INDEPENDENT NATIONAL USER GROUP

FOR THE BBC MICROCOMPUTER

BEEBUG NEWSLETTER

VOLUME 1 NUMBER 7 NOV 1982

CONTENTS

MEMBERSHIP NOW EXCEEDS 10,000

BRITAIN'S LARGEST SINGLE-MICRO USER GROUP

EDITORIAL

ACORN

ROM Replacement Charge

There appear to have been discussions about Acorn's proposed ROM replacement charge between Acorn (who were quoting around £15 a few weeks back), and the BBC (who were quoting £10). The BBC has tended to remain pretty inscrutable on this issue. The only hint that we have of their possible position appears in a statement made on 1st Nov 1982 by Colin Malone, Head of Merchandising, BBC Enterprises, to one of our members. He says that:

"there does... seem to be a moral requirement to bring the machines up to the original specification".

A recent press release from Acorn states that the upgrades will be obtainable through dealers at £10 + VAT, and that, as before, if your old operating system is in EPROM, the replacement will be made free of charge. A free replacement will also be made when Disc, Econet, Teletext, Secondary Processors or new language ROM options are added by dealers.

BEEBUG has also been in discussion with Acorn, and we hope to be able to announce a more favourable settlement than the above for owners of ROMs. If all goes well, we will announce this next month. But in the mean time, if the ROMs arrive at dealers in early December and you take your machine in, the price will probably be as above.

Guarantee Invalidation

There has been some discussion as to whether unofficial upgrades (eg adding RAM) invalidate the machine's guarantee. The October Acorn User - under the headline "Guarantee valid after DIY upgrade" implies that Chris Curry is their source of permission for DIY upgrades. It is our experience, however that staff from various sections of Acorn are still maintaining that all DIY upgrades invalidate the guarantee. A letter dated 6th October from Acorn's Technical Department to a member states categorically: "With regard to upgrading the memory on your model A, unless this is carried out by an approved service centre, this will invalidate your guarantee". It could be, of course that Chris Curry's position as implied in the "User" has just not filtered down yet. The slightly worrying thing is that some dealers are refusing to honour the guarantee when a memory upgrade has taken place. If the "Acorn User" statement is true, then we would urge Acorn to spread the word to their dealer network. We will try to get a statement from Acorn on this for next month's magazine.

BEEBUG

This issue, as well as featuring a well-thought-out game called "Racer", contains part 1 of an article on sound and envelope design, which uses a graph plotting envelope editor program to explore the Beeb's sound facilities in a simplified way. We also have a very useful program called T.BUG which will allow you to read cassettes with a corrupted block zero (cassette bug) - always assuming that you still have the tape. Next month we continue the theme of retrieving the irretrievable, and look at ways to escape from a "Bad Program" and reinstate what remains of it. We have a joystick review this issue. This is accompanied (under "Points Arising") by a joystick modification to the software library program "Starfire"; and next month we will give the listing of a joystick-based drawing/painting program, plus a definitive Screen Dump for the Seikosha printer. We are also working on a disc review which will appear shortly.

Our second software competition closed with around 100 entries; and from sampling it would appear that the standard is even higher than last time. We will not, unfortunately have time to do the judging by the next issue; though the winners will be informed by post before the appearance of the February issue (there is no January issue). If you are working on more software, we expect to announce a new competition early in the new year.

As we indicated last issue, our next magazine will cover Dec/Jan to give us a Christmas break. Christmas post permitting, it should reach you in December.

STOP PRESS - ENVELOPE EDITOR PROGRAM

The full envelope editor program (p7 this issue) will fit into 16k. To achieve this, type PAGE=&B00 before entering (or LOADing in) the program. But note that you cannot use the Bugs Fix or the User Keys.

CASSETTE BLOCK-ZERO-BUG RETRIEVE

by K Penton

Have you lost valuable programs because of the cassette bug on OS 0.1? K. Penton tells you how to recover your program from the faulty recording (if you still have it!).

The aim of this program (T.BUG) is to make it possible to recover program files which will not LOAD because they have been struck by the OS 0.1 cassette filing bug. T.BUG will not work on data files struck by the other OS 0.1 bug. The effect of the bug, is to record &28 instead of &2A as the block start marker on block 00. Hence the computer ignores the block when you try to read it back.

The program operates directly through the 6850 ACIA with a machine code program. It reads the spoilt block byte by byte, storing it in memory reserved as BLOCK, then inserts &2A in place of the faulty byte, and re-records the block.

To use T.BUG, first locate the faulty block using *CAT, and rewind the tape so it is on or just before the leader tone. Remove the cassette and load T.BUG if it is not already in memory. Now run T.BUG. In response to the prompt for the program name, this should be given exactly as for LOAD. On pressing 'return' the motor relay will switch on. Insert the cassette with the faulty block, and let the leader run for a second or so. It is essential that there be a steady leader tone from the time you press the space bar to initiate the read, to the time the data begins, as the ACIA will see any noise or glitch as the beginning of the block. If the file turns out to have a correct block start marker the program will terminate. Otherwise, insert a fresh cassette as prompted and record the corrected block. Check it has recorded properly using *CAT. If all is well, you will now be able to LOAD the corrected file using the new recording for block 00 and switching to the original for the remainder.

```
   1Ø REM == T.BUG ==
   2ØREM for misrecorded 1st block
   3ØREM  == by K.Penton = 1982 =
   4ØMODE 7
   5ØDIM BLOCK 29Ø,CODE 1ØØ
   6ØINPUT "NAME OF PROGRAM ",N$
   7ØBYTES=LEN(N$)+279
   8ØREM presuming 256 data bytes
   9ØSTART=&72:FINISH=&7Ø
  1ØØ!FINISH=BLOCK+BYTES+1
  11Ø!START=BLOCK
  12ØPROCassembler
  13Ø*MOTOR1
  14ØPRINT'"Insert tape with ";N$;" a
nd run it"
  15ØPRINT"so there is steady leader
tone sounding"
  16ØPRINT"preceding the failed block ."
  17ØPRINT'"Press space bar as leader
 is sounding"
  18ØPROCspace
  19ØPRINT''   Loading";
  2ØØCALL read_block
  21Ø*MOTORØ
  22ØIF ?BLOCK=&2A PRINT''"T.BUG can'
t help you with this one!" : END
  23ØREM == bug records &28 instead o
f &2A as block start marker ==
  24Ø?BLOCK=&2A
  25Ø!START=BLOCK
  26ØREM == reset pointer for m/c rou
tine ==
  27ØPRINT ''"Now load a blank tape a
nd"''"set it up for Record"
  28ØPRINT'"Press space bar when ready"
  29ØPROCspace
  3ØØPRINT'"Re-recording ";N$;
  31ØPROCleader
  32ØCALL record
  33ØPROCleader
  34Ø*MOTORØ
  35ØCLS
  36ØPRINT''"Now you should be able t
o load first"
  37ØPRINT "block from tape just made
 and remainder from original"
  38ØPRINT'"Use *CAT to check if you
don't want to  erase T.BUG"
  39Ø       END
  4ØØDEF PROCspace
  41Ø*FX 15,1
  42ØREPEAT UNTIL GET=32
  43ØPRINT "OK"
  44ØENDPROC
  45ØDEF PROCleader
  46Ø?&FE1Ø=&AD
  470REM switches on motor and tape t
one
  480TIME=0
  490REPEAT UNTIL TIME=500
  500ENDPROC
  510DEFPROCassembler
  520FOR PASS=0 TO 2 STEP 2
  530P%=CODE
  540[OPT PASS
  550.read_block
  560JSR INIT ;set up ACIA
  570LDX #0
  580.CONT
  590JSR TAPEBYTE ;get byte from tape
  600STA (START,X)
  610JSR INCOM ;increment pointer
  620BNE CONT
  630CLI
  640RTS
  650.record
  660JSR INIT
  670LDX #0
  680.MORE
  690LDA (START,X)
  700TAY
  710JSR SEND
  720JSR INCOM
  730BNE MORE
  740CLI
  750RTS
  760.INIT
  770SEI
  780LDA #3
  790STA &FE08  reset ACIA
  800LDA #&15 ;for 1200 Baud
  810STA &FE08 ;use &16 for 300 Baud
  820LDA #&AD
  830STA &FE10  ;switches on motor &
tape tone
  840RTS
  850.INCOM
  860INC START  ;increment start poin
ter
  870BNE P%+4 ;and compare with end p
ointer
  880INC START+1
  890LDA START
  900CMP FINISH
  910BNE OUT
  920LDA START+1
  930CMP FINISH+1
  940.OUT
  950RTS
  960.TAPEBYTE
  970LDA &FE08 ;input byte from tape
via ACIA
  980AND #1   ; test byte-received flag
  990BEQ TAPEBYTE
 1000LDA &FE09 ;data register
 1010RTS
 1020.SEND
 1030LDA &FE08 ;output byte via ACIA
 1040AND #2   ;test transmit buffer e
mpty flag
 1050BEQ SEND
 1060STY &FE09 ;Y contains the data
 1070RTS
 1080]
 1090NEXT PASS
 1100ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

RENUMBER WITH CALCULATED GOTOs

Normally a program with calculated GOTOs cannct be renumbered successfully. However Heath W. Rees points out that the RENUMBER routine is fairly simple minded and can be fooled into renumbering calculated GOTOs if they are pesented in the right form. Consider the program:

```
100 GOTO N%*10+110
110 PROCERROR
120 PROCONE :GOTO190
130 PROCTWO :GOTO190
140 PROCTHREE :GOTO190
150 REM Spaces for expansion
160 REM
170 REM
180 PROCERROR
190 REM Rest of program
```

As it stands this will not renumber correctly, but if line 100 is changed to:

```
100 GOTO 110+N%*10
```

then the RENUMBER routine doesn't get past the GOTO 110 and the RENUMBER is quite successful.

Program tested on 0·1 and 1·1 O.S.

SOUND AND ENVELOPE DESIGN (PART I)

by David Graham   Program by Ian Soutar

David Graham describes the use of an Envelope Editor program which allows full editing of Sound Envelopes, simultaneous screen plots, and instantaneous testing of all sounds created. The editor is then used to begin an exploration of the SOUND and ENVELOPE commands on the Beeb.

ENVELOPE EDITOR

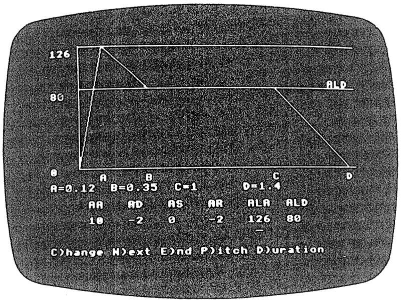
The envelope features available on the Beeb are, as you are probably well aware, pretty complex. A fourteen parameter command (ENVELOPE) is not high on the list of user-friendly notions, and the ENVELOPE command looks as if it was designed for use by people with a combined honours degree in music and computing. To produce effects that are not completely random you need ideally to sit down with a calculator and graph paper.

The program "Envelope Editor" listed at the end of the article lets the machine do all this for you; and permits, with a maximum of ease, the creation and editing of pitch and amplitude envelopes to produce the ENVELOPE parameters for the effects that you require. It is based on a program kindly submitted to us by Ian Soutar.

The program displays two alternative screens - one plots the pitch envelope, and allows editing of the pitch parameters of the ENVELOPE command. The other plots the amplitude envelope and allows editing of the ENVELOPE command's amplitude parameters. During the running of the program there are six options available:

C	(Change)	Allows parameters to be changed
N	(Next)	Moves on to next parameter
E	(End)	Switches between Pitch and Amplitude displays
P	(Pitch)	Allows editing of Pitch
D	(Duration)	Allows editing of Duration
L	(Listen)	Plays the sound

Each time a parameter is altered, a new plot is made. There is full error checking on all ENVELOPE parameters, and the sound may be heard at any time by pressing "L". It is also possible to alter Pitch and Duration, although these parameters are actually found in the SOUND rather than the ENVELOPE command.



```
A=0.12  B=0.35  C=1      D=1.4
     AA    AD    AS    AR    ALA   ALD
     10    -2    0     -2    126   80

C)hange N)ext E)nd P)itch D)uration
```

Amplitude Envelope Screen

16k VERSION

To get the program into a 16k machine, the best thing to do is to prepare two versions - one which omits the amplitude plot routine and another which omits the pitch plot routine - both versions retain full editing facilities. For most applications you will in any case probably only need to edit carefully one of the two envelopes. The pitch display version is produced by changing line 470 to:

```
470 DEF PROCA :ENDPROC
```

and deleting lines 480 to 650. The amplitude display version is produced by changing line 150 to:

```
150 DEF PROCP :ENDPROC
```

and by deleting lines 160 to 280.

32k VERSION

If you have a 32k machine, then there are a number of useful extensions to the program that you might like to add - for example a printout of envelope parameters for use when editing is complete (these are stored in the array E%); secondly it would be useful to be able to store in an array a number of sets of envelope and sound parameters that could be transferred into the editor from a menu. It would also be useful to be able to save such an array on a data tape. Even without these sophistications however, the program is really extremely useful, and an almost indispensible aid to understanding and making full use of the SOUND and ENVELOPE commands.

CREATING ENVELOPES

The fourteen parameters of the ENVELOPE command are concerned with definition of two distinct envelopes - the pitch envelope - and the amplitude envelope (page 244 of the New User Guide provides a useful explanation of what an envelope is). As the names imply, the pitch envelope defines how the pitch of a note will vary with time, and the amplitude envelope how the volume of the note will vary.

The two parameters taken together offer a considerable number of permutations, but for the purposes of simplicity we can distinguish two kinds of usage. 1) Notes of constant, or almost constant pitch, and tailored amplitude, intended to simulate natural percussive or musical sounds. 2) Sound effects which make considerable use of pitch variations. I will begin by looking at the latter, and will develop a number of sound effects using the editor. To make much sense of what follows you will really need to use the editor yourself so as to be able to see, as well as hear, the effects at each stage in the editing.

PITCH EDITING

In the following examples we are chiefly concerned with the pitch envelope, so we will leave unchanged the amplitude and duration parameters initialised by the program - and if you are using 16k, then use the pitch version of the program. The initial amplitude and duration parameters are preset as follows:

AA	AD	AS	AR	ALA	ALD	
127	0	0	-5	126	126	Duration = 20

Don't worry how this works for now - we will cover this later - but note that it gives a shape which rises sharply to a level of 126, maintains this for 1 second, then falls fairly sharply down to zero amplitude. The duration of the whole envelope is given by the value of D printed with the graph (ie 1.25 secs) - though this will change if you alter T (on the pitch screen) from the value 1.

Now go to the pitch envelope, and insert the following:

T	PI1	PI2	PI3	PN1	PN2	PN3	
1	0	0	0	40	40	40	Pitch = 50

You will see that the pitch envelope is a straight line, and if you press "L" you will hear a single non-changing note that remains of constant amplitude for 1 second, and then dies rapidly away. The frequency is determined by the value of pitch. This is part of the SOUND (third parameter) rather than the ENVELOPE command, and is altered on the editor by pressing "P". The range allowed is 0 to 255 (though the program performs no error trapping on this parameter). Try entering different values to see the effect. Note that changing P or D (see later) moves you on to the next graph due to a program quirk. Just press "E" to get back to the original screen.

VARYING THE PITCH

As things stand, the note is of constant pitch. To make it vary we need to alter one or more of the pitch change parameters PI1, PI2 or PI3. Each controls the rate of change of pitch in one of the three sectors of the pitch envelope. Try changing PI1 from 0 (no change) to 3. The pitch envelope plot shows an angled line taking the pitch up from 50 to 170, where it straightens out. If you press "L" you will hear it do this. In fact it increases at the rate of 3 units of pitch for each of 40 units of duration (since PN1 defines the duration of the first sector) - it thus increases by a total of 120 units taking it from 50 to 170.

The other two sectors can be altered accordingly. If you change PI2 to 1, you will see that in the second sector the note now continues to increase in pitch, but at only one third of the rate of the first sector. If you change PI3 to -4 the pitch will drop down by 4 x 40 units returning it to the starting value of 50. If you want to hear a number of full cycles of this sound, just increase Duration. Taking it from 20 to 60 should give 3 full cycles etc.

If you require only a two sector pitch envelope, then you can amalgamate two adjacent sectors. This is done by giving them the same PI factor. Thus if you change PI1 and PI2 both to the value 2, you will get an asymmetrical two sector inverted "V" shape envelope. If you want to make it symmetrical, just make PN1 + PN2 = PN3, and change PI3 to -2. A little experiment will show why. The values PI1 = 2, PI2 = 2, PI3 = -2, PN1 = 30, PN2 = 30, PN3 = 60 thus produce a symmetrical inverted "V" of duration 1.2 secs. If you put this with a long duration amplitude envelope it gives a siren effect. You can change the starting pitch by altering Pitch - but note that if the envelope takes the pitch outside the range 0 to 255, then there will be a wrap-around effect (so that going below zero, causes a jump to 255 etc). The audio effect can be quite interesting - though note that the screen plot does not take the wrap-around effect into account.

RAPID PITCH CYCLES

A different kind of sound effect can be achieved by executing a whole series of much shorter pitch envelopes within a single amplitude envelope. To achieve this, return Duration to 20, and enter the following values into the pitch envelope

T	PI1	PI2	PI3	PN1	PN2	PN3	
1	20	-15	-15	6	3	3	Pitch = 50

This is quite an interesting effect. Note that the duration of each pitch envelope is 0.12 secs (the 12 on the RHS of the graph), so there are about 10 full cycles in every amplitude envelope. The pitch envelope is still, as you see, an interted V; though not a completely symmetrical one. Changing the pitch to 220 to give a wrap-around makes a quite different sound. Making the envelope more oblique further changes the effect. Try the following:

T	PI1	PI2	PI3	PN1	PN2	PN3	
1	-12	-12	-12	6	4	2	Pitch = 220

There are many different effects that can be produced in this way - either by changing the overall shape of the envelope, or by using the wrap-around effect, and this can be left for experiment, though here are 3 more - the first of which produces a tremelo or warble. Each uses a pitch setting of 220.

T	PI1	PI2	PI3	PN1	PN2	PN3
1	1	-1	-1	6	3	3
1	127	-1	-1	6	3	3
1	-12	-1	-1	6	3	3

NEXT MONTH

Next month we will take a closer look at the amplitude envelope and its uses, and introduce further variations with the aid of the SOUND command.

```
   1ØREM ENVELOPE EDITOR
   2ØDIM E%(13),P$(13),l%(13,1)
   3Øp%=5Ø:d%=2Ø
   4ØFORI%=1TO13:READP$(I%):READl%(I%,
 Ø):READl%(I%,1):NEXT
   5ØDATA T,Ø,255,PI1,-128,127,PI2,-12
 8,127,PI3,-128,127,PN1,Ø,255,PN2,Ø,255
 ,PN3,Ø,255,AA,-127,127,AD,-127,127,AS,
 -127,Ø,AR,-127,Ø,ALA,Ø,126,ALD,Ø,126
   6ØMODE4:FORI%=1TO13:READE%(I%):NEXT
   7ØDATA 1,2Ø,-15,-15,6,3,3,127,Ø,Ø,-
 5,126,126
   8ØCLS:PROCA:PROCC(8,13)
   9ØCLS:PROCP:PROCC(1,7):GOTO8Ø
 1ØØEND
 11ØDEFPROCPL
 12ØENVELOPE 1,E%(1),E%(2),E%(3),E%(4
),E%(5),E%(6),E%(7),E%(8),E%(9),E%(1Ø)
,E%(11),E%(12),E%(13)
 13ØSOUND1,1,p%,d%
 14ØENDPROC
 15ØDEFPROCP:h=1152/(E%(5)+E%(6)+E%(7
))
 16ØVDU24,Ø;42Ø;1279;1Ø23;:CLG
```

»

```
  170v=p%:V=p%:FOR I%=2TO4
  180V=V+E%(I%)*E%(I%+3):IF V>v v=V
  190NEXT:V=v:v=480/v
  200MOVE1279,512:DRAW127,512:DRAW127,
1023
  210MOVE127,(512+p%*v)
  220P%=p%:Q%=0
  230FORI%=2TO 4:P%=P%+E%(I%)*E%(I%+3)
:Q%=Q%+E%(I%+3):DRAW(127+Q%*h),(512+P%
*v):NEXT
  240VDU5:MOVE0,528:PRINT"0":MOVE0,528
+p%*v:PRINT;p%:MOVE0,512+V*v:PRINT;V
  250Q%=0:FORI%=5TO7:P%=0:IFI%=5 THEN
P%=32 ELSE IF I%=7 P%=-32
  260Q%=Q%+E%(I%):MOVE63+Q%*h+P%,480:P
RINT;Q%*E%(1):NEXT
  270VDU4:VDU26
  280ENDPROC
  290DEFPROCC(S%,F%):J%=0:FORI%=S%TO F
%:J%=J%+1:PRINTTAB(J%*5,20);P$(I%);TAB
(J%*5,22);E%(I%):NEXT
  300PRINTTAB(0,26)"C)hange N)ext E)nd
 P)itch D)uration":J%=0
  310PRINT"L)isten"
  320J%=J%+1:PRINTTAB(J%*5,23)" ";
  330X$=GET$:IFX$="N"THEN430
  340IFX$="E"ENDPROC
  350IFX$="L"PROCPL:GOTO330
  360IFX$="D" PRINTTAB(0,29)d%:INPUTd%
:ENDPROC
  370IFX$="P" PRINTTAB(0,29)p%:INPUTp%
:ENDPROC
  380IFX$<>"C"VDU7:GOTO330
  390X%=S%+J%-1:PRINTTAB(0,29)"Range o
f ";P$(X%);" is ";l%(X%,0);" to ";l%(X
%,1):INPUT"Change to "E%(X%)
  400IFE%(X%)<l%(X%,0) OR E%(X%)>l%(X%
,1)VDU7:GOTO390
  410IFS%=1 PROCP ELSE PROCA
  420PRINTTAB(J%*5,22)"     ";TAB(J%*5,
 22);E%(X%):VDU28,0,31,39,29:CLS:VDU26
  430IFJ%=F%-S%+1 J%=0
  440GOTO320
  450IFS%=1 PROCP ELSE PROCA
  460GOTO330
  470DEFPROCA:VDU24,0;420;1279;1023;:C
LG
  480MOVE1279,512:DRAW127,512:DRAW127,
1023:MOVE127,512+4*E%(12):DRAW 1279,51
2+4*E%(12):MOVE127,512+4*E%(13):DRAW12
79,512+E%(13)*4:MOVE127,512
  490IFE%(8)<>0A=ABS(E%(12)/E%(8))*E%(
1)ELSE A=0
  500IFE%(9)<>0D=ABS((E%(13)-E%(12))/E
%(9))*E%(1)ELSE D=0
  510s=d%*5-A-D
  520IFE%(10)<>0S=ABS(E%(13)/E%(10))*E
%(1)ELSE S=s
  530IFS>s S=s
  540IFS<0S=0
  550IFE%(11)<>0R=E%(1)*(E%(13)-ABS(S*
E%(10)))/ABS(E%(11))ELSE R=0
  560IFR<0R=0
  570v=1120/(A+D+S+R):DRAW127+v*A,512+
4*E%(12):DRAW127+v*(A+D),512+4*E%(13)
  580H=E%(10)*S/E%(1):H=512+4*(E%(13)+
H):DRAW127+(A+D+S)*v,H:DRAW127+(A+D+S+
R)*v,512
  590VDU5:MOVE0,512:PRINT"0"
  600MOVE0,496+4*E%(12):PRINT;E%(12):M
OVE0,496+4*E%(13):PRINT;E%(13)
  610MOVE1152,546+4*E%(12):PRINTP$(12)
:MOVE1152,546+4*E%(13):PRINTP$(13)
  620PROCq(A,"A",0):PROCq(D,"B",255):P
ROCq(S,"C",511):PROCq(R,"D",786):VDU4:
ENDPROC
  630DEFPROCq(x,q$,y):IFq$="A"H=0:Q=0
  640IFx<>0H=H+x*v:Q=Q+x/100:MOVE111+H
,490:PRINTq$:MOVEy,450:PRINTq$"=";INT(
Q*100)/100
  650ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SELF VALIDATING 'GET' ROUTINE

M. Girling supplied the following useful single line GET routine:

```
10 ON INSTR("VALID",GET$) GOSUB 100,200,300,400,500 ELSE 10
```

A typical use would be when you are asking a question that requires a single key response, for instance a yes/no response. If you wanted to go to line 100 if the response was Y (or y) and 200 if an N (or n) you would use:

```
10 PRINT"Another game (Y/N)?";
20 ON INSTR("YNyn",GET$) GOTO 100,200,100,200 ELSE 20
```

You should substitute your own valid characters in the "VALID" string, and adjust the GOSUBs or GOTOs accordingly. Note that the use of ELSE corrupts the BASIC stack and therefore cannot be used inside procedures or functions.
[Ed: See the October issue where we mentioned the new release of BASIC where this bug has been cured.]

NEW CHARACTER SET FOR MODES 2 & 5 (16k/32k)

by Alan Baker

This program generates a completely new character set of 26 upper case letters for use in modes 2 or 5. The styling is different, but more importantly they are designed to give 26, rather than 20 characters per line.

It is a straightforward routine where the 26 letters of the alphabet are redefined on a 6 * 5 pixel matrix using VDU 23 commands then printed anywhere on the screen using the graphics cursor, backspacing two pixels before printing the next letter.

This technique could be extended to redefine further characters, though this is more straightforward on operating systems 1.0 and later.

```
 10REM 26 characters per line in modes 2 or 5
 15REM by ALAN BAKER August 1982
 20CLS
 30PROCinitialise
 40INPUTTAB(4,4)"X-COORDINATE (0-1279)"xcoord
 50INPUTTAB(4,6)"Y-COORDINATE (0-1023)"ycoord
 60PRINT TAB(4,8)"ENTER WORDS TO BE PRINTED"
 70INPUT TAB(4,10),a$
 80MODE 5
 90PROCalpha(xcoord,ycoord,a$)
100END
110DEF PROCinitialise
120VDU23,224,&70,&88,&88,&F8,&88,&88,0;:REM A
130VDU23,225,&F0,&50,&78,&48,&48,&F8,0;:REM B
140VDU23,226,&F8,&8E,&80,&80,&88,&F8,0;:REM C
150VDU23,227,&F8,&48,&48,&48,&48,&F8,0;:REM D
160VDU23,228,&F8,&40,&70,&40,&40,&F8,0;:REM E
170VDU23,229,&F8,&40,&70,&40,&40,&E0,0;:REM F
180VDU23,230,&F0,&80,&80,&B8,&88,&F8,0;:REM G
190VDU23,231,&88,&88,&F8,&88,&88,&88,0;:REM H
200VDU23,232,&70,&20,&20,&20,&20,&70,0;:REM I
210VDU23,233,&78,&10,&10,&10,&90,&F0,0;:REM J
220VDU23,234,&B0,&A0,&A0,&F8,&88,&88,0;:REM K
230VDU23,235,&E0,&40,&40,&48,&48,&F8,0;:REM L
240VDU23,236,&F8,&A8,&A8,&A8,&A8,&A8,0;:REM M
250VDU23,237,&88,&C8,&A8,&98,&88,&88,0;:REM N
260VDU23,238,&F8,&88,&88,&88,&88,&F8,0;:REM O
270VDU23,239,&F8,&48,&48,&78,&40,&40,0;:REM P
280VDU23,240,&F8,&88,&88,&A8,&90,&E8,0;:REM Q
290VDU23,241,&F8,&48,&48,&70,&48,&48,0;:REM R
300VDU23,242,&F8,&80,&F8,&08,&88,&F8,0;:REM S
310VDU23,243,&F8,&A8,&A8,&20,&20,&70,0;:REM T
320VDU23,244,&88,&88,&88,&88,&88,&F8,0;:REM U
330VDU23,245,&88,&88,&88,&50,&50,&20,0;:REM V
340VDU23,246,&88,&88,&A8,&A8,&A8,&50,0;:REM W
350VDU23,247,&88,&50,&20,&50,&88,&88,0;:REM X
360VDU23,248,&88,&88,&70,&20,&20,&20,0;:REM Y
370VDU23,249,&F8,&10,&20,&40,&80,&F8,0;:REM Z
380ENDPROC
390DEF PROCalpha(xcoord,ycoord,a$)
400VDU 5
410M%=1
420FOR N%=xcoord TO xcoord+(LEN(a$)*48-48)
STEP 48
430m$=MID$(a$,M%,1)
440MOVE N%,ycoord
450IF m$<>" " THEN PRINT CHR$(159+ASC(m$))
460M%=M%+1
470NEXT
480VDU 4
490ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

Jumping out of FOR....NEXT loops

The following tip was supplied by Richard Russell:
As you say in issue 4 p11, jumping out of FOR...NEXT loops is bad programming practice. However, if you are trying to convert a program written for another computer it can be difficult to re-write it using REPEAT loops, and the alternative technique of setting the loop variable to its final value won't work if its "value on exit" is used elsewhere in the program. A solution to this problem can be found if it is noted that it IS legal to jump out of a nested FOR...NEXT loop into an outer one. The "illegal" program on the left can be converted to the "legal" program on the right:

```
100 FOR I=1 TO 100
110 IF A(I)=0 THEN 130
120 NEXT I
130 ...rest of program
```

```
 95 FOR dummy=1 TO 1
100 FOR I=1 TO 100
110 IF A(I)=0 THEN 125
120 NEXT I
125 NEXT dummy
130 ...rest of program
```

JOYSTICK REVIEW

by Rob Pickering

In this article Rob Pickering reviews two types of joysticks for the BBC machine. One 'quality' joystick, and one mass production set.

The former - called the BEEBSTICK is produced by Clare's. The latter appears to be a complete look-alike for the 'official' Acorn joysticks.* It carries the word 'Acorn' moulded on the plug, and the box is coded ANH01 (the code number for Acorn's joysticks). Microage maintain that it is produced by the same manufacturer as that of the 'official' units, and they are selling it at the same price. Both are designed to work on the BBC micro with an analogue interface. This would primarily be a model B, though we will be giving full instructions on how to fit the Analogue input to a Model A in a forthcoming issue.

Although the BEEBSTICK has been made software compatible with the ANH01 joysticks, it will probably not be in direct competition with them. This is because the BEEBSTICK has been designed primarily as a high quality precision unit, the consequence is that it costs more. The ANH01 joysticks are quite obviously aimed at the budget end of the market - most probably for games, where a high degree of accuracy is hardly a major priority. Thus, the ANH01 joysticks are much cheaper.

ANH01 JOYSTICK
Supplier: Microage, 135 Hale Lane, Edgware Middx HA8 9QP. Tel: 01-959 7119
Price: £13 Date of supply: ex stock
Place of origin: Hong Kong

They are supplied as a PAIR. The case is moulded in a strong plastic; but would be better if it were half the size. As it is, the handle which contains precisely nothing, is much too big for even an average sized hand to hold comfortably. In addition to this, the finger grips cause interesting debate; the point is that no one seems to know which way round to hold the joysticks... holding them behind one's back gives the most comfortable result, though I can't seriously believe this to be 'the officially approved grip'! The nice large fire button is well designed to withstand violent pressing, the force being taken by a plastic barrier rather than the button itself. There is no facility to return the joystick to a central position when released. It has a nice light action, but perhaps just a little too light at times.

BEEBSTICK
Supplier: Clares, Providence House, 222 Townfields Road, Winsford, Cheshire
Price: £27.95 to BEEBUG members, non-members £29.95 Date of supply: ex stock
Place of origin: Britain

This is supplied as a SINGLE UNIT only. The case for this unit is an off-the-shelf box, not designed with finger grips or anything special. But, it is designed with thought! The whole box is easy for an adult to hold in one hand and operate with the other hand, while a child simply rests the box down and finds it very easy to use. I don't like the fire button. The actual button pressed is very small in diameter and although there is no trouble in finding it by touch, my complaint is that it's slightly painful to use over long periods. The fire button is quite well positioned though.

When released the joystick lever will return to its central position. The basic component is fitted with springs working in all directions, so that there is always a slight resistance on the movement away from the centre, this causes the return to the centre when released. This can be helpful during use, though I'm not sure if it is an overall advantage, since there may be times when you wish to retain a current position while you release the joystick.

GENERAL

Both types of joystick are supplied fitted with approximately one metre of

* [Acorn have confirmed that the products are indeed identical - Ed]

cable. Whereas this may cut down cost, it does make game-playing rather intimate when using a pair of them. The cable on the BEEBSTICK is 15-way ribbon cable and although very easy to replace with a longer piece, it is rather expensive cable to use... a strange choice in my opinion. The cable on the ANH01 joysticks however is nice and standard - though the plug on the end has a sealed moulding. Replacing the cable on the ANH01 joystick will mean replacing the 15-way 'D'-type connector too, and they're not cheap.

Getting inside the case is simple on both units, both are secured by four corner screws alone. Looking inside will allow you to see just what you actually get for your money. The contents of each is a switch for the fire button and a unit comprising two potentiometers operated by one free-movement lever. Both are fully proportional; that is, a small movement in any direction will give a change in reading by a corresponding amount. This is different to some other joysticks used in video games which are fitted with pressure switches; such a system can only detect a full movement or no movement in any one of a few directions.

PERSONAL PREFERENCE

My own preference is for the BEEBSTICK, though I should point out that I say this with disregard for the cost. The difference in cost is quite enormous, though worth it if you can afford that difference. One obvious target for the BEEBSTICK is EDUCATION or just simply the business world. I don't intend to test each unit to the point of destruction, but from the quality of both design and components used I would strongly suspect that a BEEBSTICK would outlast one of the ANH01 joysticks, but then it ought to at that price. Notwithstanding, the ANH01 is good value at £13 a pair.

HELP FOR THE DISABLED

Clares have also developed a heavy duty version of the BEEBSTICK fitted with a socket for an additional switch, e.g. a blow-operated switch or foot switch, both for use by the disabled. This was developed at the request of special schools in his area. To those interested, the heavy duty joystick costs £32.95.

Call for free information pack

Microware

MICROWARE (LONDON) LTD PRESENT THE "ZL"
RANGE OF DISK DRIVE SUBSYSTEMS
FOR THE BBC MICRO.

BARE DRIVES FROM ONLY	£ 125.00
IN PLASTIC ENCLOSURES	£ 135.00
DUAL UNITS WITH OWN P.S.U.	£ 295.00

INCLUDES 12 MONTHS WARRANTY ON CASED SUBSYSTEMS

EPSON PRINTERS

MX 80T/3......................£ 275.00

MX 80FT/3.....................£ 325.00

MX 100/3......................£ 425.00

PRINTER CABLE.................£ 15.00

Full range of printers carried in stock. Come and see us for a free demonstration.

PRICES DO NOT INCLUDE POSTAGE AND PACKING OR VAT.

MICROWARE (LONDON) LTD. 637 HOLLOWAY RD. LONDON N19.

PHONE 01 272 6398 FOR FURTHER DETAILS.

Microware

HARMONOGRAPH (16k)

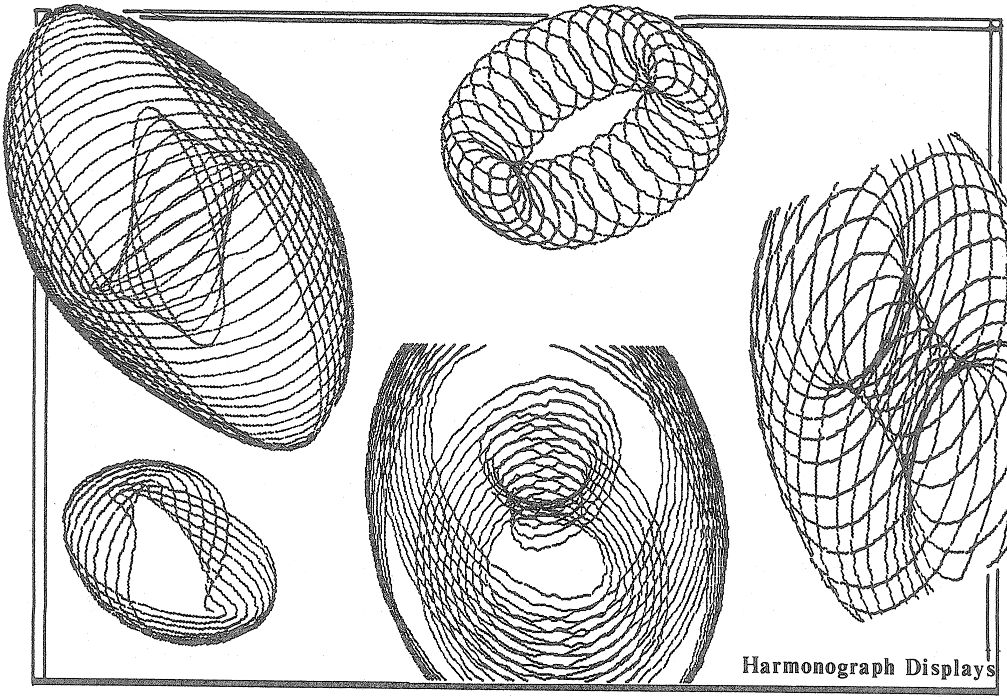
by David C. Nichols

The Harmonograph is a simple mechanical device which can draw a wide variety of complex and intricate patterns. It consists of two rigid pendula that are free to swing in any direction, with the pivot point some way down from the top; a table is attatched to the top of one pendulum and a lever with a pen is attatched to the top of the other. A drawing is made by fixing a piece of paper to the table, resting the pen on the paper, and setting the two pendula swinging. Their motion causes the pen to move and the table to 'wobble'. The combination of these two movements produces the patterns. By varying the heights of the pendulum bobs, and the amount and direction of the pendulum swings, different types of pattern can be produced.

The BASIC program shown below, "Harmony", simulates the operation of a perfect Harmonograph (friction is ignored); drawing the patterns in mode 4 on the TV screen.

To increase the execution speed of the main program loop (lines 290 to 340), the 'SIN' calculation is done by a look-up table rather than by calling the built in SIN function. The generation of this is what causes the short delay when the program is first run (lines 90 to 120). The limited size of the table is the cause of the raggedness of some of the curves drawn.

Once the program is running the drawing of the pattern can be stopped at any time by pressing the ESCAPE key, pressing the ESCAPE key again will cause the program to start drawing a new pattern. The parameters for the pattern are chosen at random (lines 170 to 210), as a consequence you may have to start several patterns off before you get an 'interesting' one.


Harmonograph Displays

The program simulates the Harmonograph by considering the motion of each pendulum to be composed of two sinusoidal components at right angles. Each component of each pendulum has an amplitude (a1,a2,a3,a4 in lines 190 and 200) and an angle of oscillation relative to some reference (r1,r2,r3,r4 in lines 170 and 180). In addition each pendulum has a frequency of oscillation (f1,f2 in line 210). The simulation consists of calculating the displacements for each motion component; adding them all together; and drawing a line to the new position (line 320). The passage of time is represented by the variable I% in the FOR-NEXT loop of lines 290 to 340.

```
 10 REM A Harmonograph Simulator
 20 REM (C) David C. Nichols 1982
 40 REM Version 3.1/1 Aug 1982
 50 REM Runs on BBC Model A or B
 60 CLS
 70 PRINT'''"Please wait"
 80 C%=255:Q%=C%/4:X%=640:Y%=512:A=400
 90 DIM sin(C%)
100 FOR I%=0 TO C%
110 sin(I%)=SIN(I%*2*PI/C%)
120 NEXT I%
130 ON ERROR GOTO 360
140 MODE 4
150 VDU 23;8202;0;0;0;
160 VDU 19,0,0;0;19,1,6;0;
170 r1=RND(C%):r2=(r1+Q%)MOD C%
180 r3=RND(C%):r4=(r3+Q%)MOD C%
190 a1=RND(A):a2=RND(A)
200 a3=RND(A):a4=RND(A)
210 f1=RND(1)*7:f2=RND(1)*9:b%=RND(C%)
220 x1=sin(r1)*a1:x2=sin(r2)*a2
230 x3=sin(r3)*a3:x4=sin(r4)*a4
240 y1=sin((Q%+r1)MOD C%)*a1
250 y2=sin((Q%+r2)MOD C%)*a2
260 y3=sin((Q%+r3)MOD C%)*a3
270 y4=sin((Q%+r4)MOD C%)*a4
280 M%=4
290 FOR I%=0 TO 200000
300 t1%=f1*I%ANDC%:t2%=b%+f2*I%ANDC%
310 t3%=Q%+t1%ANDC%:t4%=Q%+t2%ANDC%
320 PLOTM%,x1*sin(t1%)+x2*sin(t3%)
+x3*sin(t2%)+x4*sin(t4%)+X%,
y1*sin(t1%)+y2*sin(t3%)+y3*
sin(t2%)+y4*sin(t4%)+Y%
330 M%=5
340 NEXT
350 GOTO 350
360 ON ERROR GOTO 130
370 GOTO 370
```

Official BBC Programmers Kit for all BBC Microcomputer users!

consisting of:

✽100 sheet flowchart pad with de-luxe BBC grip binder ✽100 sheet screen layout pad with de-luxe BBC grip binder

✽100 sheet symbol design pad with de-luxe BBC grip binder

PLUS super quality BBC ringbinder to store your programes and notes

All items finished in Official BBC Livery

only **£15** inc VAT plus £1.00 p & p each kit

also available

BBC Computer Print-out Binders in two sizes

132 column size	**80 column size**
Price £6.99 (inc VAT) for TWO binders plus 90p p & p.	Price £4.99 (inc VAT)for TWO binders plus 90p p & p.

BBC Programmers Grip Binder

Takes two pads side by side, or, computer print-out paper.

Price £2.99 (inc VAT) plus 50p p & p.

Available from your local BBC Microcomputer dealer or direct from:

Intastor Micro Aids, FREEPOST, Stroud, Glos.,GL6 1BR

BBC BASICS

by David Graham

This month's column for the less experienced user looks at address maps, screen maps, and peeking and poking.

Memory

Microcomputer systems make use of two different kinds of memory: alterable memory (or RAM - meaning Random Access Memory), which can be both read from and written to; and non-alterable memory (either ROM meaning Read Only Memory - or EPROM meaning Electrically Programmable Read Only Memory). ROMs are totally unalterable, whereas EPROMs can be reprogrammed using ultraviolet light to erase them, and special set-ups to reprogram them.

In a fully fledged BBC machine there is 32k of RAM and 32k of ROM (or EPROM). 1k is not 1000 (as you would expect of the metric system) but 1024. Put in another way, there are 32x1024 or 32768 RAM locations into which data may be placed and subsequently retrieved; and a similar number (less the 1k set aside for other purposes) that can only be read, and not altered. This latter 31k is taken up with a 15k operating system and a 16k Basic interpreter.

Address

Each one of these 64 thousand odd locations is given an individual number by means of which the microprocessor, or central processing unit (CPU for short) identifies it. The numbers range quite simply from zero to 65535. Each number is the unique ADDRESS of one memory location.

During the day to day running of programs in Basic, you do not need to be aware of the existence of addresses or individual memory locations; the machine's Basic Interpreter handles it all for you. For example, if you want to store a piece of data, you do not have to keep a check on what memory locations are available for data storage - you just put x=75 say, and rely on the operating system to place your 75 somewhere sensible, and equally importantly, to be able to retrieve it intact when required.

There are occasions when it is useful to understand a little of what happens at the operating system level. This is essential if you are programming not in Basic but in 'Assembler' or 'Machine code', when you need to perform all data management yourself; so to store the value 75, you need to find a free memory location, instruct the machine to store 75 at that location, and you yourself need to keep tabs on where you stored it, though the Beeb's resident assembler much simplifies this task.

Memory Map

Fig 1 gives a brief MEMORY MAP of the model B. As you can see, the bottom 32k (locations 0 to 32767) are taken up with RAM, while the next 16k (locations 32768 to 49151) are used for the Basic ROM (or other so-called 'paged' ROMs), and of the last 16k (making 64k in all), the first 15k are used for the operating system ROM (or EPROM), and the last 1k for input and output devices such as the disc interface, the printer interface, the user port, the video controller chip etc. The old and new user guides give more detailed memory maps, and you may like to refer to these.

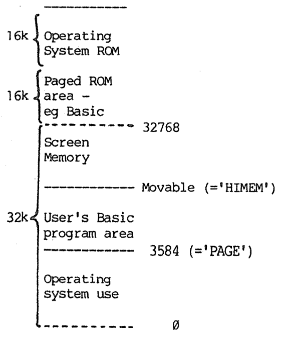


fig 1

Peeking and Poking

I said earlier that in the normal running of Basic programs you do not need to know anything about actual memory locations within the machine. There are nevertheless a set of commands in BBC Basic which enable you to access these locations directly. In standard Microsoft Basics the two most commonly used commands are PEEK and POKE. PEEK allows you to look at (or read) the contents of any location, while POKE allows you to alter its contents (provided it's RAM). In BBC Basic the two words are replaced by expressions using the '?' symbol. The expression PRINT ?n will print the contents of memory location n (where n is any integer from Ø to 65535). Try typing PRINT ?32768, the result should be 76. The address 32768 is the very first location in the operating system ROM. If you want to look at the contents of a number of locations, then the simple program below will achieve this:

```
1Ø REM Examine memory locations
2Ø PRINT TAB(5)"ADDRESS";
3Ø INPUT address
4Ø PRINT TAB(2Ø); ?address
5Ø GOTO 2Ø
```

It prompts for an address, then prints the data that it finds there. If you take a look around, you will see that the contents of all the locations are integers (whole numbers) between Ø and 255. The memory locations within the Beeb can each hold a maximum figure of 255, or put it another way, the data may take a total of 256 different values (ie 1 to 255 plus the value zero).

The capacity of each memory location is thus relatively limited. The technical term for this size of memory location is a 'byte'. This is why a 16k RAM chip is more fully termed a 16k byte RAM, since it has 16k locations, each of one byte of memory. In another issue I will look more closely at bits and bytes, and try to show why 255 (and indeed 65535) appears to be a magical number in computing - and also try to show the relevance of the so-called hexadecimal notation. For now we move on to a BBC POKE.

POKE

To alter the contents of a location, the following expession may be used

```
?n=x
```

where n is again the address of the memory location concerned, and x is the new value of data to be placed in it. This is the equivalent of POKE n,x in Microsoft Basic. Try the following sequence:

```
1Ø PRINT ?5ØØØ
2Ø ?5ØØØ=2ØØ
3Ø PRINT ?5ØØØ
```

The first number printed out will be the original contents of location 5ØØØ, and its value will depend on what you have been doing with your machine. The second printout should read 2ØØ, which means that you have successfully altered the contents of location 5ØØØ to the value required.

Before you POKE wildly all over the machine's memory, there are a few points to note. Firstly you cannot put data into ROM or EPROM (though it will not do the machine any harm if you try). Thus you can only alter the first 32k of locations. Secondly if you put data into these at random you will eventually crash the machine (still without causing any harm, though - just do a cold restart by pressing 'break' several times in rapid succession on a machine with OS Ø.1, or by pressing 'control' and 'break' together on OS 1.Ø or later. You could always switch off the machine instead). To see more precisely where you are poking, you need a more accurate memory map of the machine, and again the two user guides provide useful material. But briefly the first 3584 bytes (ie the first 3584 locations) are used in one way or another for systems purposes. At 3584 Basic program storage begins, then above this, program variables are stored (helping to explain why you lose all variables whenever you enter new program lines). From this you can guess that an address of say 5ØØØ will be free unless a Basic program of some length has been entered.

Screen Addressing

There is one use of RAM memory that has not been touched on so far. The visual display needs memory just to keep things on the screen (since a TV has no memory of its own unlike a visual display unit used on some computers). This so-called "screen memory" is provided by the top part of the RAM - that is to say from 32767 downwards. On the model A, the RAM only goes up to 16383, but due to a simple hardware trick, it behaves for the purposes of screen memory as if it also stretches from 32k downwards.

As you may know, the different graphics modes on the Beeb take up different amounts of RAM - the Teletext mode being by far the most economical using only 1k. Modes 4 and 5 take a fairly hefty 1Øk. In all screen modes there is a particular relationship between memory location (ie machine address) and the position on the screen that a particular memory location controls. This relationship is described by the so called "screen memory mapping" for the various modes. Unfortunately the mapping in modes Ø to 6 is fairly complex, but in mode 7 it is relatively simple. To find the address of the top left hand corner of the screen in mode 7 do the following:

```
MODE 7: PRINT HIMEM
```

The result should be 31744. HIMEM is one of the four pseudo-variables (see user guide or BEEBUG number 2, p15) that allows the user to discover (and sometimes alter) the way in which the Basic Interpreter has apportioned the available memory. HIMEM gives the lowest address used as screen memory. If you now type:

```
?31744=49
```

a figure 1 will appear at the top left hand corner of the screen. Location 31744 thus corresponds to the top left hand corner, and the number 49 is the ASCII code for the figure one. You can get an idea of this coding from page 486 of the user guide.

If you now execute ?31745=51, a figure 3 will now appear to the right of the figure one, and so on. There are 4Ø characters to one line, and each character position is defined by a single memory location, so that address 31784 corresponds to the leftmost character of the second line, and so on. Note however, that if you cause the screen to scroll, the screen mapping is altered somewhat.

There is plenty to experiment with here, though if you are thinking of writing programs incorporating the Beeb's equivalent of PEEK and POKE described above, it is worth noting that this will mean that they cannot be used with the second processor option, when it becomes available. Why this is so was discussed in the article "It's quicker by Tube" in BEEBUG no 6. It is therefore advisable to use the PRINT TAB( ) command for moving graphics rather than to POKE to the screen, though some loss of speed will usually result. If you are interested in examining the Beeb's memory locations in greater detail, then the article "Memory Display Utility" also in BEEBUG no 6 may be found useful.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

ON ERROR CRASH

You may like to try this program:

```
10 ON ERROR CRASH
20 ERROR
```

On our OS 1.1 it makes a horrendous buzzing sound. On our OS 0.1 all is quiet. In both operating systems the system crashes and you must perform a 'break' or power off to reset. Note that it will also crash if the word CRASH is replaced by a line number. So BEWARE. Sometimes it just erases the line that it should have gone to; sometimes it just gives "Bad program".

Thanks to Adrian Calcraft for this one.

PROCEDURE/FUNCTION LIBRARY

SORT PROCEDURES

This month we provide a most useful addition to the library in the shape of two SORT procedures.

The sorting of numerical and string data is an important function in data management programs of all kinds (Imagine an unsorted telephone book!). We present here two alternative sorts - the 'Bubble' sort and the 'Quicksort'.

The bubble sort is the easiest to follow, and many people believe it is also the shortest to program. However, the Quicksort is also a very short program provided it uses the technique of 'recursion'. (More on recursion in a future magazine).

The bubble sort suffers from the dreadful fact that in order to sort double the number of items, it takes quadruple the length of time. Eg if it takes 8 seconds to sort 5Ø items it will take 32 seconds to sort 1ØØ items. (2 minutes for 2ØØ, and a staggering 2 hours to sort 16ØØ items!).

The quicksort on the other hand takes only double the time to sort double the items (as you would expect), and hence keeps sorting times reasonable.

The same relationship will still apply in machine code, however the program will run about 2Ø times faster. (Note that in BASIC using the capital letter variables A% to Z% in the quicksort it runs 15% faster than by using lower case letters).

[We will offer a prize of £5Ø to the fastest and most useful machine code sort (whether quicksort or not). All entries should be on cassette, and the envelope marked "Sort Routine". The closing date is 1Øth January 1983. Please enclose a suitable SAE if you wish for its return.]

You will find more on sorting in many books. My source for the quicksort was: "Algorithms + Data Structures = Programs" by N. Wirth published by Prentice Hall approx hardback price is £2Ø. It is available from Mine of Information (address in our Discounts pages).

Using the procedures

The Quicksort procedure is listed from lines 1ØØØ to 11ØØ, and the Bubble sort from 3ØØØ to 3Ø8Ø. The test program (lines 1Ø to 34Ø calls both procedures and makes comparative timings for the two methods. In actual usage, of course, only one of the two procedures would be used. In either case the data to be sorted must first be placed in the array called 'file()'. However, you may sort any part of that array by placing the range of elements to be sorted as parameters for the procedure. For example - to sort elements 15 to 35 you would use PROCquicksort(15,35) You can test this in the program supplied by altering the 1 in line 4Ø to some other value.

The sorts given here sort in ascending order, but it is fairly simple to sort in descending order by changing the inequalities in lines 1Ø4Ø,1Ø5Ø,3Ø4Ø.

STRING SORTS

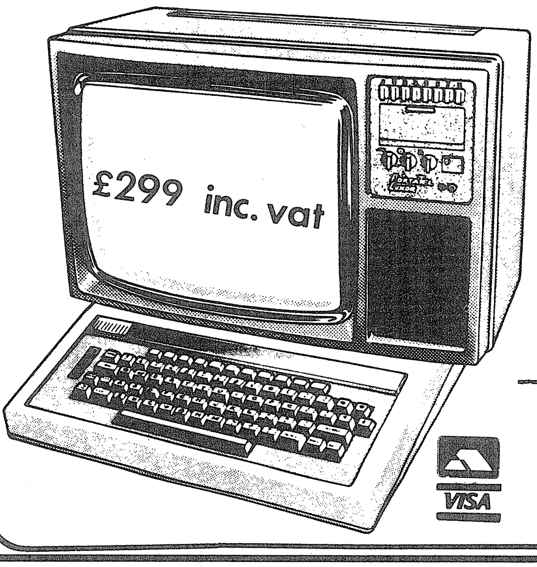
The above sorts can easily be modified to sort strings, just make the following replacements in the procedures:

```
                    anywhere you see  file(  substitute  file$(
                                for    w     substitute    w$
                                for    x     substitute    x$
```

You will, of course have to write a new testing program (ie lines 6Ø-34Ø). S.W.

```
   1ØREM Demonstration of sorts
   2ØREM  by Sheridan Williams
   3ØREM------------------------
   4Øfrom%=1: REM  Sort starting place
   5ØVDU15: REM Turn off page mode
   6ØINPUT''"How many numbers to be so
rted",to%
   65IF to%<=Ø END
   7ØDIM file(to%)
   9ØPRINT
  13Ødummy=RND(-1)
  14ØFOR item%=1 TO to%
  15Øfile(item%)=RND(1ØØ)
  16ØPRINT ;file(item%)" ";
  17ØNEXT item%
  18ØREPEAT
  19ØPRINT'"Bubble or Quick sort (B/Q)
?";
  2ØØsort$=CHR$(GET AND 223):PRINT sor
t$
  21ØUNTIL sort$="B" OR sort$="Q"
  22ØPRINT'"Sorting starts now....."
  23ØTIME=Ø
  24ØIF sort$="B" PROCbubblesort(from%
,to%)
  26ØIF sort$="Q" PROCquicksort(from%,
to%)
  27Øtime=TIME/1ØØ
  28ØPRINT"Sorting completed. It took
";time;" seconds."
  29ØPRINT"Printout (Y/N)?";
  295printout=CHR$(GET AND 223)="Y"
  3ØØIF NOT printout RUN
  3Ø5PRINT
  31ØFOR item%=1 TO to%
  32ØPRINT ;file(item%)" ";
  33ØNEXT item%
  34ØRUN
  999REM-------------------------
 1ØØØDEF PROCquicksort(P%,R%)
 1Ø1ØLOCAL x,w,I%,J%
 1Ø2ØI%=P%:J%=R%:x=file((P%+R%) DIV 2)
 1Ø3ØREPEAT
 1Ø4ØIF file(I%)<x I%=I%+1:GOTO 1Ø4Ø
 1Ø5ØIF x<file(J%) J%=J%-1:GOTO 1Ø5Ø
 1Ø6ØIF I%<=J% w=file(I%):file(I%)=fil
e(J%):file(J%)=w:I%=I%+1:J%=J%-1
 1Ø7ØUNTIL I%>J%
 1Ø8ØIF P%<J% PROCquicksort(P%,J%)
 1Ø9ØIF I%<R% PROCquicksort(I%,R%)
 11ØØENDPROC
 1999REM-------------------------
 3ØØØDEF PROCbubblesort(P%,R%)
 3Ø1ØLOCAL w,swapped,I%
 3Ø12IF R%=P% ENDPROC
 3Ø15REPEAT
 3Ø2Øswapped=FALSE
 3Ø3ØFOR I%=P% TO R%-1
 3Ø4ØIF file(I%)<=file(I%+1) THEN 3Ø6Ø
 3Ø5Øw=file(I%):file(I%)=file(I%+1):fi
le(I%+1)=w:swapped=TRUE
 3Ø6ØNEXT
 3Ø7ØUNTIL NOT swapped
 3Ø8ØENDPROC
```

Timeshare your Colour Monitor with the Family



COLOUR TV

Plus **RGB**

Plus **PAL VIDEO**

BBC Micro lead included

Excellent resolution, geometry

PortaTel LUXOR 14" Colour Monitor
Model No. RGB3711 TV Receiver

PortaTel conversions limited
television and electronic engineers
25, sunbury cross centre, staines road west,
sunbury-on-thames, middlesex. tw16 7bb
telephone: sunbury-on-thames 88972

VISA

DEBUGGING PART 3

(Tracking down subscript errors)

by Rob Pickering

This month Rob Pickering reveals some of the principles involved in tracking down subtle but quite fatal errors in Basic programs. The examples that he investigates centre on "array subscript" error messages, but the techniques used are readily applied to other kinds of error. He begins by explaining what an array is.

AN ARRAY DEFINED

I shall start by explaining briefly what an array subscript is. An array is a collection (or list) of variables with the same name, only a subscript distinguishes between each item in the list. What is a subscript you may well ask? Well it's a number used to distinguish between elements of arrays - we call each item in the array an 'element' of that array. To use examples to explain: Let's assume that we have the array A( ) and that it has eleven elements 0 to 10 inclusive. That gives us the variables:

```
A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8),A(9),A(10)
```

These variables are collectively referred to as elements of the array A( ). The number in the brackets is the subscript. Before you can use an array, you have to tell the computer how many elements are in the array, or rather, what the maximum subscript is. (Above, the number of elements is eleven, but the maximum subscript is 10). The DIMENSION or DIM statement exists primarily for this purpose, you simply enter a line such as: DIM A(10) in order to set up an array with maximum subscript of 10, before you use it. What this does is to allow the computer to set aside memory for these elements. Then, even if you only actually use the first 5 elements, memory is reserved for the remainder.

Once the computer has set aside the memory required for the elements up to 10, you cannot then try to use a subscript larger than 10. That is, the largest subscript you can use is the one given in your DIM statement. But if you do try to use, say, element number 11, the computer will tell you that you can't in the only way it knows how, with an error message: "Subscript at line ...." with an apropriate line number of course. If arrays and anything so far is new to you then I suggest you try the following program, in which an array called fred( ) is defined, and the value of its elements (each zero) printed out. There is a deliberate mistake in the program though.

```
10 DIM fred(12)
20 FOR var=1 TO 13
30   PRINT"value of element ";var;" is.... ";fred(var)
40 NEXT var
```

The array consists of 12 elements, so that when it comes to printing the value of the 13th element it produces a "Subscript At Line 30" error message. The message correctly tells us that it was in line 30 that an attempt was made to reference a non-existent element

SUBSCRIPTS IN 3D NOUGHTS & CROSSES

The (correctly listed) 3D Noughts and Crosses program in the April issue gave similar messages to a good few members. This was obviously caused by typing errors on the part of individual members. But how to find them? The main trouble was undoubtably the error message "Subscript at line 890", or at least that's how the trouble showed itself. The real cause of the error lies elsewhere, and this is what we have to track down.

To begin this process we must first look at the line on which the error message is occurring, line 890 below:

```
890 D=Q(N(M(A,C))):T=T+W(N+D)+P(E(M(A,C) DIV 4))*B(G+C)
```

You can see that the problem is not quite so straigthforward as the one given in the simple example earlier, in fact it is virtually incomprehensible in this state. What we must do is break the line down into managable steps. The line first of all divides nicely into two parts split at the colon. Let's deal with the first part, given below:

```
D = Q( N( M(A,C) ) )
```

But this is still rather too much to put into the tiny human mind for debugging to take place. Let's split it further. The variable D is being assigned a value from one of the elements of the array Q( ). That's as simple as we can get it. But which element is being assigned to it? You may at first think that the subscript is too large for the size of the array Q( ), but in this line there are several arrays, and the subscript could be wrong in any one of them, and the message would be just the same. So, we must now start to expand upon our simplification. So far we have D = Q(subscript). Taking it a step further the subscript is derived from an element of the array N( ). This now makes our expansion as below:

```
D = Q( N(subscript) )
```

and the subscript to the N( ) array is further derived from the two-dimensional array M( ). The element of array M( ) is thus defined by two subscripts. A simplification now is:

```
D = Q( N( M(subscript1,subscript2) ) )
```

all in all a rather complicated line, hardly suprising that so many people had trouble finding errors.

You must first determine whether the error is occurring in this part of the line or the next part. You can do this very easily. The section of the line described is assigning a value to D, so if an error is occurring here, the value cannot have been found. Simply print the value as it would be assigned to D. Use:

```
PRINT Q(N(M(A,C)))    followed by return.
```

If this gives the error message similar to the original- "Array", then obviously the error is occurring in this part of the line. If a value is printed then you know that this part of the line is correct and you can go on to find the error in the other part of the line.

Let us assume that the error DOES occur when you try to print the value being assigned to D. You now know where the error is being detected, and should try to narrow it down further. The first stage in the process is to print the values of each array until you find the one causing the problem. Either you can work inward from the highest to the lowest level, or work outward from the lowest to the highest level. I prefer the latter, since it will probably require the least typing !

The lowest level is the two-dimensional array M( ). Try printing the value of the element being accessed by the variables A and C as follows:

```
PRINT M(A,C)
```

This will either give a value or an error message. If it gives a value then the values of A and C are in range, in which case you should then try printing the value of the next level:

```
PRINT N(M(A,C))
```

.........and so on until you find which subscript is out of range.

Let us further assume that the error HAS occurred when trying to find the value of the N( ) array as above. This, in fact, turned out to be the most frequent cause of error. In any case the values of A and C are very unlikely to be wrong because they are simply the values of the two loops around the statement.

Now that we have found where the error is occurring, (i.e. when an attempt is made to access the element of the N( ) array with a subscript of M(A,C)), we can go back to the previous check using: PRINT M(A,C) to give the number of the element that the machine refused to read. Let us assume that the value printed was 76. (Again, this was the most common error). Now look at the line which defines the dimension of the N( ) array, line 290 in the program listing. Sure enough, the N( ) array is only dimensioned to have 75 elements. Therefore trying to access the 76th element will not be possible, which is why the error message is produced.

This now shows that the error is a wrong value in the array element M(A,C)- but which? Find out with PRINT A,C followed by return. Let us assume that this gives the values A=43 and C=3. Next we must track down the place in the program where this value is being assigned to the array. This is fairly easy in this case because the whole M( ) array is set up in lines 410 to 450 as follows:

```
 410 FOR A%=0 TO 6
 420   FOR B%=0 TO 63
 430     READ M(B%,A%)
 440   NEXT
 450 NEXT
```

All the elements of the M( ) array are read from DATA statements, so the wrong value must be in the data. The data statements for the M( ) array are conveniently arranged to hold sixteen data values on each of the lines 1230 to 1500. Some simple maths will show why this is convenient. The data is read by two loops - one inside the other (nested). The inner loop cycles from 0-63, which is 64 cycles. The outer loop cycles from 0-6, which is 7 cycles. Therefore, if there are 16 data values per line, it will take four lines of data for every 64 values read. The number of lines required=7*(64/16)=28. The point is, that if you know the values of the array subscripts where a wrong value has been read then you can easily find the place in the lines of data from which that number has been read. Assuming that we found A=49 and C=3 when the error was found, this corresponds to the 16th line of data:- line 1380 in the program.

By working out that the error is somewhere in the data still leaves a lot of work checking every line of data. By finding the line of data where the error occurs narrows down the field considerably. The first program I examined for these articles produced the error as described so far, and the cause was easy to spot in that case: the previous line, number 1370 started with the word "DARA" instead of the correct word "DATA". No syntax error is produced because this section of the program is not actually executed, but because the word "DATA" was not there, ALL the data values on the rest of the line were completely ignored! So the value being read was the first value from the following line, number 1390. It was here where the value 76 occurred in the data.

However to take another case, assume that the erroneous value had occurred at a point in a line where the following value is 76, but the one which should have been read was in range. In that case it should be fairly clear that the cause is simply that one data value has been omitted during typing one of the previous lines of data.

Clearly, the opposite holds true as well:- if the previous to the correct one had been read, then an extra value (or comma) has been mistakenly entered previous to that line. This shows that although we may not have found the actual line of data that is wrong, we have at least narrowed down the number of lines to check by a considerable number.

To Summarise:

(1) When an error occurs in a line and you don't know exactly what is causing the error, split the line at a convenient point, and narrow down the field where the error could lie, until you find the exact cause.
(2) Although you may find what is generating the error, you may not have actually found the root cause. In the above, we found that the error was occurring in line 890, but we had to check lines 410-450, then a series of lines of data before finding the true cause.
(3) Arranging data logically in some way related to the way in which it is read can be an enormous advantage when it comes to debugging. (It happens also to be useful when it is necessary to extend or reduce the amount of data.)

In a later issue I shall be looking at the subject of program testing as well as subsequent debugging. This will be aimed towards those people starting to write their own software, and will be pointing out some of the ways to achieve a user-friendly, or foolproof program.

Program tested on 0·1 and 1·1 O.S.

MUSIC WITH MEMORY (16k 32k)

by A. Calcraft

This is a simple way of adding an amusing feature to the music program on the WELCOME cassette, supplied with your machine. The following instructions will add an array to the program, which can be filled with the "notes" as you key them in. Then the data in the array can be sent to the sound routines, to "replay" the notes as originally keyed in. The larger the array, the more notes that can be stored. The variable BUFF is used to allow individual selection of array size depending on your machine size. Lines 600 and 1500 disable the mode 5 screen display, to free extra space, thus allowing a larger array. These lines are essential for 16k machines, but are only needed for 32k machines if you want to save a long tune. (If you don't use them leave the lines as they are in the original program).

```
          Without these additional lines try ..line 5 ..BUFF=3000 on 32k.
          With them in try...................line 5 ..BUFF=6000 on 32k.
          With them in try...................line 5 ..BUFF=2000 on 16k.
```

A further simple expansion to the program would be to add procedures to save and reload the array on tape. This would enable tunes to be kept and reloaded with the program itself. The New User Guide describes how to do this starting on page 394. (Though you will need our cassette bugs fix - see July BEEBUG). When running the program, line 20 will take you to the MENU if ESCAPE is hit at any time. When keying in the program ensure a cold start is used..ie do not run the old music program just before amending it, as this may cause corruption problems.

```
    5 BUFF=2ØØØ
   5Ø AB=Ø :TYPE$="" :DIM NTE$(BUFF)
  6ØØ P%=Ø:PROCPAL
 13ØØ IF TYPE$="REPLAY" N$=NTE$(AB)
ELSE N$=INKEY$(Ø) :IF TYPE$="SAVE"
NTE$(AB)=N$
 15ØØ IF N%>Ø AND N$<>"" PROCNOTE(N%)
 ELSE PROCNOTE(Ø)
 175Ø IF TYPE$<>"" AB=AB+1
 18ØØ UNTIL AB>BUFF-1
 182Ø PROCNOTE(Ø)
 185Ø GOTO 1ØØØØ
1ØØØØ MODE7 :PRINT TAB(5,1Ø) "AS ORI
GINAL PROG........1"
1ØØ1Ø PRINT TAB(5,11)"TO PLAY AND SA
VE MUSIC..2"
1ØØ2Ø PRINT TAB(5,12)"TO REPLAY MUSI
C.........3"
1ØØ3Ø AB=Ø :TYPE$=GET$
1ØØ4Ø IF TYPE$="1" TYPE$="" :GOTO 6ØØ
1ØØ5Ø IF TYPE$="2" TYPE$="SAVE" :
GOTO 6ØØ
1ØØ6Ø IF TYPE$="3" TYPE$="REPLAY" :
GOTO 6ØØ
1ØØ7Ø GOTO 1ØØ3Ø
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

USER DEFINABLE CHARACTERS (OS 1.0) - BUG IN USER GUIDE

Peter Taylor writes:

"Because I have not yet obtained a monchrome monitor I have to strain to see 80 characters across my TV screen. 0 (slashed zero) tends to look like 8, and Y looks like V, so I set about redefining these two characters. I found I could do one or other but not both without rubbish appearing. I queried this with Acorn and soon afterwards they telephoned to say that they had sorted out the problem (This applies to 1.0 OS ONLY).

On page 427 of the User Guide the description of the *FX 20 command is incomplete. The second argument should be 1 to 6 depending on how much memory is set aside, which in turn depends on the highest character code to be re-defined. Thus to redefine Y, code &59, PAGE must be set to OSHWM+&500, and the appropriate command is *FX 20,5. If memory is not short it is safest to command *FX 20,6 and set PAGE to OSHWM+&600; then any character can be re-defined without worry. On a normal system OSHWM is usually &0E00 and on a disc system &1900, but use OSBYTE &83 to check.

For O.S.0.1 Only

TRANSPARENT LOADER

FOR USER KEYS, NEW CHARACTER AND BUGS FIX

by Gwyneth Pettit

Gwyneth Pettit describes a useful method of loading the cassette bugs fix, user defined keys, and user defined characters all from one tape, which can be loaded without disturbing Basic programs already in the machine.

The cassette bug in OS Ø.1, reported in the July BEEBUG Newsletter, can be remedied without using a BASIC program. This method of patching is exceptionally useful in cases where you have already written a valuable BASIC program and realised, too late, that you haven't loaded the cassette patch yet!

You will, of course, need a BASIC program for the patch in the first instance and this is listed below. Type the program in (you can run it to place the patch before saving the machine code it is going to produce for you). Insert your own versions of the user-defined function keys, if you don't like the look of the accompanying ones. Insert also any soft character definitions you want, either instead of, or in addition to, the ones given here (which are the four playing card symbols, for those who find BBC graphics short on 'games' facilities).

Now run the program and it will load the function keys, the soft characters and the machine code patch in the memory area from &ØBØØ to just above &ØDØØ. Type

```
*SAVE "PATCH" ØBØØ ØD42 ØDØØ
```

and it will save perfectly because the patch is in place. When you want to reload the patch from cassette, either before starting your next programming session or at any time during it, place your cassette in the recorder and type

```
*RUN "PATCH"
```

Because you are loading directly into workspace memory and not affecting the BASIC area, any program you may have written will be unaffected. You can also use this method to reload your function keys if another program has reset them, or to preload your own favourite soft characters for a graphics program.

Note that key 1Ø (the BREAK key) is set up to replace the pointers in the OS workspace - if for any reason key 1Ø is reprogrammed by your BASIC, it would be safest to repeat the PATCH run in case you need to BREAK your next program. NB If you run PATCH before running any BASIC, remember that you effectively have no program in memory and if you press BREAK at this stage, you will crash the BASIC interpreter. There is no way round this problem except to switch the machine off and on again, run PATCH and DON'T PRESS BREAK.

Note also that all the keys to be reprogrammed are first cleared in lines 5Ø to 1ØØ. This seems tedious but is necessary in case the keys already hold long strings - you may get the message BAD KEY in that case. Key 4 has been loaded with the best method of verifying saved programs, as published in September BEEBUG Newsletter (page 5).

```
   1ØREM PATCH TO FIX CASSETTE BUGS
   2ØREM LOADING USER AND SOFT KEYS TOO
   3ØREM by Gwyneth Pettit, based on
   4ØREM R.T.Russell, July BEEBUG
   5Ø*KEYØ
   6Ø*KEY1
   7Ø*KEY2
   8Ø*KEY3
   9Ø*KEY4
  1ØØ*KEY1Ø
  11Ø*KEYØ|ORUN|M
  12Ø*KEY1LISTOØ|MW.Ø|M|NL.|M|O|M
  13Ø*KEY2LISTO7|MW.7Ø|M|O|BL.|M|CW.Ø|ML
ISTOØ|M
  14Ø*KEY3DIMP%-1:P.HIMEM-P%|M
  15Ø*KEY4*LOAD""8ØØØ|M
  16Ø VDU 23,224,&6C,&FE,&FE,&FE,&7C,&38
,&1Ø,Ø:REM heart
  17Ø VDU 23,225,&38,&38,&1Ø,&D6,&FE,&D6
,&1Ø,&1Ø:REM club
  18Ø VDU 23,226,&1Ø,&38,&7C,&FE,&FE,&FE
,&54,&1Ø:REM spade
```

```
  19Ø VDU 23,227,&1Ø,&38,&7C,&FE,&7C,&38
,&1Ø,Ø:REM diamond
  2ØØFORpass=ØTO1:P%=&DØØ:PROCpatch:NEXT
  21Ø CALL &ØDØØ
  22Ø*KEY1Ø?&218=&13:?&219=&D:?&2ØA=&19:
?&2ØB=&D|MO.|M
  23Ø PRINT "Save by *SAVE ""PATCH"" ØBØ
Ø ØD42 ØDØØ"
  24ØEND
  25ØDEFPROCpatch
  26Ø[OPTpass*2
  27ØLDA #&13:STA &218
28ØLDA #&D:STA &219:STA &2ØB
29ØLDA #&19:STA &2ØA
3ØØRTS
31Ø.FIX1 PHA:JSR &F521:PLA:RTS
32Ø.FIX2 CMP #&91:BNE GO:CPX #Ø:BNE GO
33ØTSX:LDA &1Ø2,X:CMP #&F7:BEQ TRAP
34ØLDX #Ø:.TX LDA #&91:STA &FEØ9:RTS
35Ø.GO JMP (&DB6Ø)
36Ø.TRAP PLA:PLA
37Ø JSR &F9D8:JSR &FB7B
38ØJSR TX:JMP &F7FB
39Ø]ENDPROC
```

The program below is Brian Carroll's combined keyset and bugs fix program. You may wish to incorporate some of his key functions in Gwyneth's invisible loader.

```
  1ØØREM** FILENAME: KEYSET (Improved Ve
rsion) ** B. Carroll. (C)1982 ***
  11ØMODE7:PROCassemble(&DDØ)
  12Ø?&218=fix1:?&219=fix1 DIV 256:?&2ØA
=fix2:?&2ØB=fix2 DIV 256
  13ØVDU28,12,5,33,Ø
  14ØPRINTCHR$134" ";:REPEAT PRINT"*";:U
NTIL COUNT=12:PRINTCHR$156
  15ØFORZ%=ØTO1:PRINTCHR$141CHR$134"*"CH
R$129"KEYSET"CHR$134"*":NEXT
  16ØPRINTCHR$134" ";:REPEAT PRINT"*";:U
NTIL COUNT=12:PRINTCHR$156
  17Ø*K.Ø RUN|M
  18Ø*K.1 AUTO
  19Ø*K.2 |L|M
  2ØØ*K.3 *TVØ,1|MVDU19,128,2,Ø,Ø,Ø,19,1
,Ø,Ø,Ø,Ø|V3|L|M
  21Ø*K.4 LISTO7|M|BLIST|M|CLISTOØ|M
  22Ø*K.5 L.|M
  23Ø*K.6 *C.|M
  24Ø *K.7 MO.7|M
  25Ø*K.8 |N|M
  26Ø*K.9 |L:@%=Ø:DIMP%-1:P.H.-P%" bytes
 free, starting at &"~P%"."'|L|M
  27Ø*K.1Ø ?&218=&DØ:?&219=&D:?&2ØA=&D6:
?&2ØB=&D|MP.">BREAK"|L|MOLD|M
  28ØVDU28,1,24,36,6
  29ØPRINT CHR$131"The User_Defined Keys
 Are Now Set"
  3ØØPRINT CHR$131"As Follows, & COS Bug
s Are Fixed:"''
  31ØVDU28,3,24,36,1Ø
  32ØPRINT" Ø - RUN"'" 1 - AUTO (Add lin
e No; RETURN)"'" 2 - CLS"
  33ØPRINT" 3 - MODE 3 (Black on green)"
'" 4 - Print listing"'" 5 - LIST"
  34ØPRINT" 6 - Catalogue Tape (*CAT)"'"
 7 - MODE 7 (White on black)"
  35ØPRINT" 8 - Paged Mode"'" 9 - Memory
 free"
  36ØPRINT"1Ø - BREAK; Set COS patch; OL
D"''
  37ØVDU28,1,24,39,22
  38ØPRINT'CHR$131"Type NEW & Press KEY
2 To Clear."
  39ØVDU26:END
  4ØØREM** Fixes for COS bugs - R.T.Russ
ell. (Ø.1 OS only) **
  41ØDEFPROCassemble(address%)
  42ØFOR pass=Ø TO 1:P%=address%
  43Ø[OPT pass*2
  44Ø.fix1 PHA:JSR&F521:PLA:RTS
  45Ø.fix2 CMP#&91:BNEgo:CPX#Ø
  46ØBNEgo:TSX:LDA&1Ø2,X:CMP#&F7
  47ØBEQtrap:LDX#Ø
  48Ø.tx LDA#&91:STA&FEØ9:RTS
  49Ø.go JMP(&DB6Ø)
  5ØØ.trap PLA:PLA:JSR&F9D8:JSR&FB7B
  51ØJSRtx:JMP&F7FB:]
  52ØNEXT
  53ØENDPROC
  54ØREM*** 1516 Bytes ***
```

Happy Memories

4816 100ns memory upgrade IC's

1 off	25-99	100 up
2.45	2.10	1.95

1 off price less 10% for Beebug members

Please add VAT. No Postage on sets of 8

Happy Memories, Gladestry, Kington,
Herefordshire. HR5 3NY

Tel: (054 422) 618/628

Access and Visa taken over the 'phone.

SERIAL PRINTER PORT (RS423) AND RGB UPGRADE

by Rob Pickering

A standard model 'A' has no printer interface of any kind. April BEEBUG included instructions for fitting a parallel interface, we now give details of how to fit the serial interface plus the RGB output, both are part of the same upgrade kit. The serial interface is called an "RS423" and is compatible with the "RS232" fitted to most of the popular printers currently on the market. The serial interface is also necessary for sending data along telephone lines.
The RGB is a colour signal output that is capable of producing a far more stable colour picture than the UHF output - though you do need a colour set that has an RGB input.

IDENTIFYING THE COMPONENTS

The serial upgrade kit consists of two Integrated Circuits and two sockets. These are labelled on the circuit board as I.C.74, I.C.75, SK3, and SK4. You should identify which component is referred to by which label. The socket for the serial interface (SK3) is known as a 'Domino-socket' because it accepts pins in a layout the same as the five on a domino. The other socket is therefore the RGB socket (SK4), this too has five pins but in a completely different layout. The I.C.s are as follows:

I.C.74 marked as: DS88LS120N   I.C.75 marked as: DS3691N

Below we give instructions in easy steps showing how to remove everything, but also telling you how to put it back together. Before you start though, I'll offer a strong warning: ACORN have implied, if not stated outright, that 'user-upgrades' will terminate the guarantee, so beware of this fact! In addition, it is necessary to solder the two sockets on to the main printed circuit board. Any soldering on the 'motherboard' as it is known, is a serious matter requiring a fine-tipped soldering iron, some fine solder AND some precision soldering experience. Please do not attempt this upgrade unless you are an experienced solderer.

TAKING IT APART

(1) Unplug the computer from the mains power supply.
(2) Remove the lid from the computer by unscrewing the four screws marked 'Fix' and located two on the back and two underneath, then lift the lid carefully away.
(3) Removing the keyboard comes next: locate and remove the bolt at each end of the keyboard. Some machines will have three fixing bolts holding the keyboard, so if it isn't loose after unscrewing two bolts, find the other and unscrew that, putting the bolts somewhere safe.
(4) On the rear of the keyboard toward the left there is a wide cable connector (normally grey), lift this slowly away vertically, being careful not to bend any of the pins.
(5) Starting at the left of the keyboard on the loudspeaker, trace the wires holding the loudspeaker to the main circuit board under the keyboard. Pull the plug firmly away from the main board.
(6) The keyboard should now be completely free. Lift it away and place it somewhere safe away from any source of static electricity.

FITTING THE ICs

(7) Looking at the rear of the motherboard in front of the two holes awaiting the RS423 and RGB sockets you will see two empty I.C. sockets lying in an East-West orientation (different to the North-South convention on the rest of the board). The sockets are labelled "I.C.74" and "I.C.75". You must push each I.C. into the CORRECT socket. To do this you should locate one row of pins into the socket without actually putting them in, then gently push the second row of pins into the socket, pushing the whole I.C. firmly home as the pins become located. Be very careful not to bend any pins out of place:- don't push the whole I.C. in until ALL the pins are located. Avoid touching the pins themselves.

REMOVING THE MOTHERBOARD

(8) Now that the I.C.s are in place, you can proceed to fit the sockets. The sockets have to be soldered from the underside of the board so you must first remove the

motherboard. Start by locating and removing the securing screws, one in each corner of the motherboard. Be careful not to lose the screws or the washers. Don't lift the board yet !

(9) The BNC connector located on the back panel (which gives the combined-video feed) is attached to the motherboard by two push-on plugs [soldered in some cases -Ed]. The wires on these are too short to allow the motherboard to be lifted, so you must remove these two plugs. The plug toward the left has a black wire attatched and the one toward the right has a white wire. Pull the plugs firmly, they may be quite tight, but DON'T pull the wires - only the plugs themselves.

(10) Leaving the power leads connected to the left of the board, you may now lift the motherboard out from the computer. To remove the board, push it fully to the back of the case, lift the rear edge of the motherboard, raise the board tilting it on to the left hand edge, finally twist it back to front so that it is upside down. If you didn't follow that... the idea is to turn the motherboard upside down. Take this opportunity to blow all the dust out from your computer.

FITTING THE SOCKETS

(11) Insert SK3 (The domino socket) into the left set of holes (this will be on the left when the board is the correct way up). Note the flange on the edge of the existing socket which complements the flange on the domino socket. Insert it by sliding it directly from above such that the flanges are together. It should just drop into place. If it does not then the pins must be out of alignment - bend each until they match up with the holes in the board, and slot them into place. Solder the socket on to the board making sure that the socket is pressed firmly against the board, and being careful not to apply too much solder so that it runs. Avoid holding the soldering iron on the contacts for any longer than necessary, since excess heat will damage the surrounding components.

(12) Fit the RGB socket (SK4) into the other set of adjacent holes and solder it in a similar manner.

REASSEMBLY

(13) Now that all the components are in place you can reassemble everything. Start by twisting the motherboard back so that it is once again the correct way up- that is with the components uppermost.

(14) Lower the front edge of the motherboard back into the case, sliding the motherboard right forward until the rear edge can fit into the case, then slide it back a little so that the fixing holes in the corners are aligned.

(15) Fit the connectors for the combined output (i.e. from the metal BNC connector) back on to the board, remembering that the plug with the white lead fits onto the board slightly to the right of the black.

(16) Replace the four screws securing the corners of the motherboard, making sure that each is accompanied by its washer. Do not over tighten them because they only fasten into the plastic case, and the thread would easily strip if forced.

(17) Rest the keyboard in place and fit the loudspeaker plug back onto the motherboard on the two pins from which you removed it in step (5), again be careful not to bend the pins be pressing at an angle or using too much force.

(18) Replace the broad 'ribbon-cable' lead on to the row of pins situated on the rear of the keyboard.

(19) Replace the two (or three) bolts which secure the keyboard to the case, making sure that they are tight enough not to work loose.

(20) Replace the four screws remaining into the holes marked 'Fix' on the case. The case should now be completely fastened together with no pieces remaining, if this is not the case... you did something wrong...!

See the User Guide page 499 for connections to the RGB socket. The User Guide also gives the connections for the serial interface at the computer end, but you will have to refer to your printer manual for the necessary connections particular to your printer. If it is possible to set the receive rate on your printer, set it to the maximum so that it will run as quickly as possible.

You will have to set the computer's transmission rate by using the command

*FX8,n where 'n' is the code for the baud-rate appropriate to your printer. The codes for the baud-rates are given on page 424 of the User Guide. You will also need to execute *FX5,2 to set the computer to serial output. To test the printer, simply enter a VDU2 command or press CTRL-B, and all subsequent output will go to both the screen and the printer.

SUPPLIERS

A number of suppliers stock the parts for this upgrade. See the advertisements in this magazine, but telephone them to check availability.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

"ELSE" WARNING

A. Williams points out that nesting of IF THEN ELSE statements can give unexpected results. Consider:

```
IF TRUE=FALSE THEN
   IF TRUE=TRUE THEN PRINT"1"
   ELSE PRINT"2"
ELSE PRINT"3"
```

You would expect the result to be "3" - but the BBC Micro prints "2". The reason for this is that BASIC searches the line for an ELSE token or return character, ignoring any intermediate IF's. This simple approach is inadequate for nested IF statements. The first ELSE is executed, regardless of nesting level. For those interested the locations in the BASIC ROM to look at are &98C2 to &98D5.

PAGE CHANGE FOR DISC SYSTEMS

When using a disc system for cassette purposes, type *TAPE. Also type PAGE=&E00 to get the maximum amount of free memory before loading a new program.

Turn to Computer Plus

BBC Referral Centre

- * MODELS "A", "B" in stock
- * UPGRADE KITS
- * COLOUR MONITORS, PRINTERS, ETC.
- * CASSETTE DECKS & LEADS.
- * SOFTWARE *from* ACORNSOFT/BUGBYTE/COMPUTER CONCEPTS/ PROGRAM POWER/RABBIT/POM

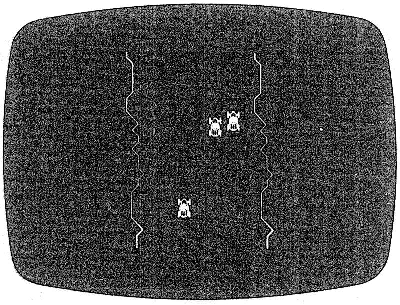
47, QUEENS ROAD, WATFORD. Telephone: WATFORD 33927

Program tested on 0·1 and 1·1 O.S.

RACER (16k)

by Simon Wilkinson

This is an very economically written program, the graphics are good, it is fairly fast moving, and is good fun to play. It must rate 5 stars especially as it only needs 16k; though it will not work across the Tube.



The object of the game is to guide a racing car along a winding track, overtaking cars in the same race, but avoiding the odd wheels that have come off the other cars, at the same time avoiding the crash barriers. To steer the car use "Z" for left, and "X" for right.

The visual effect produced by the program is very good, and the other cars, obstacles, and race track continually scroll downwards giving the impression that the racing car is moving upwards. The game gradually gets more difficult, with the track getting narrower, and your racing car creeping up the visual field (giving less notice of on-coming obstacles).

The sound effects are good too, especially if you use a larger speaker. The engine note changes as you go faster, and when you hit an object the explosion will literally blow you out of your seat - the car disintegrates nicely too. Incidently, you can make the car crash by pressing 'escape' just at the vital moment.

As it stands the game just squeezes into a 16k machine (140 bytes to spare!). To achieve this in mode 4, the program is less than 2k long, and almost all movement on the screen is achieved by downward scrolling (if you want to do this manually, you move the cursor to the top line, and execute VDU 11:VDU 11, or try the following program:

```
10 CLS:FOR X=1 TO 1000
20 PRINT TAB(0,0);X;:VDU11
30 NEXT X
```

which will print numbers scrolling down the screen.

Here is a breakdown of the variables used:

A,B,C,C%,D,E,F,G,H:	Used in initial character definition.
NL$	Newline (Cursor left 3 times + cursor down)
NC$	No car - to erase the car completely before the screen is scrolled
CF$	Car steered forwards (17 characters long)
CL$	Car steered left (17 characters long)
CR$	Car steered right (17 characters long)
W%	Width of the road (Decremented from 20 to 6)
h%	Necessary because each time the screen is scrolled the screen mapping changes.
X%,Y%	Tabulated position of the first character of the car
X	Tabulated position of the first character of the road
x%,y%	Old X% and Y% for moving the car
Y	Loop variable
Q%,i$,V	Dummy variables
TI	Time at which the road was last made narrower
TIME	Gives final score. (Saved in TI while crashing.)
D%	Deviation of new X% from old X%
MR$	Next piece of road as returned by procedure PROCMR ('more road' procedure)
R%	Random number: 1 or 2, if 1 then road will go straight, otherwise a turn left or right.
C,c	X-coordinates of bits of car

```
   10REM "Simon Wilkinson's Racer "
   20ENVELOPE1,130,80,-4,-2,2,40,50,1,
-1,1,1,1,0
   30ENVELOPE2,2,0,0,0,0,0,0,60,-1,-10
,1,126,60
   40ENVELOPE3,2,4,0,-4,2,1,2,35,0,0,1
,35,34
   50MODE4:FORC%=224TO240:READ A,B,C,D
,E,F,G,H:VDU23,C%,A,B,C,D,E,F,G,H:NEXT
   60DATA12,12,12,15,15,12,12,12,60,36
,66,129,129,129,153,165,48,48,48,240,2
40,48,48,48,1,1,2,2,2,2,2,1,60,102,90,
189,189,153,66,60,128,128,64,64,64,64,
64,128,13,13,13,15,15,13,12,12,0,36,0,
36,0,36,139,126,176,176,176,240,240,17
6,48,48
   70DATA24,24,28,15,15,14,6,6,96,96,1
12,240,248,56,24,24,6,6,14,15,15,28,24
,24,24,24,56,240,240,112,96,96,128,128
,128,128,128,128,128,128,1,2,4,8,16,32
,64,128,128,64,32,16,8,4,2,1,60,126,25
5,231,231,255,126,60
   80VDU23;11,0;0;0;0:TIME=0:ONERRORPR
OCCRASH:RUN
   90NL$=STRING$(3,CHR$8)+CHR$10:NC$=S
TRING$(3,"   "+NL$)
  100CF$=CHR$224+CHR$225+CHR$226+NL$+C
HR$227+CHR$228+CHR$229+NL$+CHR$230+CHR
$231+CHR$232
  110CL$=CHR$233+CHR$225+CHR$234+RIGHT
$(CF$,14)
  120CR$=CHR$235+CHR$225+CHR$236+RIGHT
$(CF$,14)
  130W%=20:h%=0:CLS:X%=16:Y%=25:X=9:x%
=X%:y%=Y%
  140FORY=0TO28:PRINTTAB(X-1);CHR$237;
SPC(W%);CHR$237:NEXT
  150PRINT"""Z""to go left and ""X"" t
o go right."
  160PRINTTAB(X%,Y%+1);CF$
  170INPUTTAB(5,31)"Press ""RETURN"" t
o start "i$:V=RND(-TIME):SOUND&13,3,1,
255:TIME=0:TI=TIME
  180PROCMR
  190IFINKEY(-98)CAR$=CL$:D%=-1:SOUND&
12,-6,40-X%,3:GOTO220
  200IFINKEY(-67)CAR$=CR$:D%=1:SOUND&1
2,-7,X%,3:GOTO220
  210CAR$=CF$:D%=0
  220IF?((X%+1+40*(Y%-h%))*8+HIMEM)<>0
PROCCRASH:RUN
  230IFY%=1THEN250
  240IFTIME-TI>20000/Y%THENY%=Y%-1:SOU
ND&13,3,(25-Y%)*4,255:TI=TIME
  250X%=X%+D%-(X%<2)+(X%>37)
  260PRINTTAB(x%,y%)NC$:PRINTTAB(0,0)C
HR$11TAB(X)MR$TAB(X%,Y%)CAR$:x%=X%:y%=
Y%
  270IFRND(1)<0.01 W%=W%-1-(W%<7):PRIN
TTAB(RND(39),0)CHR$240
  280h%=h%+1:IFh%>=Y%+1 h%=Y%-31
  290IFRND(1)<.97THEN310
  300PRINTTAB(X+RND(W%)-3,0)CF$
  310GOTO180
  320DEFPROCMR:R%=RND(2)
  330IFR%=1 MR$=CHR$237+STRING$(W%," "
)+CHR$237:ENDPROC
  340IFRND(40-W%)>X MR$=CHR$8+CHR$238+
STRING$(W%," ")+CHR$238:X=X+1:ENDPROC
  350IFX=1THENR%=1:GOTO330
  360X=X-1:MR$=CHR$239+STRING$(W%," ")
+CHR$239:ENDPROC
  370DEFPROCCRASH
  380SOUND&13,0,10,255:SOUND&110,2,7,2
55:SOUND&111,1,RND(20),255:TI=TIME:c=X
%:PRINTTAB(X%,Y%)"   ":C=X%-1
  390REPEATC=C+1:PRINTTAB(c,Y%-C+X%)CH
R$224TAB(C,Y%-C+X%)CHR$225TAB(C+1,Y%+C
-X%)CHR$226
  400FORT=1TO50:NEXT
  410PRINTTAB(c-2,Y%-C+X%)SPC4TAB(C-2,
Y%-C+X%)SPC3TAB(C-1,Y%+C-X%)SPC3:c=c-1
-(c<1)
  420UNTILC=39 ORY%-C+X%<1 ORY%+C-X%>3
7:*FX15,0
  430*FX12,0
  440I=INKEY(200)
  450VDU22,7:FORI%=15TO16:PRINTTAB(5,I
%);CHR$141;"YOUR SCORE - ";TI:NEXT
  460PRINT"Do you wish to play again?"
;:IFGET$="Y"ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SOUND EQUIVALENCE

A.J.Vincent has found a function to convert the frequency of a note into the required value for the SOUND command. The function is:

```
DEF FNfreq(F)=INT(((LOG(F)-LOG(33))*48/LOG(2)-42)+.5)
```

You can use this as follows:

```
10 INPUT'"What frequency",frequency
20 PRINT"Use ";FNfreq(frequency);" in the SOUND command."
```

TV/MONITOR REVIEW UPDATE

by David Graham

In the June issue we reviewed a number of TV sets and monitors suitable for use with the BBC machine. We appended to that a 'stop press' notice of a Grundig colour monitor available from Kingsley TV Services. We have since been testing one of these machines over a period of months, and present here a brief review.

Supplier: Kingsley TV Services, 40 Shields Road, Newcastle upon Tyne (Tel:0632 650653)

Model: 16" Grundig colour TV/monitor with RGB (BBC pin compatible) and video input and output, with full remote control.

Price: £263 + VAT (£10 discount before VAT to BEEBUG members).

This set is a Grundig TV that Kingsley modify themselves. It has full remote control, and for an extra £75 can be fitted with an internal Teletext adaptor (also remotely controlled). The picture quality when working as an ordinary TV is good, and certainly comparable to the SONY Trinitron in clarity. A switch at the back of the set changes it to an RGB monitor, and the stability and picture quality are also good in this mode. It even gives a quite creditable display in the 80 column mode - though not as good as is possible with some black and white monitors. Additionally there is an audio input socket which some people have wired up to the Beeb we are told.

The review machine has been in constant use at BEEBUG for three months now, and we are very pleased with it. There are really only two negative features to report. There is a video pick-up buzz on the audio of our set in monitor mode, but the audio may be switched off, and in any case Kingsley reports that they now incorporate as standard a modification to remove this. The second is that the set is on the bulky side - but it is not more bulky than comparable 16" sets. If you want a more compact unit then you could try their 14" version.

There is in fact a range of models from 14" without remote control at £212+VAT to 20" or 26" models. They also stock a Sanyo 12" black and white monitor at £90 + VAT. They offer £10 discount before VAT on all models to BEEBUG members.

MIDWICH COMPUTER COMPANY LIMITED

FAST EX-STOCK DELIVERY OF MICROCOMPUTER COMPONENTS AT UNBEATABLE PRICES

MIDWICH NOW APPOINTED OFFICIAL ACORN BBC MICRO DEALER

BBC MICRO COMPONENTS

2114 Low Power 200ns	0.80
4516/4816 100ns	2.69
6522	2.99
74LS244	0.59
74LS245	0.69
81LS97	0.90
DP8304	4.50
DS3691N	4.50
DS88LS120N	4.50
20 way Right Angle IDC Header	2.46
26 way Right Angle IDC Header	3.24
34 way Right Angle IDC Header	3.80
15 way Right Angle D. SKT	3.50

BBC Microcomputer Upgrades

*Memory Upgrade (BBC1)	21.50
*Printer & User I/O kit (BBC2)	7.60
*F Disc interfce inc DOS (BBC3)	70.00
*Analogue input kit (BBC4)	7.60
*Serial I/O & RGB kit (BBC5)	10.25
*Expansion bus & tube kit (BBC6)	5.95

*Printer cable inc Amphenol plug (not assembled) (BBC21)	13.00
*User port connector & cable (BBC22)	2.00
*Analogue input plug & cover (BBC44)	2.25
*5 pin DIN plug for serialint (BBC111)	0.60
*6 pin DIN plug for RGB int (BBC109)	0.60
*7 pin DIN plug for cassette (BBC141)	0.60
*Connector for bus & cable (BBC66)	3.50
*Single disc drive (100K) (BBC31)	225.00*
*Dual disc drives (BBC32)	345.00*
Teletext Receiver (200K) (BBC71)	144.34**
Prestel receiver (BBC72)	90.00**
*Games Paddles (per pair) (BBC45)	11.30

*Prices on these items are likely to change please contact sales office before ordering.
As some items are on extended delivery from Acorn please check availability before ordering items marked **.

ACORN FOR THE BBC MICROCOMPUTER

Graphs & charts on the BBC Micro (AC122)	7.50
Graphs & Charts Cassette (AC123)	7.50
Algebraic manipulation pk (AC124)	8.65
Forth on BBC Micro (AC125)	7.65
Forth pack (AC126)	14.65
Lisp on BBC Micro (AC127)	7.50
Lisp pack (AC128)	14.65
Games-Philosophers quest (AC129)	8.65
Games-Defender (AC130)	8.65
Games-Monster (AC131)	8.65
Games-Snapper (AC132)	8.65

EPROM PROGRAMMER FOR BBC MICROCOMPUTER

- ☆ Programs 2516, 2716, 2532,2732 Industry Standard EPROMS
- ☆ No external power supply required
- ☆ Plugs straight into expansion socket
- ☆ Easy to use
- ☆ Includes all Software required

EPROM Programmer (Kit)	49.95
EPROM PROGRAMMER (Assembled)	57.95

VISA

24 Hour Telephone order service for credit card holders. All prices exclude VAT and carriage (0.75 on orders under £10 nett). Official orders from educational and government establishments, and public companies accepted. Credit accounts available to others (subject to status). All orders despatched on day of receipt. Out of stock items will follow on automatically at our discretion or a refund will be given if requested. NO SURCHARGE FOR CREDIT CARD ORDERS.

Access

MIDWICH COMPUTER CO LTD

Rickinghall House, Rickinghall, Suffolk IP22 1HH Telephone (0379) DISS 898751

Program tested on 0·1 and 1·1 O.S.

MINI TEXT EDITOR VERSION II (16k/32k)

by David Graham

In response to numerous requests, here is the full text and program for the MkII Mini Text Editor. The main enhancement over that published in the June issue, is the inclusion of a numberless printout routine.

This program requires 32k. But some minor modifications would enable it to work in mode 7 in a 16k machine. In fact just change mode 3 in lines 25Ø9Ø and 31Ø1Ø to mode 7 to get it working.

This is a minute wordprocessor substitute. It uses the Beeb's own Basic text editing facilities to create and edit files of text. These are entered as lines of Basic using the AUTO command, and may be easily saved, loaded, edited and printed out. The program does not allow right-justification or closing up of text, so inserting words could cause a line to overflow - in which case the neatest thing to do may be to retype the paragraph.

If you have been using the MkI version, you will find that lines 95Ø to 27Ø5Ø remain essentially the same. Be sure to enter lines 1 to 1Ø exactly as they appear: this is important for the correct functioning of the numberless printout routine. Lines 1Ø to 99Ø each contain a few 'space' characters.

The function keys are used extensively in this editor, and a key strip plan, which may be photocopied, accompanies this article. It should be slid under the perspex strip above the keyboard. Type CHAIN"" to load the program, followed by key f1 or f2. Either key will initiate the AUTO mode for text entry, but f2 will give your address as a letter head (you must replace BEEBUG's address with your own here - alter lines 1Ø1Ø,1Ø2Ø etc). Alternatively, pressing f1 will erase the address from the machine.

Then simply type in the text that you require, using 'return' when you reach the end of a line. If you go over the end of a line, use the 'delete' key to reduce the line length. As the keys stand, f6 will tab along 6 spaces, and f8 will tab across to the right side for entering addresses. f9 should be used if you wish to insert a blank line (the 'return' key will not achieve this). f9 leaves the cursor indented 6 spaces to make a paragraph indentation. Key f3 is unprogrammed, and may be programmed with a word or short sentence that you use often.

When you have finished entering text, press the 'escape' key, and then use the LIST command, and editing keys. Pressing f7 will return mode 7 for better visibility. You can get back into mode 3 with fØ, which just executes RUN. To return to entering text after you have escaped from the line numbering mode, you must type AUTO x where x is greater than the number of the last Basic line that holds previously entered text. For this reason it is easier to correct mistakes at the end rather than as you go along.

To save the text created, use the SAVE"name" command, and save and load as a normal program. This means that each text file saved, conveniently has a copy of the editor with it.

Two modes of printout are available. f5 gives a printout with Basic line numbers, for ease of subsequent editing; while f4 calls a routine which ignores the line numbers. A number of possible options are available with this form of printout, and those chosen will depend on the paper feed mechanism on your printer, and the number of lines per page required. As the program stands it will send a form feed character (check your printer manual to see whether your printer responds to this) every 31 lines of text. This may be useful if you are printing double spaced text (achieved by typing *FX 6,0 - assuming that this command has not already been used). If you require a different number of lines printed before a form feed, then alter line 31052 of the program, changing the value 31 to the number required. If you do not want a form feed at all, then erase line 31052 completely. If you want the program to stop after say 50 lines of text have been printed so that you can insert another piece of paper, alter line 31052 to read *IF L%>50 THEN wait=GET:L%=0. Pressing any key will cause another page* to be printed. To increase the spacing between lines, insert 31044 IF ?K%=13 VDU 1,10 This may be extended to VDU 1,10,1,10 or VDU 1,10,1,10,1,10 etc to increase the spacing further. Each "1,10" in the VDU command adds an extra line feed after each line of text. Incidentally when you select printout without numbers using key f4, you are asked to enter a continuation line number. This is just to allow you to start printout at any line of a page. Thus as things are presently configured, if you enter 29 here, the program will print out 2 lines of text before doing its first form feed. For the benefit of printers using a continuous roll of paper (and for spacing down the letter head), printout is prefaced with a number of blank lines. These may be removed by deleting lines 950 to 990. At the end of the printout the screen will display a word count (inflated by the effect of double spaces) and a lines in last page count.

If you are using the serial interface (RS423) for your printer, you will need to customise your program in the following way. First of all change line 2 to read serial=TRUE. This sets the program for serial output at 1200 baud. If your printer requires a different baud rate, then replace line 5 with the appropriate command:

*FX 8,1	75 baud	*FX 8,5	2400 baud
*FX 8,2	150 baud	*FX 8,6	4800 baud
*FX 8,3	300 baud	*FX 8,7	9600 baud
*FX 8,4	1200 baud	*FX 8,8	19200 baud

All printers behave in different ways, and word processors costing £50 upwards are (or can easily) be customised for a given printer. This is obviously not possible in this case. Although we have tried to provide options for a number of different printer configurations, we cannot cover them all, and would urge you to experiment with the program to achieve the results that you require. Note however, if you are modifying the program, leave lines 1-10 exactly intact in order to retain the correct functioning of the printout routine.

Because of the way in which text is stored, it is necessary to avoid Basic reserved words in capitals, and the abbreviations for them. Thus P.O.Box would be printed as PRINTOLDBOX on the printout, and as something indecipherable on the numberless printout (since Basic words are stored as tokens).
Note that to keep the program compatible with earlier versions, we decided not to renumber it before publication.

IMPROVED MINI-TEXT EDITOR

A non-member! wrote saying that you can improve the mini-text editor that we gave in BEEBUG no 3, by starting each line with an asterisk. The BASIC tokenising routine then considers this to be a MOS command line and switches off the tokenising routine. Thus you can quite happily enter 'P.O.BOX' without fear of it being changed.
He suggests setting up *KEY1 |M* and using f1 instead of 'RETURN' when entering text.
A minor modification would allow the 'Printout without numbers' routine to ignore the asterisk.

»

RUN	AUTO no address	AUTO with address	PRINT no numbers	PRINT with numbers	TAB 6	MODE 7	TAB RIGHT	RETURN & INDENT

```
    1REM MINI TEXT EDITOR
    2serial=FALSE
    3 IF NOT serial THEN 9
    4 *FX 5,2
    5 *FX 8,4
    9GOTO25ØØØ
   1Ø
  95Ø
  96Ø
  97Ø
  98Ø
  99Ø
 1Ø1Ø
 1Ø2Ø
 1Ø3Ø
 1Ø4Ø
```

BEEBUG,
P O Box 5Ø,
St Albans,
Herts.

```
25ØØØREM KEY SET ROUTINE
25Ø9ØMODE3
251ØØ*KEY Ø "RUN |M"
2511Ø*KEY 1 "DELETE1ØØØ,11ØØ |M AUTO1
ØØØ,1Ø |M"
2512Ø*KEY 2 "LIST1ØØØ,1Ø99 |M AUTO11Ø
Ø,1Ø |M"
2514Ø*KEY 4 "GOTO31ØØØ |M"
2515Ø*KEY 5 "VDU2 |M LIST1ØØ,1ØØØØ |M
 VDU3 |M"
2516Ø*KEY 6 "       "
2517Ø*KEY 7 "MODE7 |M"
2518Ø*KEY 8 "
                  "

2519Ø*KEY 9 "|M       "
253ØØPROCWINDOW
26999END
27ØØØDEF PROCWINDOW
27Ø1ØS$="I          "
27Ø3ØPRINTTAB(8,23)S$S$S$S$S$S$S$"I"
27Ø5ØVDU28,Ø,22,79,Ø
27Ø52 PRINTTAB(Ø,18);"B E E B U G
Mini Text Editor"
27Ø54  PRINT"(printout routine by Ian
 Sinclair)"
27Ø6ØPRINTTAB(Ø,22);
27Ø7ØPRINT"PRESS KEY F1 OR F2 TO STAR
T (F2 PRINTS ADDRESS)"
27Ø8ØPRINT'"OR F4 OR F5 FOR PRINTOUT"
27Ø9ØPRINT''''
272ØØENDPROC
31ØØØREM PRINTOUT WITHOUT NUMBERS
31Ø1ØMODE3:PRINT'''
31Ø2ØPRINTTAB(2)"Please choose contin
uation or new print- type line number
 or zero.";:INPUT A$:L%=VAL(A$):W%=Ø
31Ø22N%=3585
31Ø23VDU 2
31Ø24REPEAT
31Ø26N%=N%+?(N%+2)
31Ø28UNTIL?(N%+1)=1Ø
31Ø29J%=N%+2
31Ø3ØREPEAT
31Ø32L%=L%+1
31Ø34FORM%=1TO6
31Ø35PRINTCHR$(1)" ";
31Ø36NEXT
31Ø37FORK%= J%+1 TO N%+?J%-1
31Ø4ØPRINTCHR$(1)CHR$(?K%);
31Ø42IF ?K%=32 THEN W%=W%+1
31Ø44IF ?K%=13 VDU 1,1Ø
31Ø5ØNEXT
31Ø4ØPRINTCHR$(1)CHR$(?K%);
31Ø42IF ?K%=32 THEN W%=W%+1
31Ø44IF ?K%=13 VDU 1,1Ø
31Ø5ØNEXT
31Ø52IF L%>=31 THEN PRINTCHR$(1)CHR$(
12):L%=Ø
31Ø6ØPRINTCHR$(1)CHR$(13)
31Ø7ØN%=N%+?J%:J%=N%+2
31Ø8ØUNTIL ?(N%+3)=244
31Ø85VDU 3
31Ø9ØPRINT" Lines of last page- ";L%'
'"Word total- "W%
311ØØ END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

A/D CONVERTER UPGRADE

If you have tried to do this upgrade by fitting IC73 a uDP7002 without success then Alan Mothersole has the answer. You must also fit IC77 a 74LS00 (Quad 2-ip NAND).

POINTS ARISING

STARFIRE JOYSTICKS
by John Yale

If you own the BEEBUGSOFT program STARFIRE, you may like to modify it for joystick control as in the following listing. This achieves proportional control, and allows you to follow the alien ship much more accurately; it even brings warp 3 operation within grasp. The joystick will act like an aircraft control column. Therefore if the enemy ship is at the top of the screen, the joystick must be pulled back. This can be confusing at first. Removing the negative signs in front of the expression for dx% and dy% in lines 1040 and 1070 will change this. If you want to experiment with the response speed of the joysticks, then try altering the value &600 in line 1040, and the &800 in line 1070.

```
1020 DEFPROCmove
1030 LOCAL dx%,dy%
1040 dx%=-(ADVAL(2) DIV &600-16)*velocity%
1050 x%=x%+dx%
1060 IF SGN(dx%)=SGN(xx%) xx%=-xx%
1070 dy%=-(ADVAL(1) DIV &800-8)*velocity%
1080 y%=y%+dy%
1090 IF SGN(dy%)=SGN(yy%) yy%=-yy%
1170 ENDPROC
1410 REM *** LASER PROCEDURE ***
1420 REM
1430 DEFPROClaser
1440 IF (ADVAL(0) AND 1)=0 THEN 1760
1450 IF next%>TIME THEN 1760
```

BUG FIX PROVISOS

Mr Nunan points out that when using the bugs fix given in the July issue a problem is encountered when using the command *TAPE. The pointer to the patch at &218 and &219 gets reset to its original value so that although you think that the patch is in operation the computer doesn't. I had left a stray *TAPE12 in my program, otherwise would not have encountered the problem. Many thanks for your disassembler program which enabled me to track this down.

Frank Huruid also informs us that the bugs fix does NOT work with the *SPOOL command. If you intend to use spooling then the fix must be disabled first.

SOUNDS EXPLAINED

On page 22 of issue 4 we said listen to the amazing sound obtained when you try:
*KEY 0 SOUND 2,-15,100,1:SOUND 3,103,100,1|M and press key f0 to hear it.
This is explained by A. Williams as follows:

The SOUND statement appears to use MOD 32 for the envelope number, so SOUND 3,103,100,1 refers to envelope 7. If this could be defined, the parameters would be stored at locations &860 to &86C, which seem to be part of the sound function workspace. In this case the following values normally result (Mine are different, so perhaps we are all hearing something different - JY):
1,1,128,100,1,128,100,1,128,100,1,128,100
Although some of these parameters are out of range, the envelope still functions. Each envelope uses three extra bytes for future expansion - location &86E contains 128, giving a continuous sound even with a duration value of 1.

WRONG AUTHOR

In BEEBUG No 6 we wrongly attributed the Memory Display Utility to Chris Bingley: our apologies to Doug Blyth who actually wrote it.

BEEBUG BACK ISSUE SERVICE FOR MEMBERS

We currently keep all back issues of the magazine in print, and these are available to members at 80p + SAE.
To order back issues, send multiples of 80p together with a stamped self addressed envelope large enough for your order. Give your membership number with your order, and send to: BEEBUG, Dept 1, 374 Wandsworth Road, London, SW8 4TE

Contents of all previous issues:-

April: Program features: '3D Surface Plotter' and 'Ellipse Drawing' plus two games - 'Moon Lander' and '3D Noughts & Crosses'. Memory and Parallel Printer Upgrade; Making Sounds; Operating System Calls; Members' Discounts. Hints and Tips cover screen scrolling (paging); Cassette File speeds; Windows, and Coloured Text.

May: Program features: 'Spirals & Chords', 'Bomber' (16k), 'Careers' (32k). Plus articles on Graphics, Using The Assembler, Writing Games Programs (Pt 1), Screen Addresses, and Composite Video Output. How to Merge Programs. A Review of Neil and Pat Cryer's book "Basic Programming on the BBC Microcomputer". Hints and Tips cover Operating System Calls; Peek and Poke equivalents; getting rid of the cursor; Printers, ROMs and EPROMs. Plus Members' Discounts, BEEBUG Hams, and Local User Group Index.

June: Program features: 'Polygon' (16k), 'Screenplay' (32k), Mini Text Editor (32k) and the game 'Maze Trap' (32k). There are articles on Structuring in BBC Basic, User Port on A and B (including upgrading Part 2); 110 Baud Fix; Games Writing (Pt 2); Assembler (Pt 2); Reading a Character from Screen; Verifying tapes; Procedure Anomaly; A Review of Videos and TVs. Discounts, User Group Index, BEEBUG Hams; Hints and Tips.

July/August: Programs: 'Patchwork'; Epson and Seikosha Screen Dumps; and the game 'Chunky Invaders' (16k). User Defined Keys; Using Mode 7; Structuring (Pt 2); Nine Programs Reviewed. User port (Pt 2); BBC Bugs Fix; INPUT function; How to transfer programs between machines; String Handling Tip. Discounts, User Group Index; and Members' corner. From this issue onwards 'Hints and Tips' are listed on the front cover.

September: Programs and Games: 'Higher/Lower' (16k); 'Hangman' (16k); String Search program; Screenplay (16k); Multicoloured Characters, and Multicoloured Beeb. From this issue onwards it is indicated whether programs have been tested on OS 1.1. Articles on 'A Safe Verify'; The new User Guide; Operating System Notes; Unexplained Errors - Explained; Postbag; Acorn's ROM replacement charge; Logic (Pt 1); Points Arising; User Key Update; Procedure/Function Library which covers 'Days Between Two Dates' and a 'Yes/No' function. Plus 11 useful Hints and Tips.

October: Programs: 'Memory Display Utility' (16k); 'Union Jack' (16/32k); 'Calendar Generator' (16k); 'Aliens' (16/32k); all these have been tested on OS 0.1 and OS 1.1 BBC Basics - ideas for the less experienced user; Screen to Cassette (16/32k); Acorn Press Release on OS 1.2 and Issue II Basic. Negative Inkey; Logic (Pt 2). The Procedure Library covers double height printing. Debugging (Pt 2) - this is continued from 'Unexplained Errors' in the Sept issue. Details are given on the 'Tube' and Second Processor options; Software Reviews; Local User Group Index and Discounts.

IF YOU WRITE TO US

Back Issues (Members only)

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

Subscriptions Address

BEEBUG
Dept 1
374 Wandsworth Rd
London
SW8 4TE

Subscriptions

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

Software (Members only)

See details overleaf. Available from the subscriptions address.

Contributions and Technical Enquiries

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

Editorial Address

BEEBUG
PO Box 50
St Albans
Herts
AL1 2AR

Ideas, Hints & Tips, Programs, and Longer Articles

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

In Next Month's Issue

How to recover from a "Bad Program". Joystick Drawing and Painting Program. Definitive Seikosha Dump. Reviews of software from the BBC. Accessing the 6845 - Software control of the video controller chip. Plus a very clever program that will edit your own Basic programs for you. More games. More Hints & Tips.

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

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