVERLAY
  2Ø REM Authors H.Bakker
  3Ø REM and A.France
  4Ø REM Version B1.6
  5Ø REM BEEBUG June 1984
  6Ø REM Program subject to copyright
  7Ø :
 1ØØ oscli=&FFF7
 11Ø brkvec=&2Ø2
 12Ø textp=&B
 13Ø top=&12
 14Ø sysvar=&4ØØ
 15Ø space%=&9ØØ
 16Ø varloc=&3A
 17Ø varpos=&2A
 18Ø FOR pass=ØTO 3 STEP 3
 19Ø P%=space%
 2ØØ [OPT pass
 21Ø   LDA brkvec      \Check to see if
 22Ø   CMP #begin AND 255
 23Ø   BNE notyetsaved
 24Ø   LDA brkvec+1    \pointers are
 25Ø   CMP #begin DIV 256
 26Ø   BEQ over        \set up already
 27Ø .notyetsaved
 28Ø   LDA brkvec      \Save old vector
 29Ø   STA brksav
 3ØØ   LDA brkvec+1
 31Ø   STA brksav+1
 32Ø   LDA #begin AND 255
 33Ø   STA brkvec      \and make new.
 34Ø   LDA #begin DIV 256
 35Ø   STA brkvec+1
 36Ø .over
 37Ø   LDA &8Ø15       \Basic I or II?
 38Ø   CMP #&31
 39Ø   BNE basic2
 4ØØ   LDA #&B6        \Basic I so
 41Ø   STA procvec     \set up
 42Ø   LDA #&92        \vectors.
 43Ø   STA procvec+1
 44Ø   LDA #&1Ø
 45Ø   STA fnvec
 46Ø   LDA #&B2
 47Ø   STA fnvec+1
 48Ø   BNE endinit
 49Ø .basic2
 5ØØ   LDA #4
 51Ø   STA procvec     \Basic II
 52Ø   LDA #&93        \similarly.
 53Ø   STA procvec+1
 54Ø   LDA #&E1
 550   STA fnvec
 560   LDA #&B1
 570   STA fnvec+1
 580 .endinit
 590   RTS
 600 .begin
 610   LDY #0
 620   LDA (&FD),Y      \Read error no.
 630   CMP #29          \No FN/PROC?
 640   BEQ findcat
 650   JMP (brksav)     \Back to Basic
 660 .findcat
 670   LDA #&FF
 680   LDA comline,Y    \Did we have
 690   CMP #ASC("P")    \FN or PROC
 700   BEQ clearproc    \loaded before?
 710   CMP #ASC("F")
 720   BEQ clearfn
 730   BNE clear        \neither!
 740 .clearproc
 750   LDY #&F6         \We have to
 760   BNE readcat      \clear the old
 770 .clearfn          \PROC/FN, so
 780   LDY #&F8         \Basic can't
 790 .readcat          \find it!
 800   JSR findvar
 810   BEQ clear
 820   JSR setbit       \hide it!
 830 .clear            \This routine
 840   LDY #0           \clears the
 850 .clear1           \command line
 860   LDA #ASC(" ")    \for oscli.
 870   STA comline,Y
 880   INY
 890   LDA comline,Y
 900   CMP #&D          \all cleared?
 910   BNE clear1       \No, so back..
 920   LDY textp-1      \Where error was
 930 .find             \Search back for
 940   LDA (textp),Y    \FN/PROC token
 950   CMP #&F2         \PROC token
 960   BEQ proc
 970   CMP #&A4         \FN token
 980   BEQ func
 990   DEY              \None found,so
1000   BPL find         \try further
1010 .proc
1020   LDA #ASC("P")    \Store F or P
1030   BNE store
1040 .func
1050   LDA #ASC("F")    \as first letter
1060 .store
1070   STA comline      \of filename.
1080   INY
1090   STY textp-1      \Store offset
1100                    \in pointer
1110   LDX #1
1120 .loop
1130   LDA (textp),Y
1140   CMP #48          \Check if alpha-
```

```
1150   BCC inctp        \numeric
1160   CMP #58
1170   BCC ok
1180   CMP #64          \If not- assume
1190   BCC inctp        \end of name
1200   CMP #91          \reached
1210   BCC ok
1220   CMP #96
1230   BCC inctp
1240   CMP #122
1250   BCS inctp
1260 .ok
1270   CPX #8           \if it is,
1280   BNE addchar      \it's too long
1290   JMP toolong      \so fault it.
1300 .addchar
1310   STA comline,X
1320   INY:INX
1330   BNE loop
1340 .inctp
1350   LDA #32          \add space after
1360   STA comline,X    \filename
1370   INX
1380   SEC              \Calculate TOP-2
1390   LDA top
1400   SBC #2           \that's where
1410   PHA              \we must load
1420   LDA top+1        \the PROC/FN
1430   SBC #0
1440   JSR conv         \Convert to hex
1450   PLA              \ASCII and put
1460   JSR conv         \ in oscli line
1470   LDX #(comline-4)AND&FF
1480   LDY #(comline-4)DIV&100
1490   JSR oscli        \execute *LOAD
1500   PLP              \Take BRK
1510   PLA              \entries from
1520   PLA              \stack
1530   LDA comline      \Was it PROC?
1540   CMP #ASC("P")
1550   BEQ proc1
1560   LDY #&F8
1570   JSR findvar      \Had we hidden
1580   BEQ restore      \this before?
1590   JSR resetbit     \get it back!
1600 .restore
1610   LDA &B           \restore text
1620   PHA              \pointer on
1630   LDA &C           \stack
1640   PHA
1650   JMP (fnvec)      \all done, so
1660                    \back to Basic
1670 .proc1
1680   LDY #&F6         \as before, but
1690   JSR findvar      \this time
1700   BEQ jproc        \for PROC
1710   JSR resetbit
1720 .jproc
1730   JMP (procvec)
1740 :
```

»

```
1750 .conv               \Routine to
1760                     \convert from
1770   PHA               \binary to ascii
1780   LSR A             \hex.
1790   LSR A             \high nibble
1800   LSR A             \first.
1810   LSR A
1820   JSR nascii
1830   PLA
1840   AND #&F           \now low nibble
1850   JSR nascii
1860   RTS
1870 .nascii
1880   CMP #10           \convert nibble
1890   BCC nas1          \to ascii and
1900   CLC               \add to command
1910   ADC #7            \line for oscli
1920 .nas1
1930   ADC #ASC("0")
1940   STA comline,X
1950   INX
1960   RTS
1970 :
1980 .findvar            \routine
1990                     \searches for
2000   LDX #0            \PROC/FN
2010 .len                \requested, and
2020   INX               \stores result
2030                     \in &2A and &2B.
2040   LDA comline,X
2050   CMP #ASC(" ")     \puts length of
2060   BNE len           \name in X
2070   STX varlen        \and stores it.
2080   LDA sysvar,Y      \on entry Y
2090                     \points to
2100                     \a linked list
2110                     \of FNs or PROCs
2120   STA varloc
2130   LDA sysvar+1,Y
2140   STA varloc+1
2150 .start
2160   LDA varloc+1
2170   BEQ endrout       \If zero, none
2180                     \have been used
2190                     \yet, so finish.
2200   LDY #0
2210   LDA (varloc),Y
2220   STA varloc+2      \store next fn
2230   INY               \or proc in
2240   LDA (varloc),Y    \varloc+2 and +3
2250   STA varloc+3
2260   INY
2270   LDA (varloc),Y    \Could this be
2280   BNE check2        \the one we are
2290   DEY               \looking for?
2300   CPY varlen
2310   BNE lencheck
2320   INY
2330   BCS storepos1
2340 .check1
2350   INY
2360   LDA (varloc),Y
2370   BEQ lencheck
2380 .check2
2390   CMP comline-1,Y\Check against
2400   BNE lencheck    \oscli command
2410   CPY varlen      \line.
2420   BNE check1
2430   INY
2440   LDA (varloc),Y
2450   BNE lencheck
2460 .storepos1
2470   LDA varloc      \It is the one
2480   STA varpos      \we are looking
2490   LDA varloc+1    \for, so store
2500   STA varpos+1    \it away.
2510 .endrout
2520   RTS
2530 .lencheck
2540   LDA varloc+3    \look for next
2550   BEQ endrout     \FN/PROC
2560   LDY #&00
2570   LDA (varloc+2),Y
2580   STA varloc
2590   INY
2600   LDA (varloc+2),Y
2610   STA varloc+1
2620   INY
2630   LDA (varloc+2),Y
2640   BNE check4
2650   DEY
2660   CPY varlen
2670   BNE start
2680   INY
2690   BCS storepos2
2700 .check3
2710   INY
2720   LDA (varloc+2),Y
2730   BEQ start
2740 .check4
2750   CMP comline-1,Y\check against
2760   BNE start       \oscli line
2770   CPY varlen
2780   BNE check3
2790   INY
2800   LDA (varloc+2),Y
2810   BNE start
2820 .storepos2
2830   LDA varloc+2
2840   STA varpos
2850   LDA varloc+3
2860   STA varpos+1
2870   RTS
2880 .setbit
2890   LDY #2          \hide FN/PROC
2900   LDA (varpos),Y \by setting top
2910   ORA #&80        \bit of the
2920   STA (varpos),Y \first character
2930   RTS             \in the name.
2940 .resetbit
```

Continued on page 25

SIX NEW GAMES REVIEWED

by David Fell and Alan Webster

Crazy Painter, together with Crazy Tracer, are two of the 'painter' games for the Beeb that are currently on the market. The basic idea is to fill in rectangular areas on the screen by totally navigating their boundaries, whilst being chased by a variety of nasty marauders.

Name	: Crazy Painter
Supplier	: Superior Software
Price	: £7.95
Reviewer	: David A. Fell
Rating	: ****
Name	: Crazy Tracer
Supplier	: Acornsoft
Price	: Tape £9.95 Disc £11.50
Reviewer	: David A. Fell
Rating	: ***

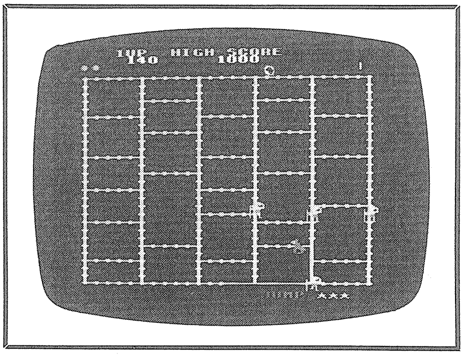
CRAZY PAINTER by Superior Software

Crazy Painter is a superb copy of the arcade game, and includes three basic screen layouts. In the first type of screen, you are a small green monkey (!) being chased by some nasty pink Indians. Your aim is to eat up the small blue dots that surround the rectangular grid work forming the screen. The Indians display a wavering intelligence; occasionally pursuing you and sometimes wandering around like aimless dodos.

By completely going round a corner square, the tune, which is perpetually played unless the 'quiet' option has been selected, will speed up. This indicates that you can temporarily stun the Indians - but beware, for this only lasts a short length of time

The second sheet pits your wits against a network of lines down which you have to guide a monkey to reach a banana for a bonus. This is much more difficult than it sounds. In the final sheet you guide a roller paint brush around another grid network, painting its borders in yellow. Unfortunately, you have to contend with a group of ferocious light blue bears and a paint supply that always has to be joined to a previously painted rectangle. Once each of three different screens has been completed, you start at the first one, but with more Indians/bears to compete with.

Overall, this game is a superb example of a non 'zap-zap' game, with attractive graphics, a pleasant tune and some realistic keys (if only all games distributors would stick to a common set of keys - but then, that is asking a lot!). If it were not for the fact that there is a small bug which very occasionally prevents you from completing a screen, I would be sorely tempted to give this game five stars.

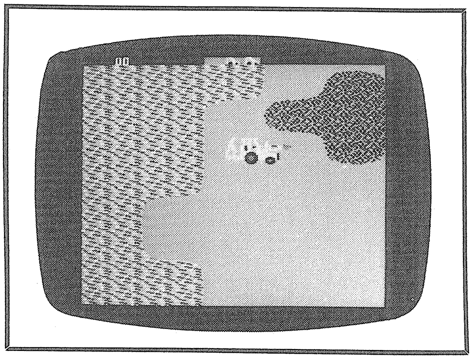


CRAZY TRACER by Acornsoft

Acornsoft's Crazy Tracer is similar in its basic concept to Crazy Painter, but lacks the same enjoyment when the game is being played. There was no music playing while the game was in progress, the keyboard didn't feel as responsive as it could have been, and the game was infuriatingly slow. The noises that were made struck me as rather unimaginative and dull. The creatures chasing you are fairly standard 'nasties', and the 'paint' mode is by no means as intelligent as the Superior Software routine.

Overall, this is not as good as earlier games from Acornsoft, who will need to produce future games to a much higher standard particularly if they are to meet the challenge of Atarisoft, who are launching their own well known computer games for the BBC micro.

Name	: JCB Digger
Supplier	: Acornsoft
Price	: Tape £9.95 Disc £11.50
Reviewer	: David A. Fell
Rating	: **

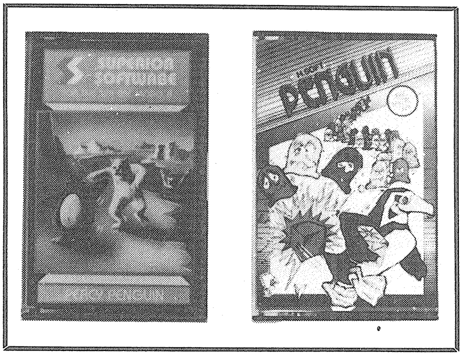


This game from Acornsoft, long announced, has at last been released, though the results do not seem to justify the wait. The basic scenario in this game is to drive a JCB 3CX Excavator Loader around two islands, scooping up the only landscape in which you can hide, and at the same time avoiding the 'Meanies'. These cannot be killed, but can be put to sleep by dumping them into the sea (fresh water won't do) or by burying them. The screen, unfortunately, scrolls in a very sluggish and disappointing way. Perhaps one could have accepted this a couple of years ago, when the Beeb was still a new entity, but now that the secrets of screen scrolling have been so successfully mastered by other software houses, one would expect Acornsoft to have produced some better scrolling.

Acornsoft produced this game in conjunction with J. C. Bamford Excavators Ltd., and I assume that this is one of the main reasons for marketing this game. However, one wonders about Acornsoft's sincerity in the comment on the rear of the packaging that "..this game is a must for the games connoisseur"!

This month sees the emergence of two more games for the BBC micro based on the very poular arcade game known as 'Pengo'. These are Penguin by H.Soft and Percy Penguin by Superior Software.

Title	: Penguin
Supplier	: H SOFT (Watford Electronics)
Price	: £7.75 + VAT
Rating	: ****
Reviewer	: Alan Webster
Title	: Percy Penguin
Supplier	: Superior Software
Price	: £7.95 inc. VAT
Rating	: ****
Reviewer	: Alan Webster



The idea of Pengo is to kill the creatures known as Snobees, by crushing them with cubes of ice, before they kill you. At the same time, your main aim, apart from staying alive, is to push three special blocks together for big bonus points. The longer you take to get the three blocks together, the fewer bonus points you gain.

PENGUIN by H Soft

This version of Pengo has transferred quite successfully to the BBC micro. The graphics, sound and speed are very good, although our copy was very fussy as to which ROMs were installed in the micro at the same time.

The music with this game is particularly noteworthy, and the key response is the best I have seen for this type of game. Unfortunately the game is spoilt by the occasional corruption of the screen display that takes place, though this seemed to be dependent on which ROMs were installed in the micro at the same time. Never the less, this is an annoying feature when it occurs.

PERCY PENGUIN by Superior Software
The second of the 'Pengo' games reviewed this month comes from Superior Software and is one of six new releases from this company.

Once again, Percy Penguin is trapped in the snow maze, trying to push the special bonus blocks together, and still trying to avoid the Snobees.

This version is not as graphically stunning as Penguin from H SOFT, but runs much better, with no problems over conflicts with any ROMs. There is also a time bonus on this version for completing a sheet quickly.

CONCLUSIONS
If it were not for the occasional corruption of the screen display then Penguin from H Soft would be the choice here. We were unable to determine the exact cause of the problem, other than that it seemed related to the selection of ROMs in our office machines. In the circumstances, Percy Penguin from Superior Software is a reliable if slightly duller choice.

Title	: Crawler
Supplier	: Watford Electronics
Price	: £6.95 + VAT
Rating	: ****
Reviewer	: Alan Webster

A long time ago, in the arcades, a game was released which caught the eye of arcade enthusiasts. The game was Centipede, and since then all of the home micros have had their own versions of the game, including the Beeb, with Bug Blaster. This version from Watford Electronics is much better, much faster, and much harder.

The game takes place in a mushroom patch full of creepie crawlies, such as centipedes, spiders, fleas and scorpions. The object is to keep these undesirables at bay, by shooting them down. The scorpion is particularly troublesome, poisoning the mushrooms, and consequently making the centipede, your principal adversary in this game, extremely difficult to deal with.

Overall, this is a very good game, and the graphics and speed are of a high standard.

Note: Both games from Watford Electronics are available to BEEBUG members at 50p off the basic price quoted above - order direct from Watford Electronics quoting your BEEBUG membership number.

Continued from page 22

```
2950    LDY #2          \restore FN/PROC
2960    LDA (varpos),Y  \by clearing
2970    AND #&7F        \that bit.
2980    STA (varpos),Y
2990    RTS
3000 .toolong
3010    BRK             \error
3020 ]
3030 ?P%=47
3040 $(P%+1)="Name too long"
3050 P%=P%+LEN$(P%+1)+2
3060 ?(P%-1)=0
3070 $P%="L.O."+STRING$(13,CHR$32)
3080 comline=P%+4
3090 P%=P%+LEN($P%)+1
3100 procvec=P%:fnvec=P%+2:P%=P%+4
3110 brksav=P%:P%=P%+2
3120 ?brksav=?brkvec:brksav?1=brkvec?1
3130 varlen=P%
3140 NEXT
3150 *SAVE U.PROCFN 900 AFF
3160 END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

TORCHNET PROBLEM
If you've been using Basic (or another language) and your machine supports the Torch O.S. MCP 0.41 with networking, then you may find that your station number has become incremented to a large number, greater than 255. You may also have found that hard breaks won't clear it back again. Poking zero to &1E8C before re-entering MCP, should reset the number to it's original status.

Tested on Basic I & II and O.S. 1·2

Programmers Workshop

STRING HANDLING

by SURAC

String handling is a very important and interesting part of programming. This month's workshop discusses four useful routines for you to experiment with, and provides a number of ideas for you to try out for yourself.

Most modern computers probably spend more of their time processing text and characters then they do handling numbers and mathematics. The use of strings and the ability to manipulate and process strings is an important part of many programs. In this workshop we will look at some basic and useful routines to show how to extract a single character from a string, how to space out text for titles, as well as converting mixed upper and lower case all into upper case, and the removal of leading and trailing spaces from a string.

The first task is a very simple one, and illustrates a useful technique that we will use in most of our other examples. Suppose we have a string, A$, and wish to extract each consecutive character beginning at the lefthand end, and proceeding to the righthand end. How do you accomplish this? The solution is in the use of the MID$ function, which allows us to extract a given character (or group of characters) from another string. There is a description of MID$ on page 298 of the User Guide, and you might like to try and work out how to use this function to extract single characters from a given string before going any further.

The program below is my routine to extract each character in turn. See if you can modify it to print the characters in reverse order; i.e. G as the first one, N as the second, etc...

```
10 A$="OUR EXAMPLE STRING"
20 FOR I%=1 TO LEN(A$)
30 B$=MID$(A$,I%,1)
40 PRINT B$
50 NEXT
```

In general, to extract character number N% from string A$, use MID$(A$,N%,1). This will return a single character, and we need to either 'put' this into another string variable, print it immediately, or to use it as a 'parameter' for another function. A$, N% and 1 are all parameters to the function MID$ in the example above. In the examples that follow, we will use this technique of extracting characters to perform some simple operations on strings.

SPACING OUT TEXT

When writing programs, it is nice to display a banner at the start giving the program name. To distinguish the name from the rest of any text that may be present on the screen, it is convenient to have some method of 'highlighting' this banner. One way to do this is to spread the string out to twice its original length by padding it out with a space after every character. Below is the function and a demonstration program that illustrates the string padding function. Type it in and run it:

```
  10 REPEAT
  20 INPUT A$
  30 PRINT FNpad(A$)
  40 UNTIL A$=""
  50 END
  60 :
1000 DEF FNpad(A$)
1010 LOCAL I%,B$
1020 IF A$="" THEN GOTO 1070
1030 IF LEN(A$)>127 PRINT"STRING TOO
LONG TO PAD!":STOP
```

»

```
1040 FOR I%=1 TO LEN(A$)
1050 B$=B$+MID$(A$,I%,1)+" "
1060 NEXT
1070 =B$
```

The string is passed as A$ to the function FNpad, which checks for a null (empty) string. The function also checks that the length of the padded string will not exceed the maximum string length of 255 characters (original string length less than 128 characters). Provided the string submitted passes these checks, the additional spaces are inserted. The result is displayed on the screen, and the program will continue looping until you enter a blank string, or escape.

What the program does is to step through the supplied string one character at a time, and append this character to the current output string (B$) followed by a space. This results in B$ containing a copy of A$, but with a space after each of the original characters from A$.

CONVERSION TO UPPER CASE

Our next example is based upon the use of MID$, as was the last one, but this one also uses a couple of other Basic functions. The purpose of the function is to take a string of text, and to force each character to upper case (this sort of conversion is very useful for checking user input which could be in a mixture of upper and lower case, for example in adventure games). It does this by going through the string supplied one character at a time, and performing a logical AND with the ASCII value of that character. If a logical AND with a character's ASCII code and &DF (the '&' means hexadecimal) is performed, then the value returned is that for the upper case equivalent, whether it was an upper case or lower case character that was used originally.

The program is used in much the same way as the last one; i.e. a string is typed in, and if not null the upper case equivalent is printed. If it is null, the program exits. Trying to obtain the upper case equivalent of characters other than alphabetical characters can produce some odd effects. As an exercise, try and amend the program so that it only attempts to alter a character if it is a letter, and not anything else.

```
  10 REPEAT
  20 INPUT A$
  30 PRINT FNucs(A$)
  40 UNTIL A$=""
  50 END
  60 :
1000 DEF FNucs(A$)
1010 LOCAL I%,B$
1020 IF A$="" THEN GOTO 1060
1030 FORI%=1 TO LEN(A$)
1040 B$=B$+CHR$(&DF AND ASC(MID$(A$,I%
,1)))
1050 NEXT
1060 =B$
```

STRIPPING SPACES

Our final example program this month is a slightly more complex one, but it is exceedingly useful. What it does is to take a given string, and to 'lop off' any leading or trailing spaces. Say you had a string such as the one below (with the ' marks just showing where the string starts and stops):

```
 '  AN EXAMPLE STRING    '
```

What the function does is to take off the spaces to leave:

```
'AN EXAMPLE STRING'
```

```
  10 REPEAT
  20 INPUT LINE A$
  30 PRINT "#";A$;"#"
  40 A$=FNstrip(A$)
  50 PRINT "#";A$;"#"
  60 UNTIL A$=""
  70 END
  80 :
1000 DEF FNstrip(A$)
1010 IF A$="" THEN GOTO 1040
1020 IF RIGHT$(A$,1)=" " REPEAT:A$=LEF
T$(A$,LENA$-1):UNTIL RIGHT$(A$,1)<>" "
1030 IF LEFT$(A$,1)=" " REPEAT:A$=MID$
(A$,2):UNTIL LEFT$(A$,1)<>" "
1040 =A$
```

You may be able to think of a way of using the MID$ function to make this more efficient. If you come up with any routines that you think other readers would like to see, then send them into SURAC at:

The Programmer's Workshop
BEEBUG
P.O. Box 50 St. Albans Herts.

AN INTRODUCTION TO SPREADSHEETS

by David Otley

Spreadsheets form one of the major and more serious applications of microcomputers. Ultracalc from BBC Soft is already available, and Viewsheet from Acornsoft is expected shortly. As a prelude to a review of these two spreadsheet packages, David Otley introduces the whole subject of spreadsheet analysis.

INTRODUCTION

The advent of two new spreadsheet programs indicates the increasing attention being paid to serious business oriented software for the BBC micro. The original spreadsheet program, VISICALC, is reputed to have been a major factor in the success of the Apple computer. For the first time, users were able to make the computer do what they wanted without the barrier of learning a complex programming language. The BBC's impact on the serious market has already begun with word processing. The availability of good spreadsheet packages must continue to enhance its attractiveness to serious users.

WHAT IS A SPREADSHEET?

A spreadsheet is a large sheet of paper ruled into rows and columns on which complicated calculations can be logically and neatly set out. Only the figures to be 'plugged' into the calculations can be seen however, and their corresponding answers. The computer-based version displays a small part of this tabular sheet on a monitor screen (usually some twenty rows by up to twelve columns) but permits the screen to act as a 'window' that can be moved to any part of the sheet. Numbers can be entered into the individual boxes (or cells) on the sheet and calculations performed such as the addition of column totals, the extension of quantities and prices into invoice totals, or the computation of specified statistics.

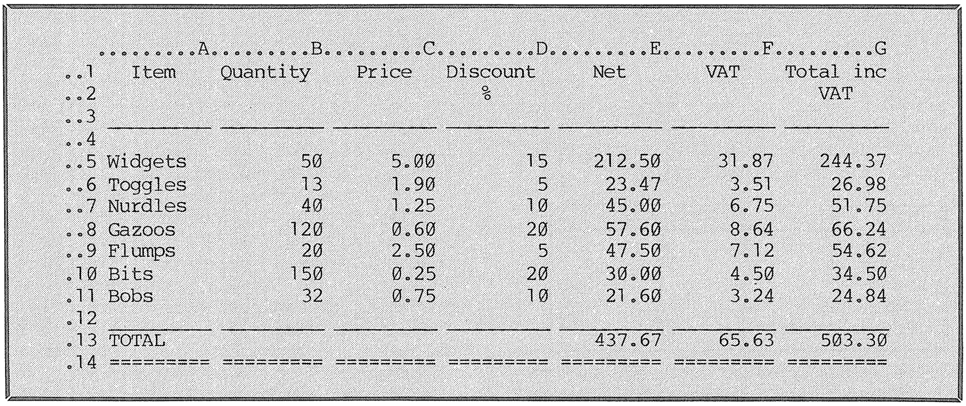
The delight of spreadsheet applications comes to the fore when you need to change the original values entered. If a new number is entered into a cell, every other number on the sheet is automatically recalculated in a matter of seconds. The sheer joy of seeing hundreds of changes calculated before your eyes, when manual methods previously took hours of tedious effort, has to be experienced to be appreciated.

The application of spreadsheets to date, has been primarily in accounting and financial planning and it's with mainframe-based financial planning packages and VISICALC on a micro that I have gained my experience. This is not to suppose that spreadsheets are limited to financial calculations however; their flexibility means that they can be used for any type of numerical manipulations such as those involved in scientific and statistical calculations; and their area of application depends, to a large extent, on the ingenuity of the user. What can be computed on a spreadsheet is about as limitless as what can be organised on a data-base filing system.

This article, the first of two, will describe this type of application in more detail, particularly for the reader with no previous experience of spreadsheet programs. In the second part, the two packages from BBC Soft and Acornsoft will be examined in some depth.

FEATURES OF SPREADSHEETS

A simple application of spreadsheet use, based on one particular package, is given in the pro-forma invoice printed out with this article. This shows how the columns are identified by letters and the rows by numbers (although these border headings could be removed in a final version for printing). The names of various items stocked are listed in the first column. The user would fill in the required quantities in the second column; prices are given in the third column and may be changed if necessary. The spreadsheet has then been programmed to calculate an appropriate quantity discount for each item (in this case on the same scale for each item, although individual discounts can easily be incorporated). It then extends the invoice to calculate net totals, VAT and the total payable. The important point to appreciate is that the necessary programming for a calculation of this kind can be entered in a few minutes, even by an inexperienced user, and then used to calculate the new results from each quantity in a few seconds.



```
        .........A.........B.........C.........D.........E.........F.........G
     ..1  Item     Quantity    Price   Discount     Net       VAT    Total inc
     ..2                                  %                             VAT
     ..3 _________ _________ _________ _________ _________ _________ _________
     ..4
     ..5 Widgets          5Ø      5.ØØ        15    212.5Ø     31.87    244.37
     ..6 Toggles          13      1.9Ø         5     23.47      3.51     26.98
     ..7 Nurdles          4Ø      1.25        1Ø     45.ØØ      6.75     51.75
     ..8 Gazoos          12Ø      Ø.6Ø        2Ø     57.6Ø      8.64     66.24
     ..9 Flumps           2Ø      2.5Ø         5     47.5Ø      7.12     54.62
     .1Ø Bits            15Ø      Ø.25        2Ø     3Ø.ØØ      4.5Ø     34.5Ø
     .11 Bobs             32      Ø.75        1Ø     21.6Ø      3.24     24.84
     .12 _________ _________ _________ _________ _________ _________ _________
     .13 TOTAL                                      437.67     65.63    5Ø3.3Ø
     .14 ========= ========= ========= ========= ========= ========= =========
```

Numbers, formulae and labels are entered directly with the program automatically distinguishing between them. Labels are left-justified and numbers right-justified by default, so the contents of columns A, B and C can be entered simply by placing the cursor in the correct box (using the cursor keys), typing the entry and pressing Return. The next cell is then accessed also using cursor keys, although advancement to the next position can be set to be automatic.

Omitting the derivation of column D for a moment, values in columns E,F and G are calculated by entering formulae in a way familiar to any Basic user, with the content of a cell being identified by its column letter and row number (e.g. E5):

```
E5=B5*C5*(1-D5/1ØØ)
F5=E5*Ø.15
G5=E5+F5
```

These formulae closely resemble Basic expressions and can include a reasonable range of in-built functions such as LOG, SIN and INT. They would be entered from the keyboard and stored with the spreadsheet but are not normally visible unless required for editing or copying.

However, having made these entries for the first row, 'Widgets', it is not necessary to re-enter equivalent formulae for the six subsequent rows. A replication device will do this for you merely by specifying the source range (i.e. E5 to G5, also stated as E5:G5) and the required destination (i.e. rows 6 to 11). For each variable used in the formula replicated, the user is prompted to specify whether the variable should be transferred exactly as it stands (i.e. E5 remains as E5 even in row 6) or whether it should be treated as a relative reference (i.e. E5 in row 5 becomes E6 in row 6). In this example, all replications should be relative. Although difficult to describe in words, the use of replication rapidly becomes second nature and is a most powerful method of writing the logic of calculations into a spreadsheet.

Next, totals are required for columns E, F and G. These can be specified by referring only to the first and last values in each column and using an automatic total feature.

Returning to column D, the discount offered depends upon the order quantity. This can be calculated using an IF statement having the format :

```
IF(logical expression,formula,formula)
```

The first formula (or value) is evaluated if the logical expression is true, the second if it is false. These statements can be nested, so a typical entry in column D reads :

```
IF(B5<1Ø,Ø,IF(B5<25,5,IF(B5<5Ø,1Ø,IF(B5
<1ØØ,15,2Ø))))
```

If such logical expressions become too complicated, lookup tables can be constructed. A lookup table is a list of values of one variable with corresponding values of a second variable. Thus a more complex set of discount percentages could be set up in a table located elsewhere on the sheet, and the correct value read off.

Order quantity	Discount %
Ø - 9	Ø
1Ø - 24	5
25 - 49	1Ø
5Ø - 99	15
1ØØ and over	2Ø

As the table might be set up on the spreadsheet quite separately from the main one, it wouldn't be displayed unless specifically requested. Such tables are particularly valuable where a relationship follows no easily represented mathematical formula (e.g. discounts offered to different customers) or formulae which are difficult to calculate each time they are required (e.g. a statistical table requiring numerical integration).

Finally, column widths and cell formats can be set so that numbers are displayed with necessary decimal places neatly aligned. If additional rows or columns are required, these can be inserted at any point; a most necessary facility in practical spreadsheet construction!

Once the results have been calculated, required portions of the sheet can then be directed to a printer (the coded row and column headings being removable) and the overall spreadsheet saved to disc or cassette for future use or amendment. Protection can be set so that other users are allowed only to enter new values for certain cells, to ensure that the model structure is not inadvertantly corrupted.

This example gives only one simple application of the general spreadsheet capability. Obvious financial applications include profit budgeting (with each column representing a week or a month), cash flow planning and investment analysis. There are further facilities that one might expect to find on larger spreadsheet 'systems' on the more capable micro's: the ability to link spreadsheets that run several sheets concurrently and carry results between them all; graphics facilities to display the results as graphs or pie-charts; spreadsheets that interface directly with certain database packages for their information, or word processors, so that tables can be neatly formatted for document printing. Non-financial applications include things like club membership records, limited resource scheduling, holiday planning charts for employees and a general home data base.

NEXT MONTH

Next month we shall be looking at two programs recently made available for the Beeb called Viewsheet and Ultracalc. Both programs possess most of the features described so far and operate in a very similar way. However there are still some significant differences in their capabilities and these we shall attempt to highlight.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

BREAK FROM WITHIN BASIC

Try CALL !-4. This has the same effect as pressing the Break key (only from within a program), but note that cassette filing system selection will take place if you enter this in immediate mode and hold the Return key down for too long.

MACHINE CODE GRAPHICS (Part 5)

by Peter Clease

In this, the concluding article in our series on machine code graphics, we present an extremely useful routine that allows a multi-coloured character to be placed anywhere on the screen with great ease.

So far I have only dealt with horizontal movement, since this is very easy to do - you add or subtract eight from the screen pointer to move two pixels right or left. If you look at the byte map for Mode 2 (given in the first article in this series, in BEEBUG Vol.2 No.8 - see also Vol.2 No.9) you will see that vertical movement is much more difficult since the boundary between character blocks vertically has to be detected and crossed, and this involves an additional increase in the pointers.

The machine code routines below, however, will do it all for you. Append them at the end of your program as an assembly subroutine and call 'plot' to plot characters. The program plots a Mode 2 8x8 character anywhere on the screen (the structure of the characters is the same as those defined earlier in this series). When calling the routine, you need to supply information about where the data is in memory, and where to put the character on the screen. This is done by making &72 and &73 point to the character data (with &72 being the low byte, and &73 being the high byte) and &70 and &71 point to the screen memory address of the top left pixel in the desired position of the shape.

```
10000 .plot LDX#0
10010 .PL1 LDY#0
10020 .PL2 LDA(&72),Y:EOR(&70),Y:STA(&7
0),Y
10030 CPY#24:BEQ PL3:CLC:TYA:ADC#8:TAY:
JMP PL2
10040 .PL3 CPX#7:BNE PL4:RTS
10050 .PL4 INX
10060 LDA&70:AND#7:CMP#7:BEQ biginc
10070 CLC:LDA&70:ADC#1:STA&70:LDA&71:AD
C#0:STA&71
10080 .PL5 CLC:LDA&72:ADC#1:STA&72:LDA&
73:ADC#0:STA&73:JMP PL1
10090 .biginc CLC:LDA&70:ADC#&79:STA&70
:LDA&71:ADC#2:STA&71:JMP PL5
```

NOTES ON SUBROUTINE

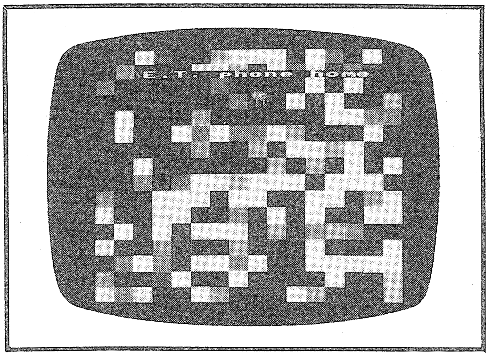
The first two lines of this routine are used to set up two counters. One of these is for the number of pixels that have been placed across the current character row (the Y register), and the other is a count of how many lines have been drawn (the X register). By altering the tests on these two registers, and a few other instructions, you could alter the size of the character plotted, but you would need to modify the character data supplied as well. The third line adds the offset to find the byte of data for each group of two pixels to be plotted for each character line, and checks to see if the character line has been fully plotted. The fourth line checks to see if the whole character has been printed, exiting if it has, running through to the next line and incrementing the X register for the next line if the routine has not finished. The remaining lines decide upon the size of the offset increment needed, and add this onto the screen pointers.

The data for the character should be arranged in column order, i.e. the first byte of data is the data for the top left pixel, the second byte is for the second byte of the first column ... the ninth byte is for the top pixel of the second column and so on. This is the same way of arranging the data that we have been using throughout these articles. How to enter the data and use the routine generally is shown in the demonstration program (Program 11). This uses the animation techniques from last month and the 'plot' subroutine already described this month to show a walking 'creature' moving across a multi-coloured background.

← 'The Plot Subroutine'

Basically, when each character is to be plotted, we want to repeat a basic sequence for each line of the character, viz: plot this line, decide upon the size of the increment necessary, add this increment, and then repeat. The more complex part of this is deciding when we have crossed the boundary between one standard character position and the one below. If we have crossed this division, then it is necessary to add on a bigger increment. Normally we add just 1 to move down to the next pixel, but we need to add &279 if we have crossed this boundary.

This program will only work in Mode 2 as it stands, since it is assumed that each character needs a data block that is four by eight bytes in size. To convert it to Mode Ø or Mode 4, change the CPY value at the start of line 1ØØ3Ø to 8; this alters the program so that each line will then only require 1 byte per horizontal line to specify all 8 pixels. To convert it to Mode 1 or Mode 5, change the CPY value to 16 (two bytes per horizontal line). These differences are summarised in the table below. In all modes, the data reads down the columns of bytes.

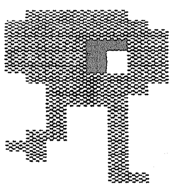


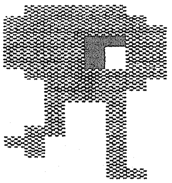
Mode	Colours	Bytes per Line	CPY
Ø,4	2	1	8
1,5	4	2	16
2	16	4	24

Table showing bytes per character line for different modes.

This program uses Exclusive-OR plotting, as you may have noticed in examining the machine code. This is the same kind of plotting as is invoked by the GCOL 3,x statement (where x is a number standing for the colour to be plotted in). The property of this method of plotting is that if you plot the same character at the same location twice, then the character will disappear, leaving the background unchanged. The outcome of this is that you can move a character very easily across any background, without any problems, provided that all actions to the screen use Exclusive-OR plotting. First plot the character, wait a while, unplot it (by plotting the same character at the same place a second time), increment the pointers and then start at the beginning of this sequence again. This gives the effect of movement. The effect is similar to the way cartoons are animated. The next program, the final demonstration program, shows nicely how the routine can be used to animate characters over a pre-defined background, without deleting any of the background through the use of Exclusive-OR plotting.

```
   1Ø REM PROGRAM 11
   2Ø REM AUTHOR PETER CLEASE
   3Ø REM VERSION B1.Ø
   4Ø REM BEEBUG JUNE 1984
   5Ø REM PROGRAM SUBJECT TO COPYRIGHT
   6Ø :
  1ØØ FORPASS=Ø TO2 STEP2
  11Ø P%=&23ØØ
  12Ø [OPTPASS
  13Ø .start LDA#Ø:STA&28ØØ:LDA#&3Ø:STA
&28Ø1
  14Ø .loop JSRfig1:JSRdelay:JSRfig1:JS
Rinc:JSRfig2:JSRdelay:JSRfig2:JSRinc:LD
A#&8Ø:BIT &FF:BMI exit:CMP&28Ø1:BPLloop
:.exit RTS
  15Ø .delay LDX#8Ø:.DE1 LDY#255:.DE2 D
EY:CPY#Ø:BNE DE2:DEX:CPX#Ø:BNE DE1:RTS
  16Ø .fig1 JSRHEAD:JSRCOORDS:LDA#&2Ø:S
TA&72:LDA#&29:STA&73:JSRplot:RTS
  17Ø .fig2 JSRHEAD:JSRCOORDS:LDA#&4Ø:S
TA&72:LDA#&29:STA&73:JSRplot:RTS
  18Ø .COORDS CLC:LDA&28ØØ:ADC#&8Ø:STA&
7Ø:LDA&28Ø1:ADC#2:STA&71:RTS
  19Ø .HEAD LDA&28ØØ:STA&7Ø:LDA&28Ø1:ST
A&71:LDA#Ø:STA&72:LDA#&29:STA&73:JSRplo
t:RTS
  2ØØ .inc CLC:LDA&28ØØ:AND#7:CMP#7:BEQ
IN1
  21Ø CLC:LDA&28ØØ:ADC#9:STA&28ØØ:LDA&2
8Ø1:ADC#Ø:STA&28Ø1:RTS
  22Ø .IN1 LDA&28ØØ:ADC#&81:STA&28ØØ:LD
A&28Ø1:ADC#2:STA&28Ø1:RTS
  23Ø .plot LDA#Ø:STA&74
  240 .PL1 LDY#0
  250 .PL2 LDA(&72),Y:EOR(&70),Y:STA(&7
0),Y
  260 CPY#24:BEQ PL3:CLC:TYA:ADC#8:TAY:
JMP PL2
  270 .PL3 LDA#7:CMP&74:BNE PL4:RTS
  280 .PL4 INC&74
  290 LDA&70:AND#7:CMP#7:BEQ biginc
  300 CLC:LDA&70:ADC#1:STA&70:LDA&71:AD
C#0:STA&71
  310 .PL5 CLC:LDA&72:ADC#1:STA&72:LDA&
73:ADC#0:STA&73:JMP PL1
  320 .biginc CLC:LDA&70:ADC#&79:STA&70
:LDA&71:ADC#2:STA&71:JMP PL5
  330 ]
  340 NEXT
  350 :
  360 MODE2
  370 VDU23,1,0;0;0;0;
  380 FORI%=0TO95
  390 READ I%?&2900
  400 NEXT
  410 FORI%=0TO1279STEP80
  420 FORJ%=0TO1023STEP64
  430 PROCblock(I%,J%,(RND(2)-1)*RND(7))
  440 NEXT,
  450 COLOUR15:PRINTTAB(3,3)"E.T. phone
 home":COLOUR7
  460 :
  470 REPEAT
  480 CALL start
  490 UNTIL 0
  500 :
  510 DATA4,12,12,12,12,12,4,0,12,12,12
,12,12,12,12,12,12,12,12,63,42,42,12,12
,0,8,12,12,12,12,12,8
  520 REM head
  530 :
  540 DATA0,0,0,4,12,4,0,0,12,8,8,8,0,0
,0,0,12,4,4,4,4,4,4,4,0,0,0,0,0,0,0,8
  550 REM tail1
  560 :
  570 DATA0,0,0,0,0,0,0,0,12,8,8,8,8,8,
8,12,12,4,0,0,0,0,0,0,0,8,8,8,8,12,0,0
  580 REM tail2
  590 :
  600 DEF PROCblock(A%,B%,C%)
  610 GCOL0,C%+128
  620 VDU24,A%;B%;A%+79;B%+63;
  630 CLG
  640 ENDPROC
```





TECHNICAL NOTES ON PROGRAM 11

Line 140 is the main program loop, and also contains code to check for Escape being pressed.

Line 150 is the delay subroutine, and merely loops round doing nothing a lot of times!

Lines 160 and 170 are the two subroutines that draw the two alternate lower half characters. This provides an animated character, as was shown in part 4 of this series.

Line 180 increments the screen pointer.

Line 190 draws the head of the character, which is constant for either of the two lower halves.

Lines 200 to 220 are concerned with checking and updating the screen pointer.

Lines 230 to 320 are the plot routine, as described earlier in this article.

Lines 360 to 450 set up the screen with some different coloured backgrounds for the character to walk over.

Lines 470 to 490 repeatedly call the character routine to walk the shape across the screen.

Lines 510 to 580 contain the character data.

Finally, lines 600 to 640 fill in a small block at a given address with a given colour.

One point to note, though, is that although no net alterations result to the background, when two colours overlap, a different colour may be displayed.

I have now given all the information you need on screen structure and layout to write a machine code graphics game, but you may still be unsure of how to write one. Although it is beyond the intended scope of this series to cover this topic, you might like to consider using the following basic structure as a starting point:

```
    .Loop JSR moveinvaders
          JSR firealiens
          JSR firebase
          JSR movebase
          LDA lives
          BNE Loop
```

You will have to work out how to make the base fire and the aliens move but in doing so you will learn how to design a program's structure.

GOOD LUCK!

[We hope to follow up this series later by looking at the problems involved in designing a complete machine code game based upon the ideas presented in this series. Ed.]

TESTING OUT YOUR MICRO (Part 4)
THE CASSETTE INTERFACE

by Mike Williams

This month we continue our series on testing out your micro, by looking at the cassette interface and the use of a cassette recorder. The article provides help to ensure that your cassette recorder is properly adjusted, as well as suggestions for sorting out any problems that may arise.

In this month's article I am going to deal with the cassette interface of the BBC micro. Clearly, if you suspect a fault, and find it difficult if not impossible to load programs from cassette, it will be equally difficult to load a program to test out the interface. This month's article does not therefore include a program, but looks at a variety of practical ways with which you can set up and test out your cassette filing system.

Of course, any test of the cassette interface is as much a test of the cassette recorder and the setting of its volume and (where fitted) tone controls, as it is of the interface itself. Remember too, that with some cassette recorders, usually those designed specifically for use with a micro (like the BBC Data Recorder), the tone and volume controls have no effect on the signal fed into the computer, as the record level is controlled automatically.

SETTING UP YOUR CASSETTE RECORDER

If the tone and volume settings on your cassette recorder affect the signal fed into the computer, then you will need to establish the optimum settings for these controls. The tone control, where fitted, should normally be set quite high (certainly on all the recorders we have seen), about 8 or 9 on a scale to 1Ø.

Every BBC micro is supplied with a 'Welcome' cassette and the first program on this can be used to achieve the best setting for the volume control. In fact, this is exactly what this program is there for. Set the volume control to a low setting, start the tape going following the instructions supplied with the 'Welcome' cassette, and very slowly increase the volume setting until you continually and reliably load this test program. You may find it worth while to mark these 'correct' settings on your recorder for future reference. It is also worth checking at this stage that you can type in a short program, and reliably save and then reload this on your micro.

If you do use the 'Welcome' tape as described, remember that it is a possibility (however remote) that your machine has been supplied with a poor copy of this cassette. If you suspect this to be the case then consult with your dealer.

You can also buy 'head cleaning' tapes for use with cassette recorders, though these do have an abrasive effect, and are not to be recommended.

Cleaning fluid sold specifically for this purpose is definitely preferable. Keeping the heads clean is probably more important when it is to be used with a computer than for just playing music. One wrong note will hardly be noticeable, but one wrong bit in a program may make it unusable!

GENERAL FAULT FINDING

There are a number of practical ideas that you can try out if you suspect a problem in loading a cassette file. Knowing exactly where best to set the volume and tone controls for optimum results is clearly important, and this should ideally be done with any new cassette recorder before trying to save and load programs.

Another very good standby in case of potential failure is to have one or more cassettes, which you can normally rely on, and which you can keep just for use in situations like this. So, make sure the settings are correct, and try loading one of your 'test' tapes.

One indication of the correct working of the cassette interface is the regular clicking of the relay inside the micro during loading or saving. If you hear no clicks at all then your cassette interface, or the micro in general, may be at fault. If you hear only one initial click, and no more, then the micro is not receiving any kind of recognizable signal, even a poor one. This is exactly what will result if the cassette recorder is disconnected, for example, but may also indicate a fault in the cassette recorder.

One of the common causes of poor recorder performance is poor head alignment. With this problem, you will very often find that you can save and load your own programs, but not those from other sources. If you suspect that the head alignment on your cassette recorder needs adjustment, then you are strongly recommended to consult your dealer on this.

Another item that may cause problems is the cable connecting the cassette recorder to the micro. If one of the wires has a break in it or has become detached from the socket at either end, then this will clearly affect the system. Such a fault may not be immediately apparent. For example, it may be impossible to record, and hence load, any new programs of your own, whereas older programs, and those you purchase, may load with no problems at all. The surest way to check, is to swap your normal cable with one from a fully working system, and see if that makes any difference.

One useful idea is to listen audibly to the computer tape, either while trying to load, or just on its own. If you cannot hear anything, then assuming a good tape, the trouble is likely to be in the cassette recorder. If you are not familiar with the rather ghastly sound of a computer program, then it is worth a brief listen at least once.

Cassette tapes also vary in quality. If you often get problems trying to load programs from a particular software company, then the cause may be poor quality tapes rather than any failing in your cassette system. The same is true for tapes that you buy for your own recordings, and here it is worth paying more for known quality.

Finally, you would be surprised at how many people have been convinced they have a major problem, only to find the 'Pause' button on the cassette recorder was on!

Next month we shall be looking at a program to help test out the other ports and interfaces to be found on the BBC micro (model B).

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SINGLE KEY BAD PROGRAM RECOVER - M.C.Behrend

```
    *KEYØM%=PA.:?M%=13:M%?1=Ø:REP.REP.N%=M%+3:REP.N%=N%+1:U.?N%<32ORN%-M%>25Ø:?N%=13
    :M%?3=N%-M%:P.M%?1*256+M%?2,~M%:M%=N%:U.M%?1>127ORINKEYØ<>-1:M%?1=M%?1OR128:P."f
    urther ?":G%=GET:IFG%=89ORG%=121M%?1=M%?1A.127:U.FA.|M
```

This will recover as much meaningful program as it is able to, so that the user can list the program again and delete or amend the corrupted lines. This is convenient but not as comprehensive as the Rescue program in BEEBUG Vol.1 No.8.

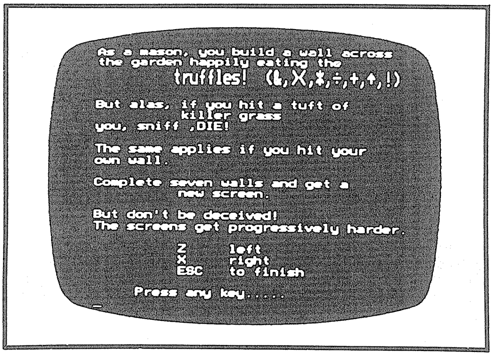
Tested on Basic I & II and O.S. 1.2

TRUFFLE HUNT (32K)

by T. Moody-Stuart

Imagine, if you will, that you're a builder and you've got to build a wall across a piece of ground. Imagine also, that you have an insatiable appetite for truffles (?!). What do you do when you find out that the said piece of ground bares richly this fruit of your desires? Well as you're a keen sort of brick-layer, you build towards these little delights of course. But there are problems (you knew that there would be, didn't you?). You've got to watch out for the Killer Grass ! (I dare say you knew that's what it would be as well).

This game is fairly self explanatory and makes good use of the Beeb's graphic and sound facilities. The 'Z' and 'X' keys control the left and right directions that your bricklayer takes with his wall. You will see it grow up the screen in plan view. When the wall goes off the top of the screen, it will immediately reappear at the bottom. Be careful not to build the wall over the top of itself though, as this is as bad as building over the killer grass. After traversing the field a few times successfully you'll find that the game will get progressively more challenging. When you have successfully completed seven walls, a new field of truffles will appear.


As a mason, you build a wall across
the garden happily eating the
truffles! (&,X,*,÷,+,↑,!)

But alas, if you hit a tuft of
killer grass
you, sniff ,DIE!

The same applies if you hit your
own wall.

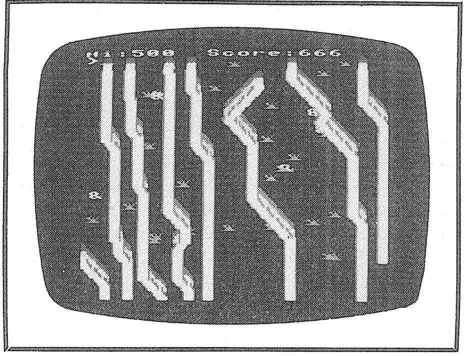
Complete seven walls and get a
new screen.

But don't be deceived!
The screens get progressively harder.

Z   left
X   right
ESC   to finish

Press any key.....

So there we are, one of the games based on gluttony and salvation rather than genocide and galactic megadeath.



```
 10 REM Program TRUFFLE HUNT
 20 REM Version B1.2
 30 REM Author T. Moody-Stuart
 40 REM BEEBUG June 1984
 50 REM Program Subject to Copyright
 60 :
100 MODE 7
110 ON ERROR GOTO 530
120 DIM A$(7)
130 hisc%=500
140 hisc$="BEEBUG"
150 PROCinit
160 REPEAT
170 MODE 2
180 PROCdefine
190 REPEAT
200 VDU 23;11,0;0;0;0;
210 PROCsetup
220 N%=0
230 REPEAT N%=N%+1
240 PROCplay
250 UNTIL dead%=1 OR N%=7
260 SC%=SC%+100
270 M%=M%+20
280 R%=(R% MOD 7)+1
290 FOR N%=1 TO 4
300 SOUND 1,-15,50+(N%*10),5
310 I%=INKEY(25)
320 NEXT N%
330 FOR N%=5 TO 2 STEP -1
340 SOUND 1,-15,50+(N%*10),5
350 I%=INKEY(25)
360 NEXT N%
370 SOUND 1,-15,100,10:I%=INKEY(50)
380 UNTIL dead%=1
390 *FX 15,0
400 SOUND 2,-15,100,10
410 I%=INKEY(100):*FX15
```

```
  42Ø IF SC%>hisc% THEN hisc%=SC%:CLS:V
DU4:COLOUR 3:INPUT TAB(5,1Ø)"Enter your
 name ",hisc$
  43Ø CLS
  44Ø VDU 4:COLOUR 5:PRINT TAB(2,9)"His
core = ";hisc%
  45Ø PRINT TAB(2,14)"Scored"TAB(9,15)"
by"TAB(12,16);hisc$
  46Ø SC%=Ø:M%=5Ø
  47Ø PRINTTAB(Ø,24)"SPACE   to continu
e"''
  48Ø PRINT"ESCAPE  to finish"
  49Ø REPEAT UNTIL GET=32
  5ØØ UNTIL FALSE
  51Ø END
  52Ø :
  53Ø ON ERROR OFF:MODE 7
  54Ø IF ERR<>17 REPORT:PRINT" at line
";ERL
  55Ø END
  56Ø :
 1ØØØ DEFPROCdefine
 1Ø1Ø X%=5ØØ
 1Ø2Ø M%=5Ø
 1Ø3Ø R%=1
 1Ø4Ø VDU 23,24Ø,8,8,28,28,62,62,62,62
 1Ø5Ø VDU 23,241,42,2Ø,42,2Ø,42,2Ø,42,2Ø
 1Ø6Ø VDU 23,242,2Ø,42,2Ø,42,2Ø,42,2Ø,42
 1Ø7Ø VDU 23,243,62,62,62,127,127,127,9
3,93
 1Ø8Ø VDU 23,244,Ø,17,1Ø6,1Ø6,6Ø,28,62,
255
 1Ø9Ø FOR YY%=1 TO 7
 11ØØ READ A$(YY%)
 111Ø NEXT YY%
 112Ø RESTORE
 113Ø DATA &,><,*,~,+,^,!
 114Ø dead%=Ø
 115Ø ENDPROC
 116Ø :
 117Ø DEFPROCsetup
 118Ø VDU 5
 119Ø CLS
 12ØØ FOR N%=1 TO M%
 121Ø T%=INT(RND(5))
 122Ø IF T%=5 THEN N$=A$(R%):T%=(R% MOD
 2)+5:ELSE N$=CHR$(244):T%=2
 123Ø GCOL Ø,T%
 124Ø MOVE RND(12ØØ),RND(1ØØØ)+2ØØ:PRIN
T N$
 125Ø NEXT N%
 126Ø ENDPROC
 127Ø :
 128Ø DEFPROCplay
 129Ø Y%=8Ø
 13ØØ REPEAT Y%=Y%+1Ø
 131Ø GCOL Ø,1
 132Ø MOVE X%,Y%    :PRINT CHR$24Ø
 133Ø MOVE X%,Y%-2Ø:PRINT CHR$241
 134Ø MOVE X%,Y%-4Ø:PRINT CHR$241
 135Ø MOVE X%,Y%-6Ø:PRINT CHR$243
 136Ø GCOL Ø,7
 137Ø MOVE X%,Y%-4Ø:PRINT CHR$242
 138Ø MOVE X%,Y%-6Ø:PRINT CHR$24Ø
 139Ø IF INKEY(-98) THEN P%=-1Ø:X%=X%+P%
 14ØØ IF INKEY(-67) THEN P%=1Ø:X%=X%+P%
 141Ø POINT%=POINT(X%+3Ø,Y%+1Ø)
 142Ø IF POINT%=3 OR POINT%=Ø THEN 146Ø
 143Ø IF POINT%=-1 THEN PROCok:GOTO 146Ø
 144Ø IF POINT%=5 OR POINT%=6 THEN SC%=
SC%+(1Ø*R%):SOUND1,-15,5Ø,5:GOTO 146Ø
 145Ø dead%=1:Y%=1ØØØ:GOTO 149Ø
 146Ø SC%=SC%+1
 147Ø VDU 4:COLOUR 3:PRINT TAB(Ø,Ø) "Hi
:";hisc%"  Score:";SC%:VDU 5
 148Ø SOUND 1,-15,2Ø,1:SOUND 2,-15,Ø,1
 149Ø UNTIL Y%=1ØØØ
 15ØØ ENDPROC
 151Ø :
 152Ø DEFPROCok
 153Ø IF P%=-1Ø THEN X%=X%+127Ø ELSE X%
=X%-127Ø
 154Ø ENDPROC
 155Ø :
 156Ø DEFPROCinit
 157Ø PRINT TAB(12,12)CHR$141 CHR$129"T
RUFFLE HUNT"
 158Ø PRINT TAB(12,13)CHR$141 CHR$129"T
RUFFLE HUNT"
 159Ø PRINT TAB(5,2Ø)"Press any key....
."
 16ØØ T%=GET
 161Ø CLS
 162Ø PRINT
 163Ø  PRINT CHR$132"As a mason, you bu
ild a wall across"
 164Ø PRINT CHR$132"the garden happily
eating the"
 165Ø PRINT TAB(8);CHR$141;CHR$134;CHR$
136"truffles!  (&,><,*,~,+,^,!)"
 166Ø PRINT TAB(8);CHR$141;CHR$134;CHR$
136"truffles!  (&,><,*,~,+,^,!)"
 167Ø  PRINT'CHR$132"But alas, if you h
it a tuft of"'TAB(1Ø);CHR$13Ø"killer gr
ass"
 168Ø  PRINT CHR$132"you, sniff ,DIE!"'
'CHR$132"The same applies if you hit your"
 169Ø PRINT CHR$132"own wall."
 17ØØ PRINT'CHR$129"Complete seven wall
s and get a"
 171Ø  PRINT TAB(1Ø);CHR$129"new screen
."''CHR$132"But don't be deceived!"
 172Ø   PRINT CHR$132"The screens get p
rogressively harder."
 173Ø PRINT'TAB(1Ø);CHR$129"Z     left"
'TAB(1Ø);CHR$129"X     right"
 174Ø PRINT TAB(1Ø);CHR$129"ESC   to fi
nish"
 175Ø PRINT TAB(5,23)"Press any key....."
 176Ø T%=GET
 177Ø ENDPROC
```

Tested on Basic I & II
and O.S. 1.2

GRAND PRIX CAR RACE (32K)

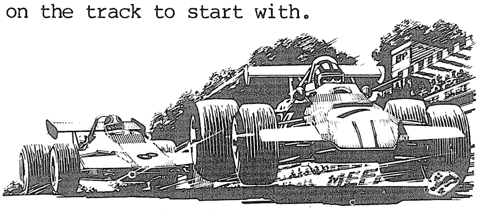
by Jeremy Graves

You know how unusual it is to find games for two players on the Beeb? Well here's one. Both players assume the roles of racing drivers, each with their own controls at opposite ends of the keyboard. It's then a question of 'driving' your car round the track as fast you can (without hitting anything) and beat your opponent across the winning line (sounds just like the real thing doesn't it?).

Acceleration and braking are provided by the 'Q' and 'A' keys for driver 1 and 'P' and ';' keys for driver 2. Cornering, and we emphasise that this IS cornering, for driver 1 and 2 respectively, is done with 'S' and 'D', and ':' and ']'. (Full instructions are provided on the screen when the program runs).

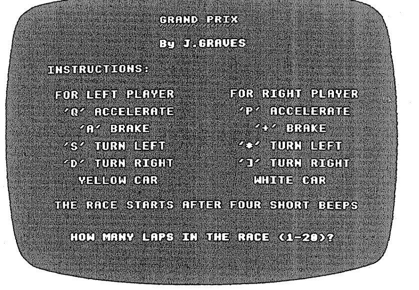
The race-track is displayed in colour on the screen, as might be seen from one of those Goodyear balloons. There are only two cars waiting on the starting grid, which is just as well (one yellow one and one white). After the fourth warning siren it's up to you. Engine revs are indicated by the pitch of the engine noise generated for each vehicle. A lap round the track may take about 2 minutes (well that's our experience anyway, but without the obstacle of a second car going round as well!). So with the option of as many as twenty laps you could set up a race lasting for as long as three quarters of an hour, if you can concentrate that long!

Due to the exacting nature of handling a formula racing car (think of the expense!), we think that there's a serious recommendation to be made that only one driver at a time is let loose on the track to start with.



TECHNICAL NOTES

Please note that the large amount of data at the end of the program is the plan for the race track; as such, any errors should be immediately apparent on the screen when you run the program. However, you could quite well change the DATA statments to redesign the track having once mastered this one.

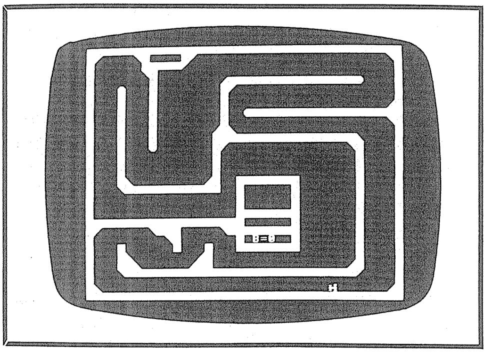


```
GRAND PRIX

By J.GRAVES

INSTRUCTIONS:

 FOR LEFT PLAYER          FOR RIGHT PLAYER
  'Q' ACCELERATE           'P' ACCELERATE
    'A' BRAKE                '+' BRAKE
  'S' TURN LEFT            '*' TURN LEFT
  'D' TURN RIGHT           ']' TURN RIGHT
    YELLOW CAR               WHITE CAR

 THE RACE STARTS AFTER FOUR SHORT BEEPS

   HOW MANY LAPS IN THE RACE (1-20)?
```

```
  10 REM PROGRAM GRAND PRIX
  20 REM AUTHOR  J.GRAVES
  30 REM VERSION B0.5
  40 REM BEEBUG  JUNE 1984
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 MODE1
 110 CLEAR
 120 PROCtitle
 130 PROCinit
 140 PROCdefine
 150 PROCtrack
 160 PROCstart
 170 TIME=0
 180 REPEAT
 190 PROCprintnew
 200 PROCrenew
 210 F%=0
 220 *FX21,0
 230 IFINKEY(-129) THENGOTO340
 240 F%=1
 250 IFINKEY(-82) THENC%(0)=C%(0)-1:IF
C%(0)=-1 THENC%(0)=7
 260 IFINKEY(-51) THENC%(0)=(C%(0)+1)M
OD8
 270 IFINKEY(-73) THENC%(1)=C%(1)-1:IF
C%(1)=-1 THENC%(1)=7
 280 IFINKEY(-89) THENC%(1)=(C%(1)+1)M
OD8
```

»

```
  29Ø IFINKEY(-17) ANDA%(Ø)<6Ø THENA%(Ø
)=A%(Ø)+1
  3ØØ IFINKEY(-66) ANDA%(Ø)>Ø THENA%(Ø)
=A%(Ø)-1
  31Ø IFINKEY(-56) ANDA%(1)<6Ø THENA%(1
)=A%(1)+1
  32Ø IFINKEY(-88) ANDA%(1)>Ø THENA%(1)
=A%(1)-1
  33Ø *FX15,Ø
  34Ø IFF%=1THENSOUND1,1,A%(Ø),255:SOUN
D2,1,A%(1),255
  35Ø FORC1%=ØTO1:X%(C1%)=X%(C1%)+(S(C%
(C1%))*A%(C1%)):Y%(C1%)=Y%(C1%)+C(C%(C1
%))*A%(C1%):NEXT
  36Ø PROCcheck
  37Ø PROCprintold
  38Ø UNTILCAR%>Ø ORL%(Ø)>=TL% ORL%(1)>
=TL%
  39Ø IFP%(Ø)>Ø ORP%(1)>Ø THENPROCcrash
:GOTO43Ø
  4ØØ *FX15
  41Ø PROCprintnew:FORD%=ØTO1ØØØØ:NEXT
  42Ø IFL%(Ø)>L%(1) THENCAR%=Ø ELSECAR%
=1
```



```
  43Ø IFCAR%=Ø THENP$="YELLOW" ELSEP$="
WHITE"
  44Ø CLS
  45Ø *FX21,Ø
  46Ø COLOUR2:PRINTTAB(15,2);"GRAND PRI
X":COLOUR1:PRINTTAB(15,3);"~~~~~~~~~~~"
  47Ø IFP%(Ø)>Ø ORP%(1)>Ø THENGOTO52Ø
  48Ø PROClaptime
  49Ø COLOUR2:PRINTTAB(9,12)"THE ";P$;"
 CAR WON"
  5ØØ PRINTTAB(2,14)"WITH A QUICKEST LA
P TIME OF ";RB%;" secs"
  51Ø GOTO53Ø
  52Ø COLOUR2:PRINTTAB(9,12)"THE ";P$;"
 CAR CRASHED"
  53Ø COLOUR3:PRINTTAB(11,17)"ANOTHER R
ACE (Y/N)"
  54Ø *FX15
  55Ø A$=GET$:IFA$<>"Y" ANDA$<>"N" THEN
GOTO54Ø
  56Ø IFA$="Y" THENGOTO1ØØ
  57Ø MODE7:PRINTTAB(1Ø,1Ø)"BYEEE"
  58Ø END
  59Ø :
 1ØØØ DEFPROCdefine
 1Ø1Ø VDU23,225,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
 1Ø2Ø VDU23,226,255,255,255,255,255,255
,255,255
 1Ø3Ø VDU23,227,255,254,252,248,24Ø,224
,192,128
 1Ø4Ø VDU23,228,255,127,63,31,15,7,3,1
 1Ø5Ø VDU23,229,1,3,7,15,31,63,127,255
 1Ø6Ø VDU23,23Ø,128,192,224,24Ø,248,252
,254,255
 1Ø7Ø VDU23,231,6Ø,126,255,255,255,255,
255,255
 1Ø8Ø VDU23,232,252,254,255,255,255,255
,254,252
 1Ø9Ø VDU23,233,255,255,255,255,255,255
,126,6Ø
 11ØØ VDU23,234,63,127,255,255,255,255,
127,63
 111Ø VDU23,235,255,255,255,9Ø,9Ø,9Ø,9Ø
,9Ø
 112Ø VDU23,236,9Ø,9Ø,9Ø,9Ø,9Ø,255,255,
255
 113Ø VDU23,237,129,66,36,24,24,36,66,1
29
 114Ø VDU23,238,9Ø,126,9Ø,24,24,219,255
,219
 115Ø VDU23,239,8,3Ø,14,223,25Ø,112,12Ø
,24
 116Ø VDU23,24Ø,224,231,66,255,255,66,2
31,224
 117Ø VDU23,241,24,12Ø,112,25Ø,223,14,3
Ø,8
 118Ø VDU23,242,219,255,219,24,24,9Ø,12
6,9Ø
 119Ø VDU23,243,24,3Ø,14,95,251,112,12Ø
,16
 12ØØ VDU23,244,7,231,66,255,255,66,231
,7
 121Ø VDU23,245,16,12Ø,112,251,95,14,3Ø
,24
 122Ø VDU23,25Ø,4,74,114,87,2Ø5,171,17Ø
,126
 123Ø VDU23,251,Ø,4,12,4Ø,5Ø,84,84,Ø
 124Ø ENDPROC
 125Ø :
 126Ø DEFPROClaptime
 127Ø RB%=1ØØØØØØ:FORC2%=1TOL%(CAR%):IF
LT%(CAR%,C2%)<RB% THENRB%=LT%(CAR%,C2%)
 128Ø NEXTC2%
 129Ø RB%=RB%/1ØØ
 13ØØ ENDPROC
 131Ø :
 132Ø DEFPROCcheck
 133Ø FORC2%=ØTO1
 134Ø IFC%(C2%)=Ø THENP%(C2%)=POINT(X%(
C2%)+16,Y%(C2%)+4)
```

»

```
 1350 IFC%(C2%)=1 THENP%(C2%)=POINT(X%(
C2%)+28,Y%(C2%)-4)
 1360 IFC%(C2%)=2 THENP%(C2%)=POINT(X%(
C2%)+32,Y%(C2%)-16)
 1370 IFC%(C2%)=3 THENP%(C2%)=POINT(X%(
C2%)+32,Y%(C2%)-32)
 1380 IFC%(C2%)=4 THENP%(C2%)=POINT(X%(
C2%)+16,Y%(C2%)-32)
 1390 IFC%(C2%)=5 THENP%(C2%)=POINT(X%(
C2%),Y%(C2%)-32)
 1400 IFC%(C2%)=6 THENP%(C2%)=POINT(X%(
C2%)-4,Y%(C2%)-16)
 1410 IFC%(C2%)=7 THENP%(C2%)=POINT(X%(
C2%),Y%(C2%))
 1420 IFX%(C2%)<SX%(0) ANDX%(C2%)>SX%(0
)-50 ANDY%(C2%)<SY%(0)+20 ANDTIME>OT%(C
2%)+4000 THENL%(C2%)=L%(C2%)+1:LT%(C2%,
L%(C2%))=TIME-OT%(C2%):OT%(C2%)=TIME:FO
RC2%=0TO1:COLOUR2+C2%:PRINTTAB(TX%(C2%)
,TY%(C2%));L%(C2%):NEXT
 1430 NEXT
 1440 IFP%(0)=0 ANDP%(1)=0 THENGOTO1460
 1450 IFP%(0)=0 THENCAR%=2 ELSECAR%=1
 1460 ENDPROC
 1470 :
 1480 DEFPROCprintold
 1490 VDU5:FORC2%=0TO1:GCOL3,2+C2%:MOVE
OX%(C2%),OY%(C2%):PRINTCHR$(238+OC%(C2%
)):NEXT
 1500 VDU4
 1510 ENDPROC
 1520 :
 1530 DEFPROCprintnew
 1540 VDU5:FORC2%=0TO1:GCOL3,2+C2%:MOVE
X%(C2%),Y%(C2%):PRINTCHR$(238+C%(C2%)):
NEXT
 1550 VDU4
 1560 ENDPROC
 1570 :
 1580 DEFPROCrenew
 1590 OX%(0)=X%(0):OY%(0)=Y%(0):OY%(1)=
Y%(1):OX%(1)=X%(1):OC%(0)=C%(0):OC%(1)=
C%(1)
 1600 ENDPROC
 1610 :
 1620 DEFPROCcrash
 1630 VDU5
 1640 PROCprintnew:VDU5
 1650 CAR%=CAR%-1
 1660 GCOL3,2+CAR%:MOVEX%(CAR%),Y%(CAR%
):PRINTCHR$(238+C%(CAR%))
 1670 GCOL0,2:MOVEX%(CAR%),Y%(CAR%):PRI
NTCHR$250
 1680 GCOL0,1:MOVEX%(CAR%),Y%(CAR%):PRI
NTCHR$251
 1690 VDU4
 1700 *FX15
 1710 SOUND0,-10,14,25:FORD%=0TO10000:N
EXT
 1720 ENDPROC
 1730 :
 1740 DEFPROCstart
 1750 PROCprintnew:TIME=0
 1760 FORC2%=0TO6:COLOUR1+(C2% MOD2):PR
INTTAB(8,1);STRING$(4,CHR$226)
 1770 FORD%=0TO1900:NEXTD%
 1780 IFC2%/2=INT(C2%/2) SOUND2,-15,200
,5
 1790 NEXTC2%
 1800 COLOUR2:PRINTTAB(8,1);"*GO*"
 1810 PROCrenew:PROCprintold
 1820 ENDPROC
 1830 :
 1840 DEFPROCtitle
 1850 *FX15
 1860 VDU23;8202;0;0;0;
 1870 COLOUR2:PRINTTAB(14,2);"GRAND PRI
X":COLOUR1:PRINTTAB(14,3)"~~~~~~~~~~~"
 1880 COLOUR3:PRINTTAB(14,5)"By J.GRAVE
S":COLOUR2:PRINT''"INSTRUCTIONS:"'''" F
OR LEFT PLAYER"SPC(7)"FOR RIGHT PLAYER"
 1890 PRINT'"  'Q' ACCELERATE"SPC(8)"'P
' ACCELERATE"''"  'A' BRAKE"SPC(13)"'+'
 BRAKE"
 1900 PRINT'"  'S' TURN LEFT"SPC(9)"'*'
 TURN LEFT"''"  'D' TURN RIGHT"SPC(8)"'
]' TURN RIGHT"''SPC(4)"YELLOW CAR"SPC(1
2)"WHITE CAR"
 1910 PRINT''" THE RACE STARTS AFTER FO
UR SHORT BEEPS"
 1920 COLOUR3:PRINTTAB(3,28)SPC(39):PRI
NTTAB(3,28)"HOW MANY LAPS IN THE RACE (
1-20)";:INPUTTL%
 1930 IFTL%>20 ORTL%<1 THEN GOTO1920
 1940 CLS
 1950 ENDPROC
 1960 :
 1970 DEFPROCinit
 1980 DIMS(7),C(7),P%(1),X%(1),Y%(1),OX
%(1),OY%(1),C%(1),A%(1),OC%(1),SX%(1),S
Y%(1),LT%(1,20),OT%(1),L%(1),TX%(1),TY%
(1)
 1990 ENVELOPE 1,1,1,1,0,1,1,1,90,0,0,-
90,90,90
 2000 *FX16,0
 2010 C%=1:FORA%=45TO316STEP90:S(C%)=SI
N(RADA%):C(C%)=COS(RADA%):C%=C%+2:NEXT:
S(0)=0:S(2)=1:S(4)=0:S(6)=-1:C(0)=1:C(2
)=0:C(4)=-1:C(6)=0
 2020 X%(0)=980:Y%(0)=124:X%(1)=980:Y%(
1)=94:C%(0)=6:C%(1)=6
 2030 FORC2%=0TO1:SX%(C2%)=X%(C2%):SY%(
C2%)=Y%(C2%):NEXT
 2040 CAR%=0
 2050 ENDPROC
 2060 :
 2070 DEFPROCtrack
 2080 RESTORE3000
 2090 COLOUR1:FORC%=0TO1239STEP80
 2100 READB$:FORC2%=1TO80
 2110 B%=ASC(MID$(B$,C2%,1))-48
```

```
 212Ø VDU (225+B%)
 213Ø NEXTC2%,C%
 214Ø COLOUR2:PRINTTAB(21,18);"LAPS"
 215Ø PRINTTAB(21,21);"A=Ø":TX%(Ø)=23:T
Y%(Ø)=21
 216Ø COLOUR3:PRINTTAB(21,23);"B=Ø":TX%
(1)=23:TY%(1)=23
 217Ø ENDPROC
 218Ø :
 3ØØØ DATA"11111111111111111111111111111
11111111111112ØØØØ31111112ØØØØØØØØØØØØØØ
ØØØØØØØØØØ311"
 3Ø1Ø DATA"1ØØØØØØØ11112ØØØØØØØØØØØØØØØØ
ØØØØØØØØØØØ11ØØØØØØ31ØØØØØØØØØØØØØØØØØØØ
ØØØØØØØØØØØØ1"
 3Ø2Ø DATA"1ØØØØØØØ1ØØØØØØØØØ41111111111
1111117ØØØØØ11ØØØ6ØØØ1ØØØØØØØØ12ØØØØØØØØ
ØØØØØØØØØØØØ1"
 3Ø3Ø DATA"1ØØØ1ØØØ1ØØØØØØØØØ1ØØØØØØØØØØ
ØØØØØØØØØØØ11ØØØ1ØØØ1ØØØØØØØØ1ØØØØØØØØØØ
ØØØØØØØØØØØ41"
 3Ø4Ø DATA"1ØØØ1ØØØ1ØØØØØØØØØ1ØØ91111111
1111111111111ØØØ1ØØØ1ØØØØØØØØ1ØØØØØØØØØØ
ØØØØØØØØØØØ31"
 3Ø5Ø DATA"1ØØØ1ØØØ1ØØØØØØØØ415ØØØØØØØØØ
ØØØØØØØØØØØ11ØØØ1ØØØ1ØØØØØØØØ11111111111
1111111500001"
 3Ø6Ø DATA"1ØØØ1ØØØ8ØØØØØØØØØ12ØØØØØØØØØ
ØØØØØ31ØØØØ11ØØØ1ØØØØØØØØØØØØ1ØØØØØØØØØØ
ØØØØØØØ1ØØØØ1"
 3Ø7Ø DATA"1ØØØ1ØØØØØØØØØØØØ1ØØØØØØØØØØØ
ØØØØØØ1ØØØØ11ØØØ1ØØØØØØØØØØØØ1ØØØØØØØØØØ
ØØØØØØØ1ØØØØ1"
 3Ø8Ø DATA"1ØØØ15ØØØØØØØØØØ41ØØ11111111Ø
ØØØØØØ1ØØØØ11ØØØ31111111111111ØØ1ØØØØØØ1
ØØØØØØØ1ØØØØ1"
 3Ø9Ø DATA"1ØØØØØØØØØØØØØØØØØØØ1ØØØØØØ1Ø
ØØØØØØ1ØØØØ11ØØØØØØØØØØØØØØØØØØØ1ØØØØØØ1
ØØØØØØØ1ØØØØ1"
 31ØØ DATA"1ØØØØØØØØØØØØØØØØØØØ11111111Ø
ØØØØØØ1ØØØØ1111111111111111111111ØØØØØØ1
ØØØØØØØ1ØØØØ1"
 311Ø DATA"12ØØØØØ3111ØØØØØ31111111111Ø
ØØØØØØ1ØØØØ11ØØØØØØØØØØ31ØØØØØØØ1ØØØØØØ1
ØØØØØØØ1ØØØØ1"
 312Ø DATA"1ØØØ1ØØØØØØØ1ØØØ1ØØØ11111111Ø
ØØØØØØ1ØØØØ11ØØØ15ØØØØØØØØØØ41ØØØØØØØØØØ
ØØØØØØØ1ØØØØ1"
 313Ø DATA"1ØØØ115ØØØØØØ411ØØØØØØØØØØØØØ
ØØØØØ41ØØØØ11ØØØ311111111111111111111111
111:111200001"
 314Ø DATA"1ØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
ØØØØØØØØØØØ115ØØØØØØØØØØØØØØØØØØØØØØØØØØ
ØØØØØØØØØØØ41"
 315Ø DATA"11111111111111111111111111111
11;111111111WWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWW"
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

FASTER TRIG FUNCTIONS - J.Uys (Pretoria)

Basic programmers longer in the tooth may be aware of this useful technique. Faster use of trig. functions in Basic can be made by storing values, prior to use, in an array used as a look-up-table (if you have the space). Plotting a circle in this way can be done four times faster than by the inclusion of the calculations of the coordinates within the PLOT statement itself. For example, set up two arrays for holding sin and cos function results, construct a loop to calculate entries into these at useful intervals and, fill them up, then construct a second loop to PLOT the contents of the arrays.

USR HINT - Laurie Nicolson

Here's a reminder: If you use the USR construct to obtain the status register of the 65Ø2, do not use the DIV instruction to shift it to the right. Mask off the top byte of the integer obtained (from USR) first. This is because if the sign bit of the status register (which is the MSB of the integer) is set, then Basic thinks the integer is a signed and therefore negative 2's complement number and may give unexpected results.

IMPLEMENTING WHILE - Maarten Oosterbroek (Holland)

If you really do miss a WHILE statement, here is the offer of an equivalent structure for

```
   WHILE X%<1ØØ DO X%=X%+1
```

(Now hold on to your hats !...)

```
   REPEAT IF X%<1ØØ THEN X%=X%+1:UNTIL FALSE ELSE UNTIL TRUE
```

(but is it structured ?).

IF YOU WRITE TO US

BACK ISSUES (Members only)
All back issues are kept in print (from April 1982). Send 90p 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. Please note that the BEEBUG Reference Card and BEEBUG supplements are not supplied with back issues.

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

MEMBERSHIP COSTS:
U.K.
£5.40 for 6 months (5 issues)
£9.90 for 1 year (10 issues)
Eire and Europe
Membership £16 for 1 year.
Middle East £19
Americas and Africa £21
Elsewhere £23
Payments in Sterling preferred.

Subscriptions & Software Address
BEEBUG
PO BOX 109
High Wycombe
Bucks

Subscriptions and Software Help Line
St.Albans
(0727) 60263
Manned Mon-Fri
1pm-4pm

PROGRAMS AND ARTICLES

All programs and articles used are paid for at around £25 per page, but please give us warning of anything substantial that you intend to write. In the case of material longer than a page, we would prefer this to be submitted on cassette or disc in machine readable form using "Wordwise", "Minitext Editor" or other means. If you use cassette, please include a backup copy at 300 baud.

HINTS

There are prizes of £5 and £10 for the best hints each month.

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

Editorial Address
BEEBUG
PO Box 50
St Albans
Herts

BEEBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams.
Production Editor: Phyllida Vanstone.
Technical Assistants: David Fell, Nigel Harris and Alan Webster.
Managing Editor: Lee Calcraft.
Thanks are due to Sheridan Williams, Adrian Calcraft, Colin Lindsay, John Yale, and Tim Powys-Lybbe for assistance with this issue.

BEEBUG NEW ROM OFFER

1.2 OPERATING SYSTEM

A special arrangement has been agreed between Acorn and BEEBUG whereby BEEBUG members may obtain the 1.2 operating system in ROM at the price of £5.85 including VAT and post and packing.
The ROM will be supplied with fitting instructions to enable members to install it in their machine.
If the computer does not subsequently operate correctly, members may take their machine to an Acorn dealer for the upgrade to be tested, which will be done at a charge of £6.00 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.

FREE

NEW ROMS FOR OLD
EXCHANGE YOUR 1.0 FOR THE 1.2

We can now exchange your old 1.0 operating system for the new 1.2, free of charge. To take advantage of this offer, please send your 1.0 (supplied on eprom with a carrier board), in good condition to the address below.

£5 FOR YOUR OLD 1.0

If you have the 1.0 operating system and have already bought a 1.2, we will exchange the 1.0 (supplied on eprom with a carrier board) for a £5 voucher. This voucher may be used against any purchase from BEEBUGSOFT.

ADDRESS FOR 1.2 OS:-
ROM Offer, BEEBUG, PO Box 109, High Wycombe, Bucks, HP11 2TD.

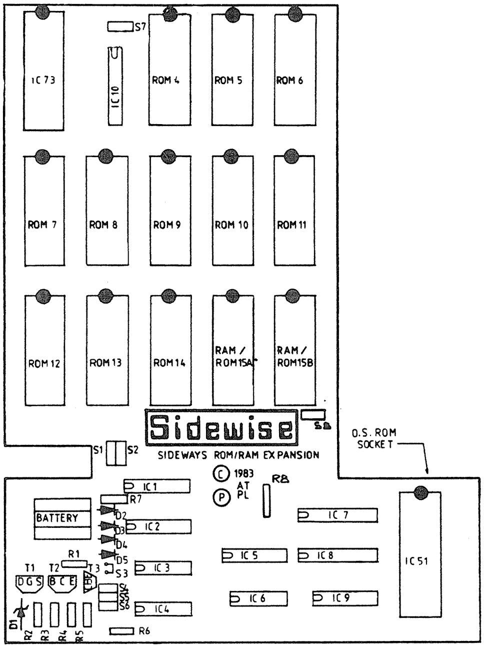
BEEBUGSOFT

ATPL'S SIDEWAYS ROM EXPANSION BOARD

SPECIAL PRICE TO MEMBERS £39.00 inc.

Save £5.70 on normal price of £44.70

- * Simply plugs into the BBC Micro
- * No soldering necessary
- * Increases the sideways ROM capacity to 16
- * Fully buffered - allows all sockets to be used
- * Complete with full and detailed instruction booklet.
- * Accepts 16K RAM in special sockets
- * Battery back up facility for RAM (parts available directly from ATPL at extra cost)
- * As used at BEEBUG
- * Reviewed in BEEBUG vol.2 number 6



<u>HOW TO ORDER</u>

Please send your order with a cheque / postal order made payable to BEEBUG, and enclose your membership number. We are unable to supply the board to overseas members.

The address for SIDEWAYS is:
BEEBUGSOFT, PO Box 1Ø9, High Wycombe, Bucks.

MAGAZINE CASSETTE OFFER

To save wear and tear on fingers and brain, we offer, each month, a cassette of the programs featured in the latest edition of BEEBUG. The first program on each tape is a menu program, detailing the tape's contents, and allowing the selection of individual programs. The tapes are produced to a high technical standard by the process used for the BEEBUGSOFT range of titles. Ordering information, and details of currently available cassettes are given below.

All previous magazine cassettes (from Vol.1 No.10) are available.

This month's cassette (Vol.3 No.2 includes: Zoom Graphics and the data file for our Zoom Treasure Hunt competition, two programs to create and play event driven music including the data for 'Jesu Joy of Man's Desiring' from Cantata No.147 by Bach, example of program design (HOUSE), utility to overlay functions and procedures, three examples from the Programmer's Workshop, Machine Code Graphics example, Truffle Hunt Game, Grand Prix Car Race game and the winning entry in our 'Niagara Falls' Brainteaser competition.

All magazine cassettes cost £3.00 each. For ordering information see separate order form in this month's magazine supplement.

MAGAZINE CASSETTE SUBSCRIPTION

We are able to offer members subscription to our magazine cassettes. Subscriptions will be for a period of one year and are for ten consecutive issues of the cassette. If required, subsriptions may be backdated as far as Vol.1 No.10, which was the first issue available on cassette. This offer is available to members only, so when applying for subscription please write to the address below, quoting your membership number and the issue from which you would like your subscription to start.

<u>CASSETTE SUBSCRIPTION ADDRESS:</u>

Please send a sterling cheque with order, together with your membership number and the date from which the subscription is to run, to:
BEEBUG, PO Box 109, High Wycombe, Bucks, HP10 8HQ.

<u>CASSETTE SUBSCRIPTION PRICE:</u>
UK £33 inc VAT and p&p
OVERSEAS (inc Eire) £39 inc p&p
(no VAT payable).

BEEBUG BINDER OFFER

<u>BEEBUG MAGAZINE BINDER OFFER</u>

A hard-backed binder for BEEBUG magazine is available. These binders are dark blue in colour with 'BEEBUG' in gold lettering on the spine. They allow you to store the whole of one volume of the magazine as a single reference book. Individual issues may be easily added or removed, thus providing ideal storage for the current volume as well.

<u>BINDER PRICE</u>
U.K. £3.90 inc p&p, and VAT.
Europe £4.90 inc p&p
(no VAT payable).
Elsewhere £5.90 inc p&p
(no VAT payable).

Make cheques payable to BEEBUG.
Send to Binder Offer, BEEBUG, PO Box 109, High Wycombe, Bucks, HP10 8HQ.
Please allow 28 days for delivery on U.K. orders.

Printed in England by Staples Printers St Albans Limited at The Priory Press. ISSN 0263-7561