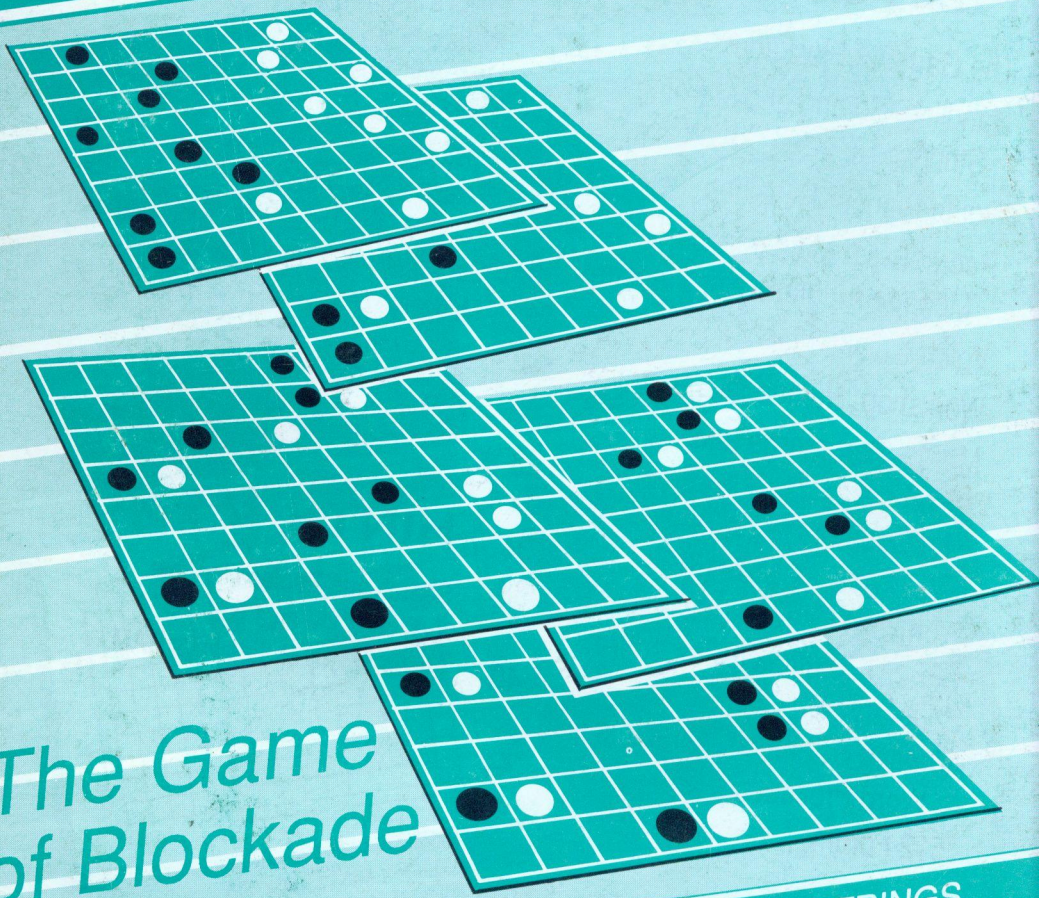
Vol.10 No.2 June 1991

BEEBUG

FOR THE BBC MICRO & MASTER SERIES



The Game of Blockade

- WORDWISE USER'S NOTEBOOK
- HANDLING STRINGS
- ROUTE PLANNER REVISITED
- PART-MERGE AND SAVE

BEEBUG Vol.10 No.2 June 1991

FEATURES

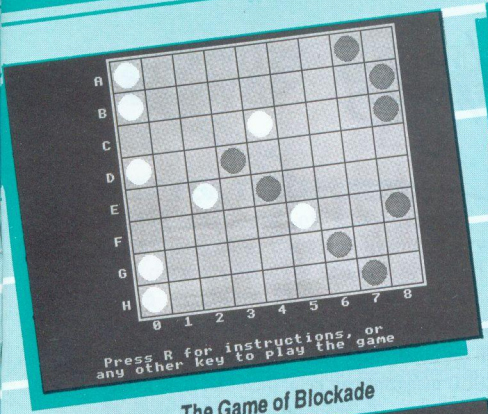
REGULAR ITEMS

HINTS & TIPS

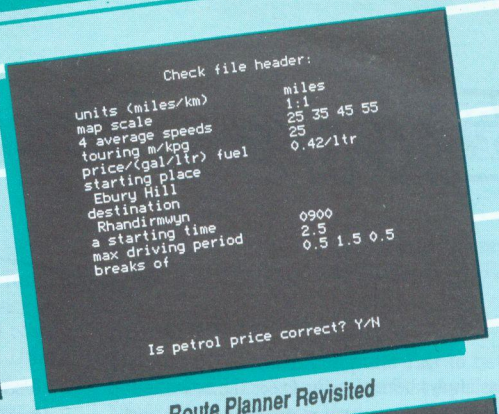
PROGRAM INFORMATION

All listings published in BEEBUG magazine are produced directly from working programs. They are formatted using LISTO 1 and WIDTH 40. The space following the line number is to aid readability only, and may be omitted when the program is typed in. However, the rest of each line should be entered exactly as printed, and checked carefully. When entering a listing, pay special attention to the difference between the digit one and a lower case l (L). Also note that the vertical bar character (Shift \) is reproduced in listings as |.

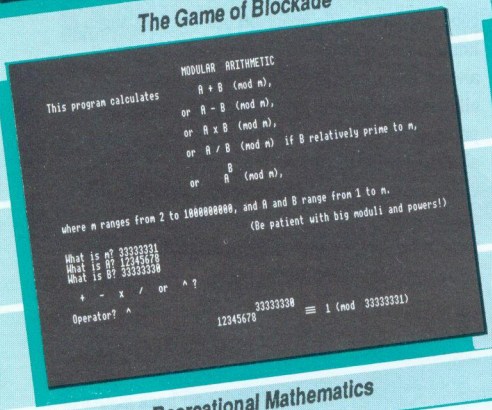
All programs in BEEBUG magazine will run on any BBC micro with Basic II or later, unless otherwise indicated. Members with Basic I are referred to the article on page 44 of BEEBUG Vol.7 No.2 (reprints



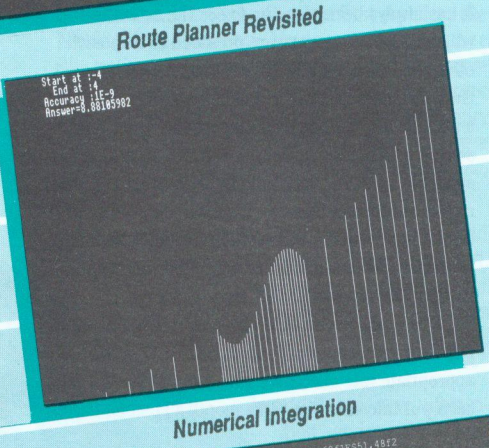
The Game of Blockade



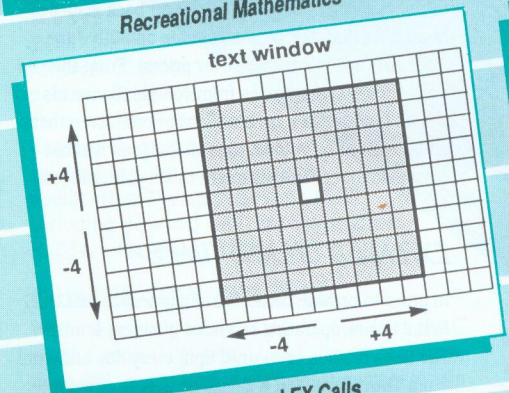
Route Planner Revisited



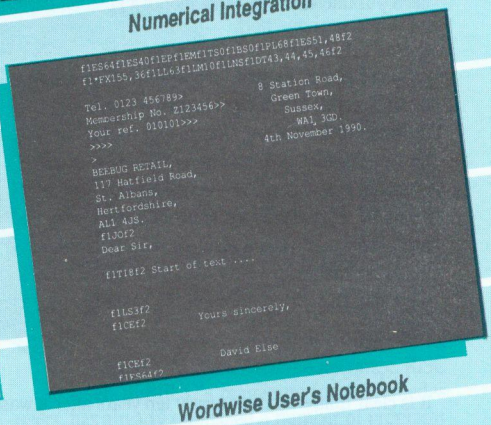
Recreational Mathematics



Numerical Integration



VDU and FX Calls



Wordwise User's Notebook

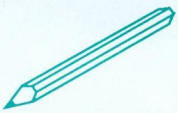
available on receipt of an A5 SAE), and are strongly advised to upgrade to Basic II. Any second processor fitted to the computer should be turned off before the programs are run.

Where a program requires a certain configuration, this is indicated by symbols at the beginning of the article (as shown opposite). Any other requirements are referred to explicitly in the text of the article.

Program needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings



It has always been pleasing to see the extent to which BEEBUG readers are prepared to help each other, and indeed other BBC micro users in general. Over the years we have published a number of letters in our Postbag pages where the writer has sought information or help that is not otherwise available, or at the very least difficult to access. One recent example of this was the letter we published from a Lithuanian student with a model B seeking contact and correspondence with other users in Britain. Within days of that issue going out I received a phone call from one reader who offered to pay for a year's subscription to BEEBUG (at the appropriate rate of course), and indeed this has now been put into effect (see this month's Postbag page).

I can remember another instance not so long ago when, on a more technical note, one reader was seeking ways of storing efficiently a triangular array, i.e. one in which only half (approximately) of the elements of a normal two-dimensional array are occupied. That clearly set many minds working and provoked a spate of suggestions which we passed on to the original letter writer (and some of the replies were also followed up through the Postbag pages as well).

It is clear that even now, nearly ten years after it was first launched, the BBC micro in its various guises is still used for a huge variety of applications. If you have a particular problem which you wish to solve using your BBC micro then the chances are that some other BEEBUG reader (if not more than one) may have the very program you need, or at least be able to provide the information you want.

It seems to me that this type of activity reflects very much the ethos of the BBC micro market, and if BEEBUG magazine can help, then that is all to the good. Of course, the space on our Postbag pages is limited, and it would be quite unfair to devote it always to appeals for help. There is another alternative, and that is to use our Personal Ads pages and submit a WANTED type of ad. Keep it short and we will include it in the next available issue. Also, as time goes by, fewer and fewer BBC micro products remain available, but what you want may be readily obtainable from some other user who no longer has a need for it.

BACK ISSUE MAGAZINES AND DISCS

Please note that our present offer on back issues of magazines and discs will terminate on 30th June 1991 (see inside back cover for prices). From then on only discs and magazines from volume 6 onwards will be kept in stock (at standard prices), Even then, all issues will be subject to availability as it is not economic to reprint magazines in small quantities.

EXTENDED OPENING TIMES

In order to provide better customer service BEEBUG Retail is now operating extended business hours. The Showroom is open until 6pm every day and until 8 pm Thursdays. You will also be able to contact Telesales six days a week, Monday to Friday from 9am to 6pm and Saturday from 9am to 5pm, if you wish to place an order or renew your membership over the phone.

Mike Williams

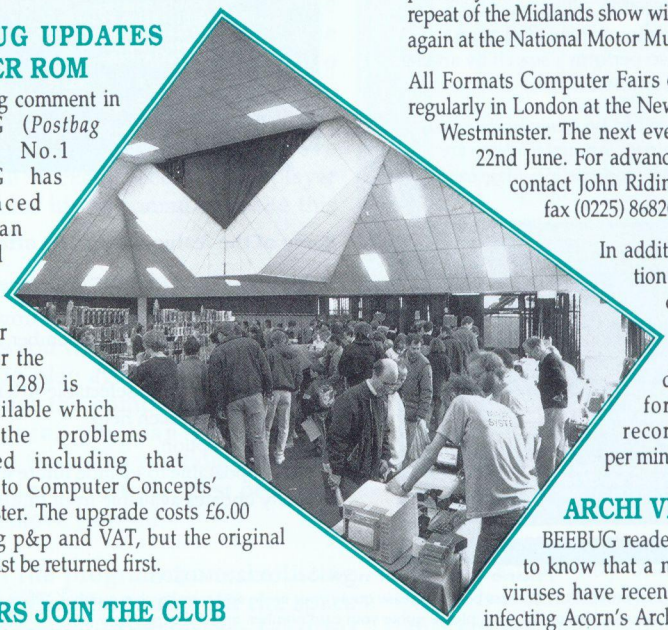
BEEBUG News

HELP FOR TORCH USERS

Former employee of now defunct Torch Computers, Mark Cook, is offering a service to Beeb owners with Torch add-ons, or indeed other Torch products including the final Unix workstation. Most spare parts, software and original manuals are usually available for all systems. All enquiries should be directed (by post - no telephone calls) to Mark at 70 Stubbs Lane, Braintree, Essex CM7 6XB. An SAE would also be appreciated.

BEEBUG UPDATES MASTER ROM

Following comment in BEEBUG (*Postbag* Vol.10 No.1 BEEBUG has announced that an updated version of its Master ROM (for the Master 128) is now available which cures the problems described including that relating to Computer Concepts' Spellmaster. The upgrade costs £6.00 including p&p and VAT, but the original ROM must be returned first.



C USERS JOIN THE CLUB

We have recently received information from the C Users Group (UK) who would like to encourage more BBC micro (and Archimedes) owners to join their ranks. CUG(UK) caters for C programmers with a bi-monthly journal (which looks most interesting from the sample copy sent). It also provides a public domain library of C programs (source and compiled), and runs a Birmingham-based bulletin board.

Standard rate subscriptions are £12 per year (reduction for students). For further information, or to join contact the Membership Secretary, CUG(UK), 64 Southfield Road, Oxford OX4 1PA.

ALL FORMATS SUCCESS

The first All Formats Computer Fair to be held outside London took place on 21st April at the National Motorcycle Museum, Solihull (see illustration). Every stand was taken and the venue was packed to capacity. As a result some future fairs will be held at other locations around Britain, probably Leeds and Bristol. First, there will be a repeat of the Midlands show with a Fair on 9th June, again at the National Motor Museum, Solihull.

All Formats Computer Fairs continue to be held regularly in London at the New Horticultural Hall, Westminster. The next event is scheduled for 22nd June. For advance tickets to all Fairs contact John Riding on (0225) 868100, fax (0225) 868200.

In addition, a new information service on all forthcoming All Formats Computer Fairs is available by dialling 0898 299 389 for a three minute recorded message (38p per min off peak, 45p peak).

ARCHI VIRUSES

BEEBUG readers may be interested to know that a number of computer viruses have recently been discovered infecting Acorn's Archimedes range. RISC User, BEEBUG's magazine for Archimedes users, has already published a number of programs which both help to diagnose a virus, cure the infected machine and give it substantial protection against the threat of future infection.

It should be pointed out that under the Computer Misuse Act (1990), the release of a virus is now illegal. New Scotland Yard has a Computer Crimes Unit at 2 Richbell Place, London WC1 8XD, and any evidence or information regarding computer viruses should be sent to them at that address.

Magscan

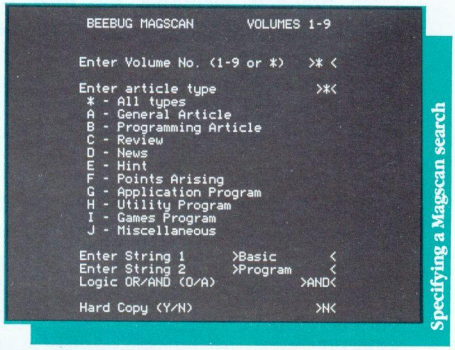
Comprehensive Magazine Database for the BBC Micro and the Master 128

An updated version of Magscan, which contains the complete indexes to all BEEBUG magazines from Volume 1 Issue 1 to the latest Volume 9 Issue 10

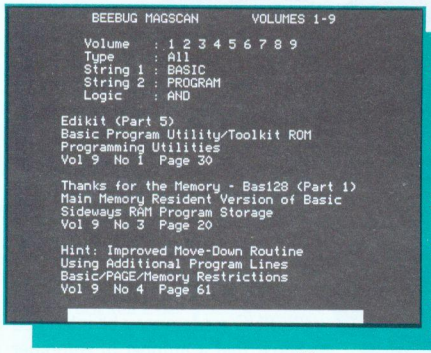
Magscan allows you to locate instantly all references to any chosen subject mentioned anywhere in the 90 issues of BEEBUG magazine.

Just type in one or two descriptive words (using AND/OR logic), and you can find any article or program you need, together with a brief description and reference to the volume, issue and page numbers. You can also perform a search by article type and/or volume number.

The Magscan database can be easily updated to include future magazines. Annual updates are available from BEEBUG for existing Magscan users.



Specifying a Magscan search



Entries retrieved from Magscan files

Some of the features Magscan offers include:

- full access to all BEEBUG magazines
- rapid keyword search
- flexible search by volume number, article type and up to two keywords
- keyword entry with selectable AND/OR logic
- extensive on-screen help
- hard copy option
- easily updatable to include future magazines
- yearly updates available from BEEBUG

Phone your order now on (0727) 40303

or send your cheque/postal order to the address below. Please quote your name and membership number. When ordering by Access, Visa or Connect, please quote your card number and the expiry date.

Magscan complete pack, contains disc, manual and release notes: **£9.95**+p&p

Stock codes:	0005b	5.25"disc 40 track DFS	1457b	3.5" ADFS disc
	0006b	5.25"disc 80 track DFS		

Magscan update, contains disc and release notes: **£4.75** +p&p

(for update, please return old disc, label or evidence of purchase)

Stock codes:	0011a	5.25"disc 40 track DFS	1458a	3.5" ADFS disc
	0010a	5.25"disc 80 track DFS		

Postage: **a** - 60p UK, £1 Europe + Eire, £2.40 Elswhere **b** - £1.50 UK, £2.50 Europe + Eire, £4.80 Elswhere

BEEBUG Ltd, 117 Hatfield Road, St Albans, Herts AL1 4JS. Telephone (0727) 40303 Fax (0727) 860263

The Game of Blockade

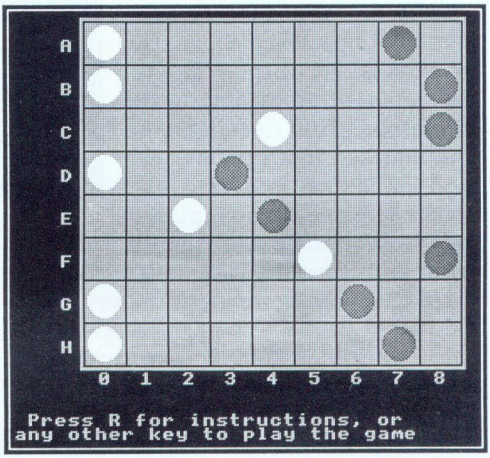
by Eric Bramley

Here is a simple game for two players, one of which can be the computer. Each player has one counter in each of eight rows of squares (see illustration), and at each turn can move any one of his counters towards that of his opponent in the same row. You can move to any vacant square between the two counters, but cannot go backwards or jump over the opponent. Eventually the two counters in each row will be adjacent to each other, and the game then ends. However, there are two different versions of this game, according to which the winner is *either* the player who made the last move when the blockade occurs, *or* the player who forces his opponent to make this last move.

This implementation of the game allows you to choose which of the two versions you want to play. You can also choose to play against another player, or against the computer, and in the latter case, you can choose, after studying the initial position, whether to make the computer play first or to make the first move yourself.

Type in the program from the listing (making sure that you omit the space (which is included for readability) between the line number and the start of the first instruction on that line (see Program Notes - the program is very tight on space). The program can then be run to play the game.

The initial positions of the counters are set up almost, but not quite, at random. If a completely random arrangement is used, the game is very heavily weighted in favour of the first player, irrespective of which version of the game is being played. In setting up the position, therefore, a constraint has been introduced which makes it equally likely that the advantage lies with either player.



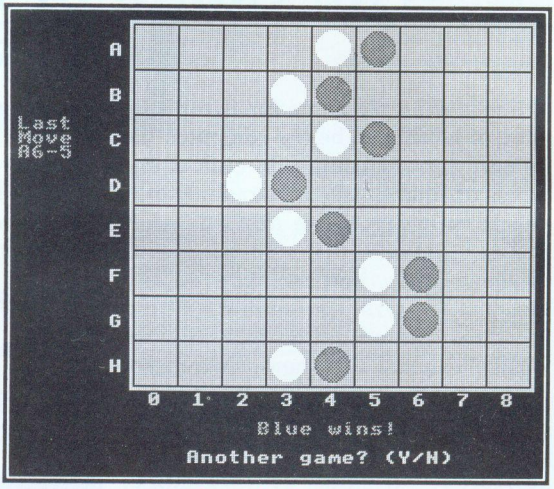
Start of a new game

In fact, this game is easily seen to be a simple version of the classical game of Nim, in which there are a number of heaps of counters, and the players take turns at removing any number of counters from any one selected heap. There is a complete theory of this game, that enables any position to be analysed as a certain win for one or other of the players (whichever version of the game is in use), provided the correct strategy is adopted at each move. This strategy has been incorporated into the program, so that if you play against the computer you will need to play very accurately in order to win! However, you may enjoy studying the computer's moves in order to work out the winning method.

PROGRAM NOTES

When the program is running there are only about 200 bytes of memory to spare on a model B, so unnecessary spaces and brackets have been omitted from the listing.

The initial arrangement of the counters is determined by PROCsetup, which includes the constraint mentioned above. The position is then displayed, with the option of proceeding to the game or first seeing the summarized instructions. The position is then recovered, and choice of game version, players and first move can be made. After each turn the last move made is displayed. The computer's moves are calculated in PROCcomp, which splits into compA or compB according to whether version A or B of the game is in use. Evaluation of the current position in order to determine the computer's next move is done by PROCcalc. A short delay has been introduced to slow down the computer's responses!



The end of a game

At the end of the game there is the option of a replay, when a new starting position is generated, and if required the instructions can again be reviewed before playing.

```
BLOCKADE

Each player in turn advances a counter,
in any one selected row, towards the
opponent's counter in the same row,
moving to any vacant square between the
two counters. This continues until no
further move can be made.

There are two versions of the game :

A : The winner is the player who makes
the final move to complete the blockade;

B : The winner is the player who forces
the opponent to make this final move.

You can choose version A or B. You can
also choose to play against another
player or against the computer. The
computer always plays the blue pieces.

        Press Space to continue
```

On-line instructions

```
   10 REM Program Blockade
   20 REM Version B1.2
   30 REM Author  Eric Bramley
   40 REM Beebug  June 1991
   50 REM Program subject to copyright
   60 :
  100 ONERRORMODE7:PROCerror:END
  110 DIMF(3),S(8),T(3),V(8),X(8),Y(8),A
(8,3),C(8,2),D$(2)
  120 ENVELOPE1,5,4,-4,0,22,22,0,120,0,0
,-20,120,120
  130 FORC=0TO8:X(C)=364+100*C:NEXT
  140 FORR=1TO8:Y(R)=1100-100*R:NEXT
  150 PROCcounter:PROCsetup
  160 MODE1:PROCboard
  170 PROCstart
  180 IFGET=82MODE7:PROCintro:REPEAT:UNT
ILGET=32:GOTO160
  190 PRINTTAB(0,27)SPC159:PRINT TAB(13,
27)"Choose version required."TAB(12,29)"
Press A (final move wins)"TAB(14,30)"or
```

```
 B (final move loses)"
  200 REPEAT:V=GET:UNTILV=65ORV=66:IFV=6
5V$="wins."ELSEV$="loses."
  210 PRINTTAB(0,27)SPC159:PRINTTAB(5,27
)"Player making the final move "V$TAB(8,
29)"Do you want to play against"TAB(13,3
0)"the computer? (Y/N)"
  220 G=GET:IFG=89PROCone ELSEPROCtwo
  230 PROCwin
  240 IFGET=89PROCclear:PROCsetup:GOTO17
0ELSEMODE7:END
  250 END
  260 :
 1000 DEFPROCerror
 1010 REPORT:PRINT" at line ";ERL
 1020 ENDPROC
 1030 :
 1040 DEFPROCcounter
 1050 D$(1)="Red":D$(2)="Blue"
 1060 VDU23,224,0,0,0,1,3,7,15,31
 1070 VDU23,225,0,0,126,255,255,255,255,
255
 1080 VDU23,226,0,0,0,128,192,224,240,24
8
 1090 VDU23,227,31,63,63,63,63,63,63,31
 1100 VDU23,228,255,255,255,255,255,255,
255,255
 1110 VDU23,229,248,252,252,252,252,252,
252,248
 1120 VDU23,230,31,15,7,3,1,0,0,0
 1130 VDU23,231,255,255,255,255,255,126,
0,0
 1140 VDU23,232,248,240,224,192,128,0,0,
0
 1150 C$=CHR$224+CHR$225+CHR$226+CHR$8+C
HR$8+CHR$8+CHR$10+CHR$227+CHR$228+CHR$22
9+CHR$8+CHR$8+CHR$8+CHR$10+CHR$230+CHR$2
31+CHR$232
 1160 ENDPROC
 1170 :
 1180 DEFPROCsetup:S=RND(-TIME)
 1190 REPEAT:S=0
 1200 FORR=1TO7
 1210 S(R)=RND(7):S=S+S(R)
 1220 IFS(R)=7C(R,1)=0:C(R,2)=8 ELSEC(R,
1)=RND(8-S(R))-1:C(R,2)=C(R,1)+S(R)+1
 1230 NEXTR
 1240 PROCcalc(7)
 1250 UNTILF=1
 1260 S8=0:FORK=1TO3:S8=S8+F(K)*2^(K-1):
NEXT
 1270 REPEAT:P=RND(7):UNTILP<>S8
 1280 Z=RND(2):IFZ=1S(8)=S8 ELSES(8)=P
 1290 S=S+S(8)
 1300 IFS(8)=7C(8,1)=0:C(8,2)=8 ELSEC(8,
1)=RND(8-S(8))-1:C(8,2)=C(8,1)+S(8)+1
 1310 ENDPROC
 1320 :
 1330 DEFPROCboard
 1340 VDU23,1,0;0;0;0;
 1350 GCOL0,131:VDU24,352;196;1268;1012;
:CLG
 1360 VDU19,2,6,0,0,0
 1370 GCOL0,0
 1380 FORI=0TO9:MOVE360+100*I,204:DRAW36
0+100*I,1004:NEXT
 1390 FORI=0TO8:MOVE360,204+100*I:DRAW12
60,204+100*I:NEXT
 1400 VDU26,5:GCOL0,3
 1410 FORK=1TO8:MOVE304,1060-100*K:VDU(6
4+K):NEXT
 1420 FORK=0TO8:MOVE392+100*K,188:PRINT;
K:NEXT
 1430 ENDPROC
 1440 :
 1450 DEFPROCstart
 1460 VDU19,1,7;0;19,2,7;0;
 1470 FORR=1TO8:FORI=1TO2
 1480 GCOL0,I:MOVEX(C(R,I)),Y(R):PRINTC$
 1490 NEXTI:NEXTR
 1500 VDU19,1,1;0;19,2,6;0;
 1510 VDU4:COLOUR3:PRINTTAB(7,29)"Press
R for instructions, or"TAB(6,30)"any oth
er key to play the game"
 1520 ENDPROC
 1530 :
 1540 DEFPROCintro
 1550 VDU23,1,0;0;0;0;
 1560 PRINT TAB(12,2)CHR$133CHR$141"BLOC
KADE"TAB(12,3)CHR$133CHR$141"BLOCKADE"
 1570 PRINT'"Each player in turn advance
s a counter,"'"in any one selected row,
```

```
towards the"'"opponent's counter in the
same row,"'"moving to any vacant square
between the"'"two counters. This continu
es until no"'"further move can be made."
 1580 PRINT'"There are two versions of t
he game :"'"A : The winner is the playe
r who makes"'"the final move to complete
 the blockade;"'"B : The winner is the p
layer who forces"'"the opponent to make
this final move."
 1590 PRINT'"You can choose version A or
 B. You can"'"also choose to play agains
t another"'"player or against the comput
er. The"'"computer always plays the blue
 pieces."
 1600 PRINT'TAB(6)CHR$134"Press Space to
 continue"
 1610 ENDPROC
 1620 :
 1630 DEFPROCone
 1640 PRINTTAB(0,29)SPC79TAB(7,29)"Do yo
u want to go first? (Y/N)"
 1650 REPEAT:G=GET:UNTILG=78ORG=89
 1660 IFG=89T=1:PROCplay
 1670 T=2:REPEAT:PROCcomp:PROCplay:UNTIL
S=0
 1680 ENDPROC
 1690 :
 1700 DEFPROCtwo
 1710 T=RND(2)
 1720 REPEAT:PROCplay:UNTILS=0
 1730 ENDPROC
 1740 :
 1750 DEFPROCplay
 1760 IFS=0ENDPROC
 1770 PRINTTAB(0,29)SPC79
 1780 COLOURT:PRINTTAB(2,13)D$(T)" to "T
AB(3,14)"play"TAB(5,29)"Enter row letter
 and column number"TAB(5,30)"of the squa
re you want to move to"
 1790 PRINTTAB(2,18)SPC(6)TAB(2,16)"Row?
"
 1800 REPEAT:REPEAT:G=GET:M=G-64:UNTILAB
S(M-4.5)<4:UNTIL(C(M,2)-C(M,1))>1
 1810 R$=CHR$G:PRINTTAB(6,16)R$TAB(2,18)
"Col?"
 1820 REPEAT:G=GET:N=G-48:UNTILN>C(M,1)A
NDN<C(M,2)
 1830 PRINTTAB(6,18);N
 1840 PROCmove
 1850 ENDPROC
 1860 :
 1870 DEFPROCmove
 1880 VDU5:C=C(M,T)
 1890 GCOL0,3:MOVEX(C),Y(M):PRINTC$
 1900 GCOL0,T:MOVEX(N),Y(M):PRINTC$
 1910 VDU4:VDU28,0,20,8,11,12,26
 1920 C(M,T)=N:S(M)=S(M)-ABS(C-N):S=S-AB
S(C-N)
 1930 PRINTTAB(3,7)"Last"TAB(3,8)"Move"T
AB(3,9)R$;C"-";N
 1940 SOUND3,-13,90,4:T=3-T
 1950 ENDPROC
 1960 :
 1970 DEFPROCcomp
 1980 IFS=0ENDPROC
 1990 PRINTTAB(0,29)SPC79
 2000 COLOUR2:PRINTTAB(15,29)"Computing.
.."
 2010 W=INKEY(150):T=2
 2020 IFV=65PROCcompA ELSEPROCcompB
 2030 ENDPROC
 2040 :
 2050 DEFPROCcompA
 2060 PROCcalc(8)
 2070 IFE=0REPEAT:Z=RND(8):UNTILS(Z)>0:M
=Z:N=C(M,2)-1:R$=CHR$(M+64):PROCmove:END
PROC
 2080 M=0:REPEAT:M=M+1:UNTILA(M,E)=1
 2090 FORK=1TOE:IFF(K)=1A(M,K)=1-A(M,K)
 2100 NEXTK
 2110 SP=0:FORK=1TO 3:SP=SP+A(M,K)*2^(K-
1):NEXT
 2120 N=C(M,2)-S(M)+SP:R$=CHR$(M+64)
 2130 PROCmove
 2140 ENDPROC
 2150 :
 2160 DEFPROCcompB
 2170 n1=0:nn=0
 2180 FORR=1TO8
 2190 IFS(R)=1n1=n1+1
```

Continued on page 14

Handling, Sorting and Merging Strings

Ruben Hadekel discusses some of the finer points of string handling in Basic.

MINIMISING MEMORY USAGE

When a string variable is assigned a value, information on it is stored in a sequence of four bytes called the *string information block* (SIB). The addresses of the SIB's themselves are allocated by the DIM statement if they are array elements, or otherwise assigned when the string is referenced in the program.

The first two bytes of the SIB contain the address of the first character of the string, in the usual low-byte/high-byte order. The third byte contains the number of bytes allocated, which is the length of the string when the address is assigned plus two, and the fourth byte contains the current length of the string. If the string variable is allocated a new value, it is stored at the same address if its length does not exceed the number of bytes allocated. Otherwise, unless the string is at the top of the variables heap, it has to be stored at a new address, and the memory space taken up by the previous value is then wasted.

If a program involves assigning a succession of values to a string variable, and say for instance these are successively 40, 50 and 60 characters long, then the total memory space taken up is 150 bytes, plus whatever is required to store the string name and other identifying header elements. On the other hand, if it is known beforehand that the string is unlikely to exceed say 70 characters, you can assign that length to it say by A$=STRING$(70,"*"), and as long as any subsequent assignment does not exceed 70 characters, the total memory space used remains 70, plus one header.

This method of saving on memory space is valid for single strings and very short arrays, but for an array of any length it would again waste memory.

SORTING

Sorting algorithms are based on comparing pairs of array elements, and interchanging their positions if they are not in correct order. For string arrays (to be sorted in alphabetical order), however, direct swapping of array elements has a serious drawback. In the process, elements would frequently be assigned a string value with a greater length than before, with wastage of memory as explained above. This drastically reduces the amount of memory space that can be used for the array without running into trouble.

An efficient method of sorting string arrays is to interchange not the elements directly, but their respective String Information Blocks.

Thus if *addr%* is the address of the SIB for A$(0), and A$(I%) and A$(J%) are out of order, we can write:

```
IF A$(I%) > A%(J%) temp%=!(addr%+4*I%):
   !(addr%+4*I%) = !(addr%+4*J%):
   !(addr%+4*J%) = temp%
```

The trick is to ascertain the value of *addr%*. In other words, where are the SIB's?

With ordinary Basic, there is a fairly direct way. The command CALL to a machine code routine, with any string as parameter, places the address of the SIB for that string in locations &601 and

&602 (1537 and 1538 decimal). For this purpose the CALL can be to a "dummy" machine code program consisting just of the instruction RTS.

```
 10 DIM H% 20
 20 [
 30 .START
 40 RTS:]
 50 DIM A$(...)
.
[define string A$(bm%)]
.
100 CALL START,A$(bm%)
110 CLS:addr%=?1537+(256*?1538)-4*bm%
```

Here *bm%* is the index (identifier) for any (in practice, you would probably use the lowest ranking) non-empty array element. This is not necessarily element 0 if an array has been edited, and some of its elements deleted. *addr%* is then the SIB address for element 0 of the array. The CLS in line 110 gets rid of anything displayed by the assembler.

Note that CALL may not be used in a procedure.

If you use Basic 128, however, this doesn't work. Basic 128 is nowhere documented, and it is probably best not to attempt to use assembler or machine code with it. There is, however, another way of ascertaining *addr%*, which incidentally isn't a full address (addresses in Basic 128 are 17 bit numbers) but merely the two lower order bytes of the actual address - the Basic interpreter in effect adds &10000 (65536 decimal).

The method consists of exploring the contents of RAM from TOP upwards, trying each location in turn to see if it leads to the string in question (since array DIM statements reserve locations for SIBs as soon as they can after TOP, it saves time if the array in question is dimensioned as early as possible in the program). The following program does this, from PROCfind onwards. What precedes PROCfind is merely a check used by the author to confirm that the string in question has indeed been located, i.e. that PROCfind works. It also illustrates the speed of the procedure, which is quite fast enough to make it unnecessary to resort to machine code.

```
  10 DIMT$(10)
  20 INPUT"Enter String "T$(1)
  30 PROCfind (TOP-1,T$(1))
  40 FOR X%=1 TO LEN(T$(1)):PRINT
CHR$(?(try%-1+X%));:NEXT
  50 END
5000 DEF PROCfind(p%,X$)
5010 len%=LEN(X$):N1%=p%
5020 REPEAT
5030 hit%=2:U1%=0:N1%=N1%+1
5040 try%=65536+?N1%+256*?(N1%+1)
5070 REPEAT
5080 U1%=U1%+1:W1%=?(try%+U1%-1)
5090 V1%=ASC(MID$(X$,U1%))
5100 IF V1%<>W1% hit%=0:UNTIL hit%<2
5110 IF U1%=len% hit%=1
5130 UNTIL hit%=1
5140 ENDPROC
```

The actual search in the procedure PROCfind(p%,X$) starts at address p%+1. If the string in question is the first referenced in the DIM statement, p% can be TOP, which starts the search at TOP+1 - the extra 1 is harmless since the spaces reserved for the SIBs must be preceded by a header. If the string is preceded by some other arrays in the DIM statement, maximum speed is achieved if p% = TOP+n, where n is the number of array elements in all preceding string and integer arrays times 4, plus 4 for each of these arrays (to allow for the fact that space is reserved for index 0). For any

real number array, add 5 times the number of elements, plus 5. These refinements, however, are merely for the sake of speed.

Note that this procedure also works for ordinary Basic. The number 65536 in line 5040 should in principle be deleted, but actually the routine works even if it is not - the Basic interpreter subtracts 65536 (&10000) from any number larger than 65535.

MERGING STRING ARRAYS

In a program of the index or database type, it is common to have two string arrays that require merging. The first is the original record, which may include deletions in editing, signified say by assigning a null value ("") to the deleted array element, the second being entries added in editing. Let the two arrays be denoted by A$(..) and B$(..) respectively, and let *bm%* and *tp%* be the lowest and highest ranking surviving elements in A$(..). The remaining elements in A$(..) are assumed to be in alphabetical order, having been sorted in processing previous to the editing run.

Elements in B$(..) would not necessarily have been entered in alphabetical order, and should first be sorted by the method described above, to avoid an over-long routine in merging the two arrays.

If the array resulting from the merging is to be immediately saved, this can be done easily in a 'save' operation, with the following lines:

```
 260 DEF PROCsave
 270 REPEAT
 280 IF A$(P%)="" P%=P%+1:GOTO 280
 290 IF A$(P%)<B$(R%) PRINT#F%,A$(P%):
P%=P%+1 ELSE PRINT#F%,B$(R%):R%=R%+1
 300 UNTIL P%>tp% OR R%>M%
 310 IF P%>tp% FOR X%=R% TO M%:
PRINT#F%,B$(X%):NEXT:CLOSE#F%:ENDPROC
 320 FOR X%=P% TO tp%:
IF A$(X%)="" THEN 330 ELSE PRINT#F%,A$(X%)
 330 NEXT X%
 340 CLOSE#F%:ENDPROC
```

If, however, the merged array is to be immediately available for further processing, it would be tedious and time-consuming to effect merging by saving, and then to load back from disc, and it is then desirable to effect the merging in RAM, without involving the disc drive at all.

A useful procedure for such a case is to reserve space above HIMEM for temporary storage of the SIBs for the merged array. If A$(..) is dimensioned say to a value 750, then write, at the beginning of the program:

```
HIMEM = HIMEM - 4*750
```

followed at any suitable place by:

```
SIB% = HIMEM + 1
```

where SIB% is the first SIB location for the merged array.

Once again the B$(..) array, which has a variable length of M%, has to be sorted first. SIB addresses for the A$(..) and B$(..) arrays are ascertained as explained above, and found to be M1 and N1 respectively for A$(0) and B$(0). We then write:

```
1760 DEF PROCmerge
1770 R%=0:P%=0
1770 R%=0:P%=0:tp%=10:M%=3
1775 REM Arbitrary assignment for tp% & M%
1780 REPEAT
1790 IF A$(P%)="" P%=P%+1:GOTO 1790
1800 IF A$(P%)<B$(R%) !SIB%=!(M1%+4*P%):
P%=P%+1 ELSE !SIB%=!(N1%+4*R%):
R%=R%+1
```

```
1810 SIB%=SIB%+4:N%=N%+1
1820 UNTIL P%>tp% OR R%>M%
1830 IF P%>tp% FOR X%=R% TO M%:!SIB%=
!(N1%+4*X%):SIB%=SIB%+4:N%=N%+1:NEXT:
GOTO 1860
1840 FOR X%=P% TO tp%:IF A$(X%)="" THEN
1850 ELSE !SIB%=!(M1%+4*X%):N%=N%+1:
SIB%=SIB%+4
1850 NEXT
1860 FOR X%=0 TO N%:!(M1%+4*X%)=
!(HIMEM+1+4*X%):NEXT
1870 FOR X%=0 TO M%:!(N1%+4*X%)=0:
NEXT:ENDPROC
```

The procedure counts the number of elements N% (now all non-null) in the merged array, and then line 1860 ensures that it is still referenced by A$(..), enabling it to be used in further processing.

Line 1870 is essential if the program allows more than one editing run before recording. This is because SIBs for elements of B$(..) are transferred to A$(..) by the merging. The second time round, if an element in B$(..) is not longer than the element with the same index in the first cycle, it will acquire an SIB with the same address as an element in A$(..), so that there will be two different strings with the same address, with horrendous results. Line 1870 ensures that at each run the B$(..) elements have addresses different from any belonging to A$(..).

With the BBC model B, the above procedure may not be used if MODE is changed during the program. With a Master series machine, the position of HIMEM is safeguarded if the shadow modes, 128 to 135, are used instead of the normal modes 0 to 7. In Basic 128 HIMEM is fixed regardless of mode. B

The Game of Blockade (continued from page 10)

```
 2200 IFS(R)>1nn=nn+1:k=R
 2210 NEXTR
 2220 IFnn>1PROCcompA:ENDPROC
 2230 IFnn=0REPEAT:Z=RND(8):UNTILS(Z)=1:
M=Z:N=C(M,2)-1:PROCmove:ENDPROC
 2240 M=k
 2250 IFn1 MOD2=1N=C(M,1)+1ELSEN=C(M,1)+
2
 2260 PROCmove
 2270 ENDPROC
 2280 :
 2290 DEFPROCcalc(n)
 2300 FORR=1TOn:V(R)=S(R)
 2310 FORK=1TO3
 2320 A(R,K)=V(R)MOD2:V(R)=V(R)DIV2
 2330 NEXTK:NEXTR
 2340 E=0:F=0
 2350 FORK=1TO3:F(K)=0:T(K)=0
 2360 FORR=1TOn:T(K)=T(K)+A(R,K)
 2370 NEXTR
 2380 IFT(K)MOD2=1F(K)=1:E=K:F=1
 2390 NEXTK
 2400 ENDPROC
 2410 :
 2420 DEFPROCwin
 2430 VDU28,0,22,8,11,12,26
 2440 PRINTTAB(0,27)SPC159
 2450 IFV=65T=3-T
 2460 COLOURT:PRINTTAB(20,28)D$(T)" wins
!"
 2470 COLOUR3:PRINTTAB(15,30)"Another ga
me? (Y/N)"
 2480 SOUND&11,1,90,40
 2490 ENDPROC
 2500 :
 2510 DEFPROCclear
 2520 VDU28,0,22,8,6,12,26
 2530 PRINTTAB(0,27)SPC159
 2540 PRINTTAB(15,29)"Resetting..."
 2550 VDU5,19,1,2;0;19,2,2;0;
 2560 GCOL0,3:FORR=1TO8:FORI=1TO2:MOVEX(
C(R,I)),Y(R):PRINTC$:NEXTI:NEXTR
 2570 ENDPROC
```

B

Route Planner Revisited

Jim Phillips adds some further improvements to our popular route planning program, covering use with the ADFS, pricing in litres, and creation of route files.

Having made use of Barry Thorpe's Motorists Route Planner (Vol.7 No.6) with some success, the improvements suggested by Alan and Philip Prior (Vol.9 No.9) were a welcome addition. Using a Master computer, however, I had filed my program on ADFS, with which the new routines are not compatible. I have, therefore, added routines to make the program run with either filing system. The routines are based on those described by Lee Calcraft in Vol.6 No.9 and also make use of the fact that the current ADFS directory is held in memory on the Master.

```
Check file header:

units (miles/km)          miles
map scale                 1:1
4 average speeds          25 35 45 55
touring m/kpg             25
price/(gal/ltr) fuel      0.42/ltr
starting place
 Ebury Hill
destination
 Rhandirmwyn
a starting time           0900
max driving period        2.5
breaks of                 0.5 1.5 0.5


Is petrol price correct? Y/N
```

Screen display produced by the Route Planner program

LITRES

Since the original was written the practice of pricing petrol in litres has become almost universal and I have incorporated the ability to choose either gallons or litres. A small irritant in the original program was the fact that if a break was due after the last entry this was printed out regardless of the fact that "Q" had been entered. Repositioning line 2000 as line 1925 has overcome this by moving the test for a break from the end of the loop to the beginning.

Type in Listing 1, *SPOOL it as a text file then load the previously updated MoPlan program. Now *EXEC in the text file and save the result as a new program with the name MoPlan2. Alternatively, load the original program, type in the new lines directly, then save it as MoPlan2 as before.

CREATING ROUTE FILES

When preparing the text route programs, I have found that while in the midst of reading the map and making decisions about the route my mind occasionally becomes a blank and I cannot remember the format of the entry. To overcome this I have written the short program in listing 2.

The first part of the program prompts for the Header information and, after confirmation, creates a file and saves it. The second part prompts for the information on the current leg of the route. When this has all been entered it is displayed. If satisfactory it is saved and the next leg processed. This cycle is continued until the letter "Q" is entered when the file is closed.

Listing 1 incorporates in addition the amendments detailed in Points Arising, Vol.10 No.1, and also a useful improvement suggested by A.J. Lambell, which ensures that petrol consumption is correctly calculated when working in kilometres.

Listing 1

```
10 REM          >RoutMod
20 REM Version B1.3
30 REM Author   Barry Thorpe
31 REM Mods     A & P Prior and
32 REM          Jim Phillips
40 REM BEEBUG   June 1991
```

```
   50 REM Program subject to copyright
 1015 IF INKEY(-256)=253 catad%=&C400 EL
SE catad%=&1200
 1020 DIM avspeed(4),estmpg(4),break(10)
,ldir%(20),loc%(20)
 1400 UNTIL A%=13 OR LENS$>40 OR EOF#X
 1510 DATA units (miles/km),map scale,4
average speeds,touring m/kpg,price/(gal/
ltr) fuel,starting place,destination,a s
tarting time,breaks,intermediate places
 1680 NEXT:PRINT
 1710 pprice$=FNgetstring
 1711 IF INSTR(pprice$,"/") pprice=VAL(L
EFT$(pprice$,(LENpprice$-4))):pprice$=RI
GHT$(pprice$,3) ELSE pprice=VAL(pprice$)
:pprice$="gal"
 1720 READ item$:PRINT item$ TAB(25);ppr
ice;"/";pprice$
 1805 IF INSTR("Nn",V$) PRINTTAB(5,21)"E
nter new price/";pprice$;:INPUT" : "npri
ce$:pprice=VAL(nprice$):@%=&20210:PF%=1
 1810 PRINTTAB(4,19)SPC35:PRINTTAB(0,21)
SPC39:PRINTTAB(0,6)SPC30:PRINTTAB(0,6)"p
rice/(gal/ltr)"TAB(25,6);pprice;"/";ppri
ce$:@%=10
 1925 IF brktimer>=break(0) THEN PROCins
ertbreak
 2000   2035 IF LEFT$(pprice$,1)="l"
gallons=4.
55*gallons
 2060 PRINT''TAB(5)"fuel used (rounded u
p): ";gallons;" ";pprice$;", costing FAL
SE";
 2335 IF unit$="kms" ktom=0.621 ELSE kto
m=1
 2350 gallons=gallons+dist%*ktom/estmpg(
type)
 2830 Y%=0:A%=0:IF (USR(&FFDA) AND &F)=8
 PROCacat ELSE PROCdcat
 3500 DEFPROCdcat
 3520 R%=0:buffer%=&7000:FOR dr%=0 TO 2
STEP 2
 3530 ?&70=dr%:!&71=buffer%:?&75=3:!&76=
&22000053
 3580 IF ?C%<>32 name$(A%)="":FOR B%=0 T
O 7:name$(A%)=name$(A%)+CHR$((C%?B%)AND1
27):NEXTB%
 3590   4500 DEFPROCacat
 4510 R%=0:csd$="$":dirno%=0
 4520 PROCreadcat
 4530 ENDPROC
 4540 :
 4550 DEF PROCreadcat
 4560 cat%=catad%
 4570 dir=0
 4580 back=FALSE
 4590 PROCgetname
 4600 IF dir PROCdirup
 4610 IF back GOTO 4580
 4620 IF dirno%<>0 PROCdirdn
 4630 ENDPROC
 4640 :
 4650 DEFPROCgetname
 4660 REPEAT
 4670 fln$="":item%=5:none=FALSE
 4680 REPEAT
 4690 dir=cat%?8 AND &80
 4700 fln$=fln$+CHR$(cat%?item% AND &7F)
 4710 item%=item%+1
 4720 IF (cat%?item% AND &7F)<&20 THEN i
tem%=16
 4730 UNTIL item%=16
 4740 IF fln$="" ORcat%?5=0 ORdir THEN n
one=TRUE
 4750 IF NOT none PROCgetfacts
 4760 UNTIL dir OR none
 4770 ENDPROC
 4780 :
 4790 DEFPROCdirup
 4800 ldir%(dirno%)=LENcsd$:loc%(dirno%)
=cat%+26:dirno%=dirno%+1
 4810 csd$=csd$+"."+fln$
 4820 OSCLI"DIR "+fln$
 4830 PROCreadcat
 4840 ENDPROC
 4850 :
 4860 DEFPROCdirdn
 4870 dirno%=dirno%-1
 4880 cat%=loc%(dirno%)
 4890 csd$=LEFT$(csd$,ldir%(dirno%))
 4900 back=TRUE
 4910 OSCLI"DIR ^"
 4920 ENDPROC
 4930 :
```

```
 4940 DEFPROCgetfacts
 4950 X=OPENUP(fln$)
 4960 check$=FNupper(FNgetstring):IF che
ck$<>"M" AND check$<>"K" CLOSE#X:cat%=ca
t%+26: ENDPROC
 4970 name$(R%)=csd$+"."+fln$
 4980 PROCbegend
 4990 beg$(R%)=beg$:end$(R%)=end$:R%=R%+
1
 5000 CLOSE#X:cat%=cat%+26
 5010 ENDPROC
```

Listing 2

```
   10 REM          >CrPlan
   20 REM Version B1.0
   30 REM Author  Jim Phillips
   40 REM BEEBUG  June 1991
   50 REM Program subject to copyright.
   60 :
  100 MODE7:REM ON ERROR GOTO 170
  110 PROCinit
  120 PROCheader
  130 PROCfileroute
  140 PROCclose
  150 END
  160 :
  170 MODE7:CLOSE#ch%
  180 REPORT:PRINT" at line ";ERL
  190 END
  200 :
 1000 DEF PROCinit
 1010 DIM ans$(15)
 1020 FOR n%=1 TO 2
 1030 PRINTTAB(2,n%)CHR$141;CHR$131;CHR$
(157);CHR$132"MOPLAN :- Source File Crea
tor ";CHR$156
 1040 NEXT
 1050 INPUTTAB(5,5)"Directory Name (RETU
RN if $) ";dir$
 1060 IF dir$="" dir$="$"
 1070 OSCLI"DIR "+dir$
 1080 INPUTTAB(5,7)"File name :- ";fln$
 1090 ch%=OPENOUT(fln$)
 1100 PRINTTAB(2,4)"Creating "dir$"."fln
$
 1110 VDU28,0,24,39,5,12
 1120 ENDPROC
 1130 :
 1140 DEFPROCheader
 1150 PROCsetheader
 1160 PRINTTAB(5,19)"All OK (Y/N)?"
 1170 REPEAT:rep$=GET$:UNTIL INSTR("YyNn
",rep$)>0
 1180 IF INSTR("Nn",rep$)>0 CLS:GOTO 115
0
 1190 FOR N%=1 TO 7
 1200 text$=ans$(N%)
 1210 PROCsave
 1220 NEXT
 1230 text$=ans$(9)+"/"+ans$(8)
 1240 PROCsave
 1250 text$=ans$(10)
 1260 PROCsave
 1270 text$=ans$(11)
 1280 PROCsave
 1290 text$=ans$(12)
 1300 PROCsave
 1310 text$=ans$(13)+"/"+ans$(14)
 1320 PROCsave
 1330 CLS
 1340 ENDPROC
 1350 :
 1360 DEF PROCsetheader
 1370 FOR N%=1 TO 13
 1380 READ item$
 1390 PRINTTAB(0,N%)item$;:IF N%=9 PRINT
ans$(8);
 1400 INPUTans$(N%):IF N%=8 AND ans$(N%)
="G" ans$(N%)="gal" ELSE IF N%=8 AND ans
$(N%)="L" ans$(N%)="ltr"
 1410 NEXT
 1420 READitem$
 1430 PRINTTAB(0,14)item$
 1440 INPUTTAB(0,15)ans$(14)
 1450 DATA Units miles/km. (M/K),map sca
le,av. speed (urban),av. speed (outer su
burbs),av. speed (open country),av. (spe
ed motorways),best m/kpg/l figure,fuel i
n gallons or litres (G/L),price fuel/,st
art point,destination
 1460 DATA start time (24 hr. 0000),max
time before break,length of break (/) be
tween entries
 1470 RESTORE 1450
 1480 ENDPROC
 1490 :
```

Continued on page 31

Recreational Mathematics
Modular Arithmetic for Pleasure

by Michael Taylor

It is easy to use Basic to evaluate an expression like (7^10)MOD13, but even (7^12)MOD13 cannot be calculated as an exact integer before 'MOD13' has a chance to bring it back into the range of the integer variables. With the program listed here (called Fermat) the 'MOD' function of BBC Basic is exploited so that modular arithmetic can be carried out with moduli as big as a billion. Expressions such as 123456789^999999936 (mod 999999937) can be evaluated - and on a model B the program takes about 20 seconds to arrive at the answer: 1!

The program can be used for the pleasure of handling big numbers and exploring the patterns of modular arithmetic. Advantage can be taken of prime numbers found with the program Euler of two months ago (BEEBUG Vol.9 No.10). It can also be used to demonstrate the arithmetic of the new 'public key' RSA ciphers, as we shall do in the next article in this series.

When the program is run, the user is asked to enter the modulus 'm' and then the two operands, 'A' and 'B'. Finally he is asked for the operator, '+', '-', 'x', '/' or '^'.

One slight complication: A and B are restricted to be less than or equal to m. This restriction can be removed but is kept here to simplify the explanation and for initial use by the reader.

The famous 'little' theorem of Fermat (not to be confused with his 'last' theorem) says that if x is any number, then when it is raised to the power of p-1, where p is a prime, and then reduced modulo x, it yields the value 1 (so long as it is not a multiple of p).

```
                    MODULAR  ARITHMETIC

This program calculates      A + B  (mod n),
                          or A - B  (mod n),
                          or A x B  (mod n),
                          or A / B  (mod n)  if B relatively prime to n,
                                 B
                          or     A  (mod n),

where n ranges from 2 to 1000000000, and A and B range from 1 to n.
                                        (Be patient with big moduli and powers!)

What is n? 33333331
What is A? 12345678
What is B? 33333330

  +   -   x   /   or   ^ ?

Operator? ^
                                  33333330
                          12345678         ≡ 1 (mod  33333331)
```

The program shows that 12345678^33333330 (mod 33333331)=1

The program Euler (Vol.9 No.10) can be used to provide a prime number. For example 33333331 is prime. Key in 33333331 for m, 12345678 for A, and 33333330 for B. Choose '^' for the operator and after about four seconds on a model B, the answer 1 will appear. Of course, this is equivalent to saying that, if we multiply once more by p, then x^p (mod p) should equal p. Again, this can be tested by keying in 33333331 for both m and B, and 12345678 for A. The answer will be 12345678.

The converse of Fermat's little theorem is not true. You can use the program to show that 2^340 = 1 (mod 341), but 341 is a pseudoprime with factors 11 and 31 (see Lines, page 67).

Division is only possible if the divisor and the modulus are relatively prime. If B and m are relatively prime, then 1/B (mod m) can be found. Try entering 33331 (another prime) for the modulus, 1 for A and 12345 for B. Choose the operator '/' to find that 1/12345 (mod 33331) is 20004. But try 30000 as the modulus while entering, as before, 1 for A and 12345 for B, and you will get the message that division is not possible since 12345 and the modulus 30000 have a gcd of 15.

The program can also be used to illustrate a generalisation of Fermat's theorem, Euler's theorem. For any number n relatively prime to the modulus m, $n^{\phi(m)} = 1 \pmod m$, where $\phi(m)$ is the Euler number of m. The numbers n and m can be tested to see if they are relatively prime (by attempting to find 1/n mod m as in the last paragraph), and then the Euler number of n can be found - using the program Euler if necessary. For example, the Euler number of 30,000 is 8000. You can check that $11111^{8000} = 1 \pmod{30000}$. But $12345^{8000} \pmod{30000}$ need not be (and is not) 1 since 12345 and 30000 are not co-prime - a glance shows that they have at least a common factor of 5 and as we saw above their gcd is 15.

Before using the program to illustrate trap-door ciphers next month, we can use it to explain two simpler ciphers - and to see their disadvantages. The main aim is not to explain about ciphers - for that, see Bernard Hill's article (Workshop, BEEBUG Vol.9 No.8) - but to illustrate modular arithmetic.

Here is the first method. Suppose the message to be sent is represented by the code 343434343 and we choose a prime number greater than it, say 999999937, as modulus. We can use the program to find the inverse of 343434343 (mod 999999937) which is 701762700. 701762700 is our encoded message. If we receive it and calculate the inverse of 701762700 (mod 999999937) we recover our original number, 343434343. Just as 1/4 is 0.25 and 1/0.25 is 4, so finding the inverse of the inverse recovers our original number.

```
                        MODULAR  ARITHMETIC

This program calculates      A + B  (mod m),
                          or A - B  (mod m),
                          or A x B  (mod m),
                          or A / B  (mod m)  if B relatively prime to m,
                                 B
                          or   A    (mod m),

where m ranges from 2 to 1000000000, and A and B range from 1 to m.
                                          (Be patient with big moduli and powers!)

What is m? 30000
What is A? 1
What is B? 12345

  +  -  x  /  or  ^ ?

Operator? /

12345 and 30000 are not relatively prime (their gcd is 15),
and so there is no multiplicative inverse and division cannot be defined.
```

The problem with division if B and m are not relatively prime

This first method has the obvious disadvantage that the deciphering algorithm is the same as the enciphering one.

In the second method, we can at least make the two algorithms different. To give a trivial example, if someone took any number, say 12 and multiplied it by 3, we would receive 36, multiply it by the reciprocal of 3, 1/3, and we would have 12 x 3 x 1/3 which is 12 again. As an analogy we can find a pair of reciprocals modulo a prime and use one for enciphering and one for deciphering. For example if, continuing to work modulo 999999937 for convenience, we choose any number, say, 463576545, we can find its reciprocal modulo 999999937 which our program shows to be 890790152.

To encipher a number such as 222222222 we can now multiply it by 463576545, using the 'x' operator to find 387054998. If we were to receive 387054998 we could recover our original number simply by multiplying by 890790152. Try it and you will find that:

```
222222222*463576545
```

is:

```
387054998 (mod 999999937)
```

and:

```
387054998*890790152
```

is:

```
222222222 (mod 999999937).
```

But again, though we have different algorithms for enciphering and deciphering, it would not take the Gnomes of Cheltenham long to find them - especially if they intercepted the courier who took the enciphering number 46357645 to the sending station.

Next time we will make use of the programs of the last three articles to model the RSA cipher where the enciphering algorithm can be made public - no courier need visit the sender. Here is an example:

Enciphering:

```
666666666^254665717=304117466
          (mod 796652189)
```

Now that you know the enciphering algorithm involves raising to the power of 254665717 (mod 796652189), can you work out how to decipher the number received as 304117466 back to 666666666? The answer can be given away: 304117466 is now raised to the power of 161722333 - try it!

But there is still the question: How could anyone work out that this was the deciphering algorithm without being told the magic number 161722333? Perhaps the reader can anticipate this if he can factorise 796652189 and find, modulo the Euler number of 796652189 the inverse of 254665717.

PROGRAM OUTLINE

This program outline is rather long, and there is nothing to stop the reader using the program to explore modular arithmetic while ignoring the following details.

ADDITION AND SUBTRACTION

Addition and subtraction are trivial. The MOD function of Basic is used. They are only included in the program for completeness.

MULTIPLICATION

Multiplication can be done using the multiplication routine of Basic followed by MOD m, but *only* if the product A*B does not exceed 2^31-1. The procedure PROCProduct avoids this restriction. One of the two numbers to be multiplied together is in effect represented as an array of coefficients D%(Z%). For the moment let us assume they are binary coefficients. Powers of two are generated and those for which D%(Z%) is 1 are multiplied by the second number and added together. There is no calculation if D%(Z%) is zero. These products of the second number multiplied by the powers of 2 selected when D%(Z%) is 1 could be then summed (mod m).

However, the intermediate products formed can easily exceed the range for integer variables. Also, though powers of 2 are needed if the modulus is very large (above INT((2^31-1)/2) or 2^30-1), it is quicker to use powers of larger integers than 2 so that the number of terms is smaller. The time taken by PROCProduct matters when powers are

being calculated as it is repeatedly called by PROCIndex. Hence, instead of 2, an integer x (W% in the program) is used where x = INT((2^31-1)/modulus). What has to be evaluated is now a polynomial of the form:

```
A0+A1*x+A2*x^2+A3*x^3+ ... +AN*x^N
```

In the program the modulus is limited to 1,000,000,000. It could be increased to INT(2^31-1)/2 = 2^30 = 1,073,741,824. If the modulus is less than this but more than 2^29, then x must be no more than 2, the calculation is very slow, but no higher integer than 2 can be used or intermediate products may exceed 2^31-1.

But for lower values of the modulus, time is saved by having a *larger* x and *fewer* terms in the polynomial. One might expect this to lead to very much shorter calculation times, but MOD itself takes a time dependent on the size of coefficients being multiplied, and so there is not as much gain in calculation speed as one might expect.

The polynomial is evaluated by a standard method which is not to work out each term and add. It is quickest to give a simple numerical example. If the polynomial were, x^4 + 5*x^3 + 7*x2 + 3*x + 6, and the second number were 19, the polynomial could be evaluated as:

```
((((x + 5)x + 7)x + 3)x + 6)
```

and similarly the product with 19 could similarly be evaluated as:

```
((((19x + 5)19x + 7)19x +3)19x + 6)19
```

Here, each of the multiplications takes place (mod x) and so in the worst case when the modulus is more that 2^30-1 and x is 2, the intermediate product never exceeds twice the modulus, 2,000,000,000 which is safely under 2^31-1. For other x the intermediate product never exceeds x*modulus - and as x is (2^31-1)/modulus, any intermediate product never exceeds 2^31-1.

DIVISION

Division, as we saw last month, is only possible if the quotient is relatively prime to the modulus. To divide A by B, we first of all try to calculate the inverse of B (mod m) by calling PROCInverse. An extension of Euclid's algorithm is used. It has been derived from a matrix method of finding the inverse given on page 14 of Humphreys and Prest (1989). It either finds that the gcd of B and m is *not* 1 - in which case it prints the gcd together with a message that division is not possible - or it finds that the gcd *is* 1 and it successfully finds the inverse of B which is then multiplied by A using PROCProduct.

POWERS

When raising one number to the power of another, huge numbers are soon reached. PROCIndex prevents this happening. To find A^B (mod m) the method is to write the exponent B as a sum of powers of two. Since the exponent B is represented as a sum of terms, A^B can be written as a product of terms. Each of these terms can be separately found by repeatedly squaring A (mod m) and multiplying the terms together (mod m). For example, 7^25 may be written as:

```
7^(1*16+1*8+0*4+0*2+1*1)
    = 7^16*7^8*7*1
```

Thus, in this example, 7 can be repeatedly squared (mod m) and terms for the product selected according to the values of the binary coefficients of the exponent 25. Since the coefficients are used as they are generated, there is no need to store them in an array and they are temporarily

Continued on page 52

Volume Controller

Al Harwood shows how to implement a variable volume control on your computer - all done by software.

On the whole, it is easy enough to turn the volume on or off on your computer, using the *FX210 command, and many programs incorporate this facility. It would often be nice to have more flexibility, i.e. to have a variable volume control facility.

This facility is provided by the program listed here. Type the program in and run it - it assembles the code which should be saved with the name *Volume* as prompted. Subsequently the code can be installed by typing *Volume. Once the routine has been installed, pressing Ctrl-@ allows the sound volume to be increased or decreased by using the cursor up and cursor down keys respectively. Once the correct volume has been achieved, pressing Return exits from the routine.

HOW IT'S ALL DONE

To implement a variable volume control we have to be able to change each sound command to the level we require. This is surprisingly easy to do, as sound commands are basically glorified Osword commands. Osword commands indirect through a vector called *Wordv*, and can therefore be intercepted. By changing the value of *Wordv* to the address of our intercepting routine we intercept each Osword command. Next we check for the *make-a-sound* Osword command, and if the sound is too loud we quieten it. This is what the *check* routine in the program does. It gets a bit more complicated where sound envelopes are concerned, but these are also catered for.

We need a way of altering the maximum allowable volume. This is done in the listing within the routine *keychk*, so named because it first checks for a Ctrl-@ key press, and then allows the volume to be changed. This routine intercepts the operating system routine *Remv*, which is used each time an item is removed from a buffer. Our routine is only interested in removals from the keyboard buffer; therefore it is called each time a Basic GET or INPUT command is issued. When Ctrl-@ is pressed, it continues by displaying the current volume level in the top left corner of the screen. By using the up and down cursor keys this can be altered - press Return when completed.

The listing, when assembled is stored in pages &900 and &A00, but the start page of this can be altered in line 140. Also note that the zero page locations &70 and &71 are used and therefore are liable to be corrupted if another program should happen to use these same locations. You may notice the volume controller does not change the volume of the Bell (the sound when VDU7 is issued in), this is because it does not go through Osword, but the bell's amplitude can be altered using the *FX212 command.

The volume controller was designed for use within other programs. The Ctrl-@ key presses will be detected whenever the computer is waiting for a key press, i.e. on a game title page, or while in the Basic programming environment.

Summary of controls:

Ctrl-@	allow volume to be changed
Up/Down Cursor Keys	change the volume
Return	exit from the routine

```
 10 REM Program Volume Control
 20 REM Version B1.0
 30 REM Author  Al Harwood
 40 REM BEEBUG  June 1991
 50 REM Program subject to copyright
 60 :
100 wordv=&20C:remv=&22C
110 oswrch=&FFEE:osbyte=&FFF4
120 envs=&8C0:v=&70
130 FORpass=0TO3STEP3
140 P%=&900
150 [OPTpass
160 .setup LDAwordv:STAnope+2
170 LDAwordv+1:STAnope+3
180 LDAremv:STAre0+1:STAre1+1
190 LDAremv+1:STAre0+2:STAre1+2
200 LDA#keychk MOD256:STAremv
210 LDA#keychk DIV256:STAremv+1
220 LDA#check MOD256:STAwordv
230 LDA#check DIV256:STAwordv+1:RTS
240 .vol EQUB&F1
250 .check PHP:CMP#7:BNEnope
260 STXv:STYv+1
270 LDYy:LDAala:BEQjmp:STAenvs,Y
280 .jmp INY:LDAald:BEQjmp1
290 STAenvs,Y:.jmp1
300 LDA#0:STAala:STAald
310 LDY#2:LDAvol:BNEnt0
320 STA(v),Y:INY:STA(v),Y:JMPnope2
330 .nt0 LDA(v),Y:BEQnope1
340 CMP#&F0:BCCenv:CMPvol:BCSnope1
350 .nope2 LDAvol:STA(v),Y
360 .nope1 LDYv+1:LDA#7
370 .nope PLP:JMP&FFFF
380 .env SEC:SBC#1
390 ASLA:ASLA:ASLA:ASLA
400 CLC:ADC#11:STAy:TAY
410 LDA#&FF:SEC:SBCvol
420 CLC:ADC#2:ASLA:ASLA:ASLA:STAt
430 LDAenvs,Y:CMPt:BCCjp:STAala
440 LDAt:STAenvs,Y
450 .jp INY:LDAenvs,Y:CMPt:BCCjp1
460 .jp1 STAald:LDAt:STAenvs,Y
470 JMPnope1
480 .ala BRK:.ald BRK
490 .y NOP:.t NOP
500 .jout JMPout
510 .keychk
520 PHP:BVSjout:CPX#0:BNEjout
530 LDA#255:BITn64:.re0 JSR&FFFF
540 CMP#0:BNEjout
550 LDA#134:JSRosbyte
560 STXxpos:STYypos
570 LDY#0:.lp LDAscr,Y:JSRoswrch
580 INY:CPY#21:BNElp
590 .lp0 LDA#8:JSRoswrch:JSRoswrch
600 LDA#255:SEC:SBCvol
610 TAY:INY:LDA#48:CPY#10:BCCless
620 TYA:SEC:SBC#10:TAY:LDA#49
630 .less JSRoswrch
640 TYA:CLC:ADC#48:JSRoswrch
650 LDY#0:LDX#0:.lp1 NOP:NOP
660 DEX:BNElp1:DEY:BNElp1
670 OPT FNinkey(-58):BEQninc
680 LDAvol:CMP#&F1:BEQninc:DECvol
690 .ninc OPT FNinkey(-42):BEQndec
700 LDAvol:CMP#0:BEQndec:INCvol
710 .ndec OPT FNinkey(-74):BEQlp0
720 LDA#31:JSRoswrch
730 LDAxpos:JSRoswrch
740 LDAypos:JSRoswrch
750 LDA#15:LDX#1:JSRosbyte
760 LDX#0:.out PLP:.rel JMP&FFFF
770 .n64 EQUB64
780 .scr EQUB31:EQUW0:EQUS"| 00 |"
790 EQUB31:EQUB0:EQUB1:EQUS"======"
800 EQUB31:EQUB4:EQUB0
810 .xpos NOP:.ypos NOP
820 ]NEXT
830 PRINT''"Type  *SAVE Volume "+STR$~
setup+" "+STR$~P%'"Or use  CALL &"+STR$~
setup+"  to install now."'
840 END
850 :
860 DEF FNinkey(A)
870 [OPTpass:LDY#255
880 LDX#A+256:LDA#129
890 JSRosbyte:TYA:]:=pass
```

BEEBUG WORKSHOP

Numerical Integration

by Bernard Hill

This month we turn to the slightly more mathematical topic of numerical integration, or to put it simply, finding the area under a particular curve. This has a great many applications in mathematics and science. In fact, I remember one particular problem from my research past where I had to solve 50 equations in 50 unknowns (see last month) where each of the 2500 coefficients was in fact a particular integral, i.e. an area under a curve!

But let us begin with a concrete example of a function which we are going to 'integrate' or find the area underneath. I am going to use the function:

```
DEF  FNf(x)=EXP(x/5)
```

where we require to find the area from x=a to x=b. Those of you who have a knowledge of elementary calculus will know that for this very simple function the correct answer is 5*(EXP(b/5)-EXP(a/5)), and we use this below to compare the different methods. In general, there is no way of determining a formula for the answer to an area problem, so we have to use the numerical methods below.

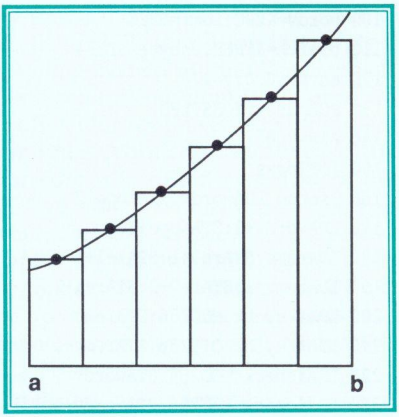


Figure 1

One approach is to divide the interval into a number of rectangles whose heights lie on the curve (see figure 1). Suppose there are N rectangles; then each has a width of (b-a)/N, and the height of the nth is:

```
FNf(a+(n-1)h+h/2)
```

i.e.:

```
FNf(a+(n-0.5)*h)
```

Thus we could define a function FNrect(a,b,N) which evaluates the approximation to the area under the curve as the sum of N rectangles:

```
10000 DEF FNrect(a,b,N)
10010 LOCAL sum,n,h
10020 h=(b-a)/N
10030 FOR n=1 TO N
10040 sum=sum+FNf(a+(n-0.5)*h)
10050 NEXT n
10060 =sum*h
```

(as usual we don't need to initialise *sum* to zero as the LOCAL statement does that for us).

This method relies upon using a large value for N; in fact, the following table gives the error between the true value of this particular FNf(x) using limits of a=-4 and b=4 for the following values of N:

N	Value found	Error
10	8.87159376	-0.009466063
100	8.88096509	-0.000094730
1000	8.88105886	-0.000000957
10000	8.88105981	-0.000000015

Another possible method is the *Trapezoidal Rule*, where we draw a sequence of trapeziums under the curve and find the area of that (see figure 2).

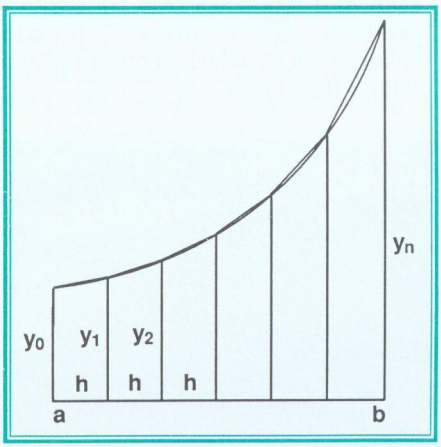


Figure 2

Now each trapezium has an area which is the average height multiplied by the width of the base, or in total:

```
(y0+y1)*h/2 +
    (y1+y2)*h/2 +
            (y2+y3)*h/2 +  ...
```

and collecting and factorising gives:

```
(0.5*y0 + y1 + y2 + y3 + ...
                    + 0.5*yn)*h
```

The following routine handles this:

```
11000 DEF FNtrap(a,b,N)
11010 LOCAL n,h,sum
11020 h=(b-a)/N
11030 FOR n=1 TO N-1
11040 sum=sum+FNf(a+n*h)
11050 NEXT n
11060 sum=sum+0.5*(FNf(a)+FNf(b))
11070 =sum*h
```

This gives a set of evaluations similar to before:

N	Value found	Error
10	8.89999801	+0.018938183
100	8.88124927	+0.000189450
1000	8.88106169	+0.000001870
10000	8.88105981	+0.000000015

In fact, for N>10 these are about twice as inaccurate as the previous results, but in the opposite direction on each line.

We can improve on them both by using them in combination. For the example in figure 3, the trapezium rule acting on the panel shown calculates an area too large, and the rectangle has an area too small. So by taking the average of the two would be an improvement, and mathematically it can be shown, just as here, that we should take a weighted average of 2:1 for the rectangle and trapezium methods:

```
DEF FNbetter(a,b,N)
=(2*FNrect(a,b,N/2)+FNtrap(a,b,N/2))/3
```

Note that in order to be fair we must do half our number of intervals in each

function so that we still do N calculations of the function overall. This method is called *Simpson's One-Third Rule*, and those of you who did this at school may recall that this method is the same as fitting a quadratic curve through every set of three points and finding the area under that.

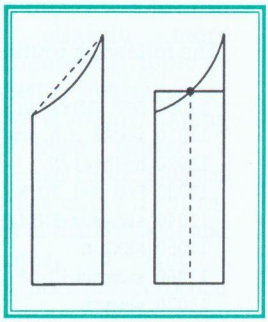


Figure 3

It gives error values as follows:

N	Value found	Error
10	8.88109206	+0.000032239
100	8.88105983	+0.000000007
1000	8.88105982	0.000000000
10000	8.88105981	-0.000000011

This is much better! In fact we find that algorithm is so good that the error of adding up 10,000 values is larger than the error of the method itself. On most computers, taking values of N over about 100 does not usually give better results for Simpson's Rule simply because of the errors in adding a large number of values together, but with the high accuracy which BBC Basic affords we continue with an improvement up to about N=1000.

One possible drawback of Simpson's rule is that N must be an even number. If N is odd, we can modify the rule slightly by calculating the area of the first three panels by *Simpson's Three-Eighths Rule* and then using the rest as the normal one-third rule. DEF FNsimp(a,b,N%) in lines 2000-2110 of listing 1 show us the code of such a composite Simpson's rule fully worked out.

ADAPTIVE INTEGRATION

Now in the case above we knew that the answer was 5*(EXP(0.8)-EXP(-0.8)) = 8.88105982 but how do we know how good our answer is when we don't know what it should be?

The answer is to try two methods and compare them. If we were to evaluate the area twice as (say) FNsimp(a,b,5) and FNsimp(a,b,10) and then compare these values, we would be satisfied if the two different answers give values closer than an agreed limit. Should they not be close enough then we could divide the range into two, and try the same method on each half again. In this way we can produce a recursive function FNadapt(a,b,N) which is given in listing 1, lines 1000-1070.

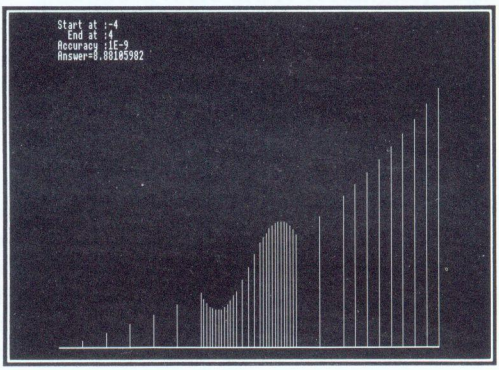


Figure 4. Visual representation of recursive evaluation of area under curve

Running the program in listing 1 not only evaluates the answer, but demonstrates visually this recursive subdivision of the interval into smaller

and smaller sub-intervals at places where the curve becomes more involved. Try running this program with a=-4, b=4 and an accuracy of 1E-9. The function FNf(x) has been replaced by that of line 3000 to produce a more complex function which shows how the method homes in on areas of rapid change of the graph in order to satisfy itself that the answer is accurate. In fact the answer should be the same as before as the wobble in the middle of the curve adds as much to the area as it takes away (see figure 4).

If you want a version of the program without graphics then delete lines 140, 1020 and all the lines from 4000 onwards.

Listing 1

```
   10 REM Adaptive Integration
   20 REM Version  B1.0
   30 REM Author   Bernard Hill
   40 REM Beebug   June 1991
   50 REM Program subject to copyright
   60 REM :
  100 MODE0
  110 INPUT"Start at :"xmin
  120 INPUT"  End at :"xmax
  130 INPUT"Accuracy :"accuracy
  140 PROCrange
  150 answer=FNadapt(xmin,xmax)
  160 PRINT"Answer="answer
  170 END
  180 :
 1000 DEF FNadapt(a,b)
 1010 LOCAL A1,A2
 1020 PROCplot(a,b)
 1030 A1=FNsimp(a,b,10)
 1040 A2=FNsimp(a,b,5)
 1050 IF ABS(A1-A2)<accuracy THEN =A1
 1060 REM otherwise subdivide again
 1070 =FNadapt(a,(a+b)/2)+FNadapt((a+b)/
2,b)
 1080 :
 2000 DEF FNsimp(a,b,N%)
 2010 LOCAL n%,m%,x,sum,h
 2020 h=(b-a)/N%
 2030 IF N%=0 THEN =0
 2040 IF N%=1 THEN =(FNf(a)+FNf(b))/2
 2050 IF N% MOD 2=1 THEN =0.375*h*(FNf(a
)+3*FNf(a+h)+3*FNf(a+2*h)+FNf(a+3*h))+FN
simp(a+3*h,b,N%-3)
 2060 FOR n%=1 TO N%-1
 2070 x=a+n%*h
 2080 IF n% MOD 2=0 THEN m%=2 ELSE m%=4
 2090 sum=sum+m%*FNf(x)
 2100 NEXT n%
 2110 =h/3*(FNf(a)+sum+FNf(b))
 2120 :
 3000 DEF FNf(x)=EXP(x/5)+(ABSx<1)*SINx*
(x*x-1)/2
 3010 :
 4000 DEF PROCplot(a,b)
 4010 MOVE FNx(a),FNy(FNf(a))
 4020 DRAW FNx(a),FNy(ymin)
 4030 DRAW FNx(b),FNy(ymin)
 4040 DRAW FNx(b),FNy(FNf(b))
 4050 ENDPROC
 4060 :
 5000 DEF FNx(x)=xs*(x-xmin)
 6000 DEF FNy(y)=ys*(y-ymin)+10
 7000 DEF FNmax(x,y):IF x<y THEN =y ELSE
=x
 8000 DEF FNmin(x,y):IF x>y THEN =y ELSE
=x
 8010 :
 9000 DEF PROCrange
 9010 LOCAL y,max,min:min=1E30:max=-min
 9020 FOR x=xmin TO xmax STEP (xmax-xmin
)/10
 9030 y=FNf(x)
 9040 max=FNmax(max,y)
 9050 min=FNmin(min,y)
 9060 NEXT
 9070 ys=800/(max-min)
 9080 ymin=min
 9090 xs=1200/(xmax-xmin)
 9100 ENDPROC
```

Merge, Part-merge and Part-save

by Jagdish Sah

The User Guide describes two methods of merging a Basic program from a disc file into a program in memory. The first is based on *SPOOLing the program to be merged out to a text file, which is then *EXECed into the current program in memory. Although it achieves the objective adequately, the method is somewhat long-winded.

The second method involves *LOADing the file to be merged at the end of the program in memory, and so can only be used to append one program to the end of another.

What is needed is a utility which will extract the required lines from a program file and merge them into the program in memory without going through the rigmarole described above. The program in Listing 1 assembles a machine code utility which does just that, providing a new command, *MERGE to do the job. As an added bonus, it also provides a second command, *PSAVE, which will save any part of the program in memory.

Type in the listing carefully and save it as MergSrc before running it. When run, it will assemble the code and save it automatically as MergeMC.

USING THE UTILITY

To install the utility, type *MergeMC, and the program will be loaded from disc. The commands *MERGE and *PSAVE are now ready to use. Note that they must be upper case. The first command has the syntax:

```
*MERGE <filename>,<startline>,<endline>
```

It will extract all lines between the startline and endline (inclusive) from the specified file on disc and merge them into the program in memory. New lines from the file will replace the old ones in memory if they have the same line numbers. For example:

```
*MERGE MyProg,1000,1999
```

will merge all lines between 1000 and 1999 from the file MyProg.

```
*MERGE MyProg,0,1999
```

will merge all lines from the start of the file up to 1999.

```
*MERGE MyProg,0,0
```

will merge the whole file. The routine replaces the second 0 with 32767, which is the largest possible line number.

The *PSAVE command has the following syntax:

```
*PSAVE <filename>,<startline>,<endline>
```

This will save all lines from startline to endline (inclusive) to the named file. The line numbers have the same meaning as in the *MERGE command, but this time they refer to the program in memory.

If Escape is pressed during the merge or partsave operation, everything is tidied up before exiting from the routine. When merging, this means inserting the current line properly, so that the program in memory is not corrupted, before closing the file. With partsave, the current line together with the end-of-program marker is sent to the file before the latter is closed.

ERROR MESSAGES

The program also detects and reports on any errors that may arise. The routine first checks whether a valid program is present at PAGE. If not, a "Bad program" message is displayed. So if you want to use *MERGE to effect a part-load prior to typing in a program, remember to issue a NEW command first.

If an invalid line number is specified, a "Bad Arg" message results. The message "Can't open file" means that the specified file was not found on the disc. For the partsave operation it may mean that the disc or catalogue is full.

PROGRAM NOTES

The machine code requires slightly more than two pages of memory, and so it has been arranged to start at &8C0. This means that it will overwrite the sound envelope workspace, but it was felt that this was preferable to using part of page &B, which provides function key workspace on the model B and Econet workspace on the Master. If you find &8C0 inconvenient you may alter the value in line 100.

The two new commands will work in immediate mode only (i.e. from the command line, not from within programs). The code is assembled with reference to the version of Basic in your machine (though Basic I is not catered for), so it cannot be transferred to a computer with a different version without being first re-assembled on that machine.

This program was first published in BEEBUG Vol.5 No.6, but as it provides some very useful functions we are repeating it for the benefit of newer readers.

```
   10 REM Program >Merge
   20 REM Version B0.17
   30 REM Author  Jagdish Sah
   40 REM BEEBUG  November 1986
   50 REM         June 1991
   60 REM Program subject to copyright
   70 :
  100 code=&8C0
  110 PROCassemble:PROCchecksum
  120 IF S%<>chksum%:PRINT'"Checksum err
or"':END
  130 OSCLI"Save MergeMC "+STR$~code+" "
+STR$~P%
  140 END
  150 :
 1000 DEF PROCassemble
 1010 REM Syntax:
 1020 REM *MERGE <fsp>,<start>,<end>
 1030 REM *PSAVE <fsp>,<start>,<end>
 1040 osfind=&FFCE:osbget=&FFD7
 1050 osbput=&FFD4:osbyte=&FFF4
 1060 osargs=&FFDA:osfile=&FFDD
 1070 page=&18:cmdptr=&19:ycmd=&1B
 1080 newno=&2A:lineptr=&70:start=&72
 1090 end=&74:handle=&76:savey=&77
 1100 newlen=&78:temp=&79:stack=&7A
 1110 pblk=&7E:anycom=&F8
 1120 IF ?&8008=1 RESTORE 2440 ELSE IF ?
&8008=4 RESTORE 2470 ELSE IF ?&8008=64 R
ESTORE 2500 ELSE IF ?&8008=7 RESTORE 253
0 ELSE PRINT"Unknown version of Basic -
assembly aborted"
 1130 READ insline,chkprog,getnumB
 1140 READ getcharB,basic,chksum%
 1150 :
 1160 FOR opt=0 TO 2 STEP 2
 1170 P%=code:[OPT opt
 1180 LDA &209:CMP #newrtn DIV 256
 1190 BEQ out:STA oldrtn2+2:LDA &208
 1200 STA oldrtn2+1:LDA #newrtn MOD 256
 1210 SEI:STA &208:LDA #newrtn DIV 256
 1220 STA &209:CLI
 1230 .out RTS
 1240 .oldrtn LDX cmdptr
 1250 LDY cmdptr+1:PLA:PLP
 1260 .oldrtn2 JMP 0
```

```
1270 :
1280 .getval:JSR getnumB:BEQ badarg
1290 BMI badarg:BIT &F8:BMI nocom
1300 JSR getcharB:CMP #ASC(",")
1310 BNE badarg
1320 .nocom:LDA newno+2
1330 ORA newno+3:BNE badarg:LDX newno+1
1340 BMI badarg:LDA newno:RTS
1350 .badarg BRK:EQUB &FE
1360 EQUS "Bad Arg":EQUB 0
1370 :
1380 .newrtn:PHP:PHA:STX cmdptr
1390 STY cmdptr+1:JSR chkprog
1400 LDY #0:STY lineptr
1410 .chkcmd LDA cmd,Y:BEQ merge
1420 CMP (cmdptr),Y:BNE chk2
1430 INY:BNE chkcmd
1440 .chk2 LDY #0
1450 .chkcmd2 LDA cmd2,Y
1460 BEQ partsave:CMP (cmdptr),Y
1470 BNE oldrtn:INY:BNE chkcmd2
1480 :
1490 .merge:LDA #&40:JSR init
1500 .merge2 JSR getbyte:CMP #&0D
1510 BNE close:JSR getbyte:STA newno+1
1520 JSR getbyte:STA newno:JSR getbyte
1530 STA newlen:LDA newno:CMP start
1540 LDA newno+1:SBC start+1
1550 BCS merge3:LDX #4
1560 .skipline JSR getbyte:INX
1570 CPX newlen:BNE skipline:BEQ merge2
1580 .merge3 LDA end:CMP newno
1590 LDA end+1:SBC newno+1
1600 BCC close:LDX #4
1610 .getall JSR getbyte
1620 STA &700,X:INX:CPX newlen
1630 BNE getall:LDA #&0D:STA &700,X
1640 JSR setbasic:LDY #4:JSR insline
1650 CLC:BCC merge2
1660 :
1670 .close LDA #0:TAY:JSR osfind
1680 .close2 LDX stack:TXS:JSR setbasic
1690 JMP basic
1700 :
1710 .getbyte:LDY &FF:BMI close
1720 LDY handle:JSR osbget
1730 BCS close:RTS
1740 :
1750 .partsave:LDA #&80:JSR init
1760 .psave2 LDY #2:LDA (lineptr),Y
1770 CMP start:DEY:LDA (lineptr),Y
1780 SBC start+1:BCS psave3
1790 JSR inclineptr:BCC psave2
1800 .psave3 LDA &FF:BMI exitpsave
1810 LDY #2:LDA end:CMP (lineptr),Y
1820 DEY:LDA end+1:SBC (lineptr),Y
1830 BCC exitpsave:LDY #3
1840 LDA (lineptr),Y:STA temp
1850 LDY #0:STY savey
1860 .psave4 JSR putbyte
1870 LDY savey:CPY temp:BNE psave4
1880 JSR inclineptr:BCC psave3
1890 .exitpsave LDA #&0D:JSR putbyte2
1900 LDA #&FF:JSR putbyte2:LDA #0:TAY
1910 JSR osfind:LDA #0:TAY:JSR osargs
1920 CMP #4:BNE exitps2
1930 LDX #pblk MOD 256
1940 LDY #pblk DIV 256
1950 LDA #1:JSR osfile
1960 .exitps2 JMP close2
1970 :
1980 .inclineptr LDY #3:LDA (lineptr),Y
1990 CLC:ADC lineptr:STA lineptr
2000 BCC inclineptr2:INC lineptr+1
2010 .inclineptr2 CLC:RTS
2020 :
2030 .putbyte:LDY savey:INC savey
2040 LDA (lineptr),Y
2050 .putbyte2 LDY handle:JMP osbput
2060 :
2070 .setbasic:LDA #&BB:LDX #0:LDY #&FF
2080 JSR osbyte:TXA:TAY:LDA #&97
2090 LDX #&30:JMP osbyte
2100 :
2110 .init:PHA:STY ycmd:LDY#0
2120 STY anycom:LDA ycmd:CLC:ADC cmdptr
2130 TAX:STX pblk:LDA cmdptr+1:ADC #0
2140 TAY:STY pblk+1:LDY ycmd
2150 .retlp LDA(cmdptr),Y:INY
2160 CMP#ASC(","):BNE retlp:LDA#13
2170 STY ycmd:DEY:STA (cmdptr),Y
2180 LDY pblk+1:PLA:JSR osfind
```

```
 2190 TAX:BNE opened:BRK:EQUB 200
 2200 EQUS "Can't open file":EQUB 0
 2210 .opened STA handle:LDY ycmd
 2220 JSR getval:STA start:STX start+1
 2230 DEC anycom:JSR getval:STA end
 2240 STX end+1:ORA end+1:BNE notzero
 2250 LDA #&FF:STA end
 2260 LDX #&7F:STX end+1
 2270 .notzero LDA page:STA lineptr+1
 2280 TSX:DEX:DEX:DEX:DEX:STX stack
 2290 LDX #7
 2300 .init2 LDA addr,X:STA &38C,X
 2310 STA pblk+2,X:LDA #0:STA pblk+10,X
 2320 DEX:BPL init2:RTS
 2330 :
 2340 .addr EQUD &FFFF1900
 2350 EQUD &FFFF8023
 2360 .cmd EQUS "*MERGE ":EQUB 0
 2370 .cmd2 EQUS "*PSAVE ":EQUB 0
 2380 ]:NEXT:ENDPROC
```

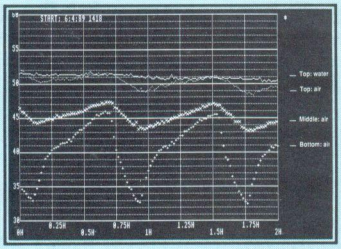
```
 2390 :
 2400 DEF PROCchecksum
 2410 S%=0:FOR J%=code TO P%-1:S%=S%+?J%
 2420 NEXT:ENDPROC
 2430 REM Basic II entry points and chec
ksum value
 2440 DATA &BC8D,&BE6F,&9B29
 2450 DATA &8A8C,&8AF3,59754
 2460 REM Basic IV entry points and chec
ksum value
 2470 DATA &BB15,&BDE5,&9D3B
 2480 DATA &8ED5,&8F83,59740
 2490 REM Basic VI entry points and chec
ksum value
 2500 DATA &BA65,&BD45,&9D1B
 2510 DATA &8EAE,&8F64,59557
 2520 REM Basic VII entry points and che
cksum value
 2530 DATA &BB45,&BE25,&9DFF
 2540 DATA &8F92,&9048,59669
```

Route Planner Revisited (continued from page 17)

```
 1500 DEF PROCsave
 1510 FOR count%=1 TO LEN(text$)
 1520 BPUT#ch%,ASC(MID$(text$,count%,1))
 1530 NEXT
 1540 BPUT#ch%,&0D
 1550 ENDPROC
 1560 :
 1570 DEF PROCfileroute
 1580 REPEAT
 1590 PROCentry
 1600 PROCsave
 1610 UNTIL text1$="Q"
 1620 ENDPROC
 1630 :
 1640 DEF PROCentry
 1650 CLS
 1660 IF ans$(1)="M" THEN units$="ins" E
LSE units$="cm"
 1670 PRINTTAB(0,0)"Length of leg ("unit
s$") or Q to finish "
 1680 INPUTTAB(0,1);text1$
 1690 IF text1$="Q" text$="Q":ENDPROC
```

```
 1700 PRINTTAB(0,2)"Road name or number"
 1710 INPUTTAB(0,3);text2$
 1720 PRINTTAB(0,4)"End of leg (Place or
 junction)"
 1730 INPUTTAB(0,5);text3$
 1740 PRINTTAB(0,6)"Type of road (1-4)"
 1750 INPUTTAB(0,7);text4$
 1760 text$=text1$+"/"+text2$
 1770 IF text2$<>""text$=text$+" "
 1780 text$=text$+text3$+"/"+text4$
 1790 PRINTTAB(5,9)text$
 1800 PRINTTAB(5,11)"All OK (Y/N)?"
 1810 REPEAT:rep$=GET$:UNTIL INSTR("YyNn
",rep$)>0
 1820 IF INSTR("Nn",rep$)>0 GOTO 1650
 1830 ENDPROC
 1840 :
 1850 DEF PROCclose
 1860 VDU12,26
 1870 CLOSE#ch%
 1880 PRINTTAB(5,5)"All done"
 1890 ENDPROC
```

Best of BEEBUG Software



Applications II Disc

Crossword Editor - for designing, editing and solving crosswords
Monthly Desk Diary - a month-to-view calendar which can also be printed
3D Landscapes - generates three dimensional landscapes
Real Time Clock - a real time digital alarm clock displayed on the screen
Running Four Temperatures - calibrates and plots up to four temperatures
Julia Sets - fascinating extensions of the Mandelbrot set
Foreign Language Tester - foreign character definer and language tester
Label Processor - for designing and printing labels on Epson compatible printers
Share Investor - assists decision making when buying and selling shares.

General Utilities Disc

Printer Buffer **(Master series only)**
Sprite Editor/Animator
Mode 7 Screen Editor
Epson Character Definer
ROM Filing System Generator
Multi-Column Printing
Multi-Character Printer Driver for View
ROM Controller
BEEBUG MiniWimp **(Requires sideways RAM)**

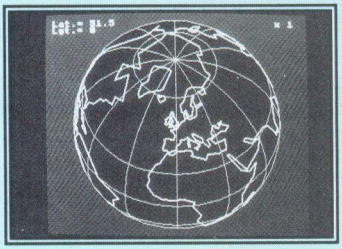
EDIKIT ROM

An indispensible utility ROM for all Basic programmers, containing the following commands:

***FTEXT** (find text) ***FBASIC** (find Basic) ***FPROCFN** (find proc/function)
***LPROC** (list proc) ***LFN** (list function) ***LFROM** (list 8 lines of program)
***RTEXT** (replace text) ***RBASIC** (replace Basic) ***SYSINF** (system info)
***VARLIST** (list program variables) ***FKDEFS** (function key definitions)

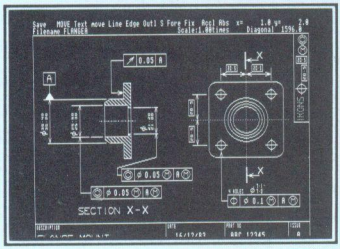
Incorporating the updated Basic Booster utilities:

SUPER SQUEEZE PARTIAL RENUMBER PROGRAM LISTER
RESEQUENCER SMART RENUMBER TEXTLOAD AND TEXTSAVE



Applications I Disc

Business Graphics - for producing graphs, charts and diagrams
Video Cataloguer - catalogue and print labels for your video cassettes
Phone Book - an on-screen telephone book which can be easily edited and updated
Personalised Letter-Headings - design a stylish logo for your letter heads
Appointments Diary - a computerised appointments diary
Mapping the British Isles - draw a map of the British Isles at any size
Selective Breeding - a superb graphical display of selective breeding of insects
Personalised Address Book - on-screen address and phone book
The Earth from Space - draw a picture of the Earth as seen from any point in space
Page Designer - a page-making package for Epson compatible printers
World by Night and Day - a display of the world showing night and day for any time and date of the year



ASTAAD

Enhanced ASTAAD CAD program for the Master, offering the following features:

- full mouse and joystick control
- built-in printer dump
- speed improvement
- STEAMS image manipulator
- Keystrips for ASTAAD and STEAMS
- Comprehensive user guide
- Sample picture files

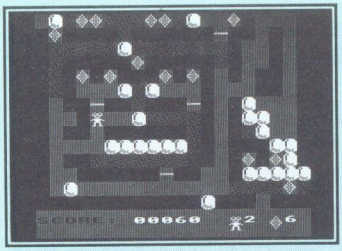
	Stock Code	Price		Stock Code	Price
ASTAAD (80 track DFS)	1407a	£5.95	**ASTAAD** (3.5" ADFS)	1408a	£5.95
EDIKIT (EPROM)	1451a	£7.75			
EDIKIT (40/80T DFS)	1450a	£5.75	**EDIKIT** (3.5" ADFS)	1452a	£5.75
Applications II (80 track DFS)	1411a	£4.00	**Applications II** (3.5" ADFS)	1412a	£4.00
Applications I Disc (40/80T DFS)	1404a	£4.00	**Applications I Disc** (3.5" ADFS)	1409a	£4.00
General Utilities Disc (40/80T DFS)	1405a	£4.00	**General Utilities Disc** (3.5" ADFS)	1413a	£4.00

Please add p&p

BEEBUG Ltd, 117 Hatfield Road, St Albans, Herts AL1 4JS.

Best of BEEBUG Software

Arcade Games



George and the Dragon - Rescue 'Hideous Hilda' from the flames of the dragon, but beware the flying arrows and the moving holes on the floor.

Ebony Castle - You, the leader of a secret band, have been captured and thrown in the dungeons of the infamous Ebony Castle. Can you escape back to the countryside, fighting off the deadly spiders on the way and collecting the keys necessary to unlock the coloured doors?

Knight Quest - You are a Knight on a quest to find the lost crown, hidden deep in the ruins of a weird castle inhabited by dangerous monsters and protected by a greedy guardian.

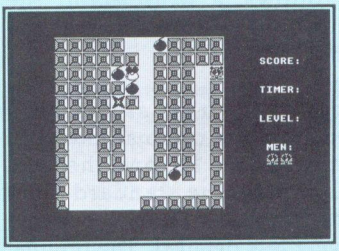
Pitfall Pete - Collect all the diamonds on the screen, but try not to trap yourself when you dislodge the many boulders on your way.

Builder Bob - Bob is trapped on the bottom of a building that's being demolished. Can you help him build his way out?

Minefield - Find your way through this grid and try to defuse the mines before they explode, but beware the monsters which increasingly hinder your progress.

Manic Mechanic - Try to collect all the spanners and reach the broken-down generator, before the factory freezes up.

Quad - You will have hours of entertainment trying to get all these different shapes to fit.



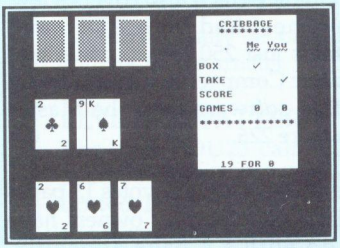
Board Games

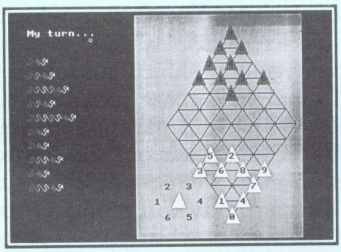
Solitaire - an elegant implementation of this ancient and fascinating one-player game, and a complete solution for those who are unable to find it for themselves.

Roll of Honour - Score as many points as possible by throwing the five dice in this on-screen version of 'Yahtze'.

Patience - a very addictive version of one of the oldest and most popular games of Patience.

Elevenses - another popular version of Patience - lay down cards on the table in three by three grid and start turning them over until they add up to eleven.





Cribbage - an authentic implementation of this very traditional card game for two, where the object is to score points for various combinations and sequences of cards.

Twiddle - a close relative of Sam Lloyd's sliding block puzzle and Rubik's cube, where you have to move numbers round a grid to match a pattern.

Chinese Chequers - a traditional board game for two players, where the object is to move your counters, following a pattern, and occupy the opponent's field.

Aces High - another addictive game of Patience, where the object is to remove the cards from the table and finish with the aces at the head of each column.

	Stock Code	Price		Stock Code	Price
Arcade Games (40/80 track DFS)	PAG1a	£5.95	**Arcade Games** (3.5" ADFS)	PAG2a	£5.95
Board Games (40/80 track DFS)	PBG1a	£5.95	**Board Games** (3.5" ADFS)	PBG2a	£5.95

Please add p&p:

UK:	*£1.00 for the first item and 50p for every additional item*
Europe and Eire:	*£1.60 for the first item and 80p for every additional item*
Elswhere:	*£2.60 for the first item and £1.30 for every additional item*

Tel. (0727) 40303 **Fax. (0727) 860263**

1st course

VDU and FX Calls (3)

by Mike Williams

This month I propose to conclude what I wish to say with regard to VDU calls, but first, by way of an introduction to what follows, a word about the documentation supplied with the different BBC micros.

It is a fact of life that as the Beeb developed from the original model B, through the Master 128 and then the Master Compact, the standard User Guide (or Welcome Guide) became progressively less detailed. The original User Guide contained over 500 pages packed with information, the Master 128 Welcome Guide (which also covers View, Edit, ViewSheet and both ADFS and DFS) contains just 250 pages, while that for the Master Compact (with admittedly less bundled software than the Master 128) has a mere 226.

The model B User Guide devotes 13 pages to VDU calls, yet Compact owners have to make do with a 4 page appendix. Similarly, the User Guide allows 24 pages for FX calls; the Compact Welcome Guide has an 8 page appendix. Of course, Master and Compact users can have recourse to the two volumes (at extra cost) of the Master Series Reference Manual (which I thoroughly recommend). This covers all features of these systems in reasonable detail. However, not all users feel this expenditure can be justified, and are left with the limited coverage of VDU and FX calls already referred to.

What makes this all the more frustrating is that the number of such calls was extended with the operating system for the Master series. All VDU and FX calls are listed in the Welcome Guides, but briefly and thus frustratingly described.

To redress the balance I will end the VDU story by discussing some of those extra VDU calls (for Master and Compact owners) and ask model B users to bear with me this time around. Do note that many of these calls are more technical and complex to use than those discussed hitherto.

In one sense there has been no increase in the number of VDU codes at all; after all these are the ASCII codes in the range 0 to 31. However, the number of options controlled by VDU23 has been increased, and the number of graphics functions controlled either by the Basic PLOT command, or by its VDU25 equivalent (see last month's First Course) has also been extended. I do not feel that it is appropriate to discuss the latter in any detail here, as this would be better done in one or more articles on graphics.

Some of the variants of VDU23 are listed in Table 1. In essence these provide direct control of the 6845 CRTC chip which manages the screen display. Note that all VDU23 calls require a function identifier followed by eight further parameters. Where less than eight parameters are required by the function in question, zeros are added to make the parameter count correct; alternatively (on a Master or Compact) the symbol '|' may be used to represent all remaining zero parameters for any call (see later examples).

The first of the new VDU23 calls (VDU23,0) really is for specialist users, allowing the 18 registers of the 6845 chip to be programmed directly, and has no place in a First Course article.

VDU call	Function
VDU23, 0	Writes to one of the 6845 registers (for advanced users only)
VDU23, 1	Control cursor appearance: VDU23,1,0;0;0;0; turn cursor off VDU23,1,1;0;0;0; turn cursor on
VDU23, 2-5	Define extended colour fill patterns
VDU23, 6	Set dot pattern for dotted lines: VDU23,6,n,0,0,0,0,0,0 sets the dot pattern according to the value of n as an 8 bit binary number
VDU23, 7	Scroll any rectangular area of the screen directly: VDU23,7,m,d,z,0,0,0,0,0 where m is 0 (current text window) or 1 (whole screen); d is in the range 0-7 and indicates the direction of scroll; and z is 0 (scroll by 1 character) or z is 1 (scroll by 1 character vertically but by 1 byte horizontally).
VDU23, 8	Clears block of text to background colour: VDU23,8,t1,t2,x1,y1,x2,y2,0,0 where t1, t2 are in the range 0 to 10
VDU23, 9	Same as *FX9
VDU23,10	Same as *FX10
VDU23,11	Set default ECF patterns
VDU23,12-15	Simple ECF pattern definitions
VDU23,16	Cursor movement control

Table 1. Summary of the more useful VDU23 calls

CURSOR CONTROL

VDU23,1 we have already met, as it is implemented on a model B, and controls the appearance of the flashing cursor on the screen. Note that Master and Compact Users can write these commands in the form:

```
VDU23,1,0|      turn cursor off
VDU23,1,1|      turn cursor on
```

There are two further variants:

```
VDU23,1,2|      make cursor steady
VDU23,1,3|      make cursor flash
                slowly
```

COLOUR FILL PATTERNS

The commands in the range VDU23,2 to VDU23,5 (four in all) are used to define the so-called Extended Colour Fill (ECF) patterns:

```
VDU23,2      set pattern 1
VDU23,3      set pattern 2
VDU23,4      set pattern 3
VDU23,5      set pattern 4
```

There are four further VDU23 calls which enable simpler forms of ECF patterns to be defined:

```
VDU23,12     pattern 1
VDU23,13     pattern 2
VDU23,14     pattern 3
VDU23,15     pattern 4
```

Even so the definition and use of ECF patterns is really beyond the scope of a First Course article.

LINE DOT PATTERN

The next VDU23 call (VDU23,6) is used to define the dot pattern for dotted lines drawn with the Basic PLOT command. It takes the form:

```
VDU23,6,n,0,0,0,0,0,0
```

where the value of 'n' is treated as a binary 8-bit number (from left to right) with a '1' indicating a dot, and a '0' indicating a space. Thus:

```
VDU23,6,255|     (or VDU23,6,&FF|)
```

defines a solid line since 255 (&FF in hex) is 11111111 as a binary number. The default is 170 (&AA hex) which is 10101010 in binary giving a dotted line. Putting n=238 (&EE hex) equivalent to 11101110 produces a dashed line.

SCROLLING WINDOWS

The somewhat complex VDU23,7 call allows any rectangular area of the screen

to be scrolled in any direction, and is thus an obvious target for exploration by games programmers. You probably wondered how screen scrolling could be handled so flexibly; well here's one answer. The syntax for the command is shown in table 1. For example, in mode 1 (four colours):

```
FOR I%=0 TO 1279 STEP 4
VDU23,7,1,0,1|
NEXT
```

would scroll an entire mode 2 screen (4 pixels per step) from left to right. As with normal vertical scrolling, whatever moves off the screen is then lost.

Following VDU23,7 there are three parameters. The first is either 0 (to scroll the current text window), or 1 (to scroll the entire screen). The second parameter (treated simply here) can be 0 (to scroll right), 1 (to scroll left), 2 (to scroll down) or 3 (to scroll up). The third parameter is 0 for character scrolling and 1 for byte scrolling. One application of this call would enable greater variety to be incorporated in clearing text say from a window when displaying instructions for a game.

CLEAR TEXT

The next call (VDU23,8) allows a defined area of the current text window to be cleared to the current text background colour, and in that respect is a fancy (and more involved) form of Basic's CLS command. It takes six parameters:

```
VDU23,8,t1,t2,x1,y1,x2,y2,0,0
```

(plus two zeros to make eight). Values for t1 and t2 (which must each be in the range 0 to 10) determine a base position for the start and end of the rectangular text area to be cleared, while (x1,y1) and (x2,y2) specify displacements from the positions indicated by t1 and t2. Confused? Well, like many VDU23 calls, this one is not simple. For example:

```
VDU23,8,5,5,-4,4,4,-4
```

would clear an area from 4 characters to the left to 4 characters to the right, from 4 characters above to 4 characters below, taking the current position of the cursor (t1=5, t2=5) as the base point (see figure 1).

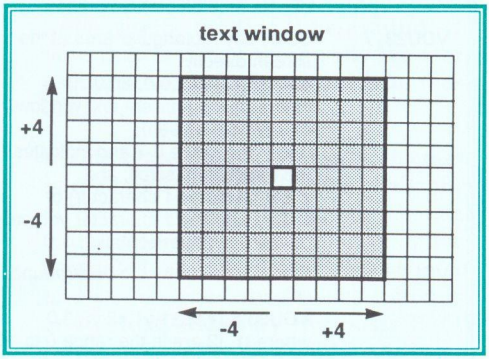


Figure 1. Using VDU23,8 to clear part of a window

If you feel from this brief account that VDU23,8 would be of use to you, then I suggest you refer to the Master Reference manual which gives the full range of values and meanings for the parameters t1 and t2 (but unfortunately no examples). As far as this article is concerned I feel our discussion of this particular call has gone quite far enough.

We've nearly finished our scroll through these extra VDU23 calls, and what is left will not be too taxing. VDU23,9 and VDU23,10 correspond exactly to the more commonly used *FX9 and *FX10 calls for setting the flash times for flashing colours (and we'll deal with these when we deal with FX calls in general). However, it is sometimes more convenient to express a call in VDU format, rather than as a star command, so it may be worth noting these alternatives.

VDU23,11 sets the ECF patterns (see earlier) to their default values, and as we said then, ECF patterns are better dealt with in a separate article on graphics. However, using this command is very simple:

```
VDU23,11|
```

The VDU calls in the range VDU23,12 to VDU23,15 also relate to ECF patterns, and were referred to when we briefly covered VDU23,2 to VDU23,5. That brings us on to VDU23,16.

CURSOR MOVEMENT CONTROL

VDU23,16 provides us with detailed control over the way in which the cursor moves on the screen. The call takes the form:

```
VDU23,16,x,y,0,0,0,0,0
```

where the values of x and y produce effects as we shall see. Normal cursor movement is from left to right, with the cursor moving to the start of the next line down (scrolling if necessary) if moved beyond the right-hand edge of the current text window.

With VDU23,16 cursor movement is controlled by a set of eight bits called flag bits (i.e. a number in the range 0 to 255 decimal, &00 to &FF in hex). Thus each flag is either 0 or 1. When a VDU23,16 call is issued, the current set of flags is replaced by a new eight bit value formed as follows:

```
(<current value> AND y) EOR x)
```

In effect the eight bits forming the number y set or unset the flag bits, while the eight bits in x if 1 (then) switch the state of each flag. Because this is once more quite an involved command, I will deal only with a selection of possibilities illustrated by examples.

```
VDU23,16,0,0|
```

This sets cursor movement to the default state.

```
VDU23,16,&20,0|
```

This affects horizontal cursor movement. Normally when a character is printed, the cursor moves to the right. With this command executed, the cursor does not move horizontally.

```
VDU23,16,&10,0|
```

This one affects vertical movement. Usually when the cursor reaches the foot of the screen, any cursor-down movement causes the screen to scroll vertically. With this VDU23,16 call in effect, the cursor moves to the opposite edge of the screen in the same circumstances.

```
VDU23,16,&01,0|
```

This enables the scroll protect option. Printing a character (at the extreme right and at the foot of the current window) would normally cause the screen to scroll. With this VDU23,16 call in operation, the scroll is suspended until the next character is about to be printed. This provides one solution to the annoying problem that printing a character in the bottom right-hand corner of the screen usually causes the screen to scroll. This call postpones that until the next character is about to be printed.

That's it as far as VDU23 is concerned. As I warned you at the outset, this month's article is not light and easy reading, and I have left out much of the more complex detail even then. Next time we'll be back to the simpler things of life (at least temporarily) when we make a start on those long promised *FX calls.

Wordwise User's Notebook

David Else presents the first of a short series to help users of Computer Concepts' Wordwise and Wordwise Plus word processors. This month he describes how he handles the requirements of a standard letter layout.

The following notes are written for use with Wordwise Plus, a Canon PW-1080A printer and cut sheets. Readers with other printers should amend where appropriate. When using Wordwise (rather than Wordwise Plus) the code 'OC27,' should be used in place of the code 'ES'.

Many computer users who have a requirement for writing letters with their computer set up one or more standard letter layouts for this purpose. An example of one which I use is shown in figure 1. Note that the character '>' is used to indicate the use of the Tab key in setting up the layout. The terms 'f1' and 'f2' are used to indicate the *GREEN* and *WHITE* codes respectively for the start and end of the embedded commands.

```
f1ES64f1ES40f1EPf1EMf1TS0f1BS0f1PL68f1ES51,48f2
f1*FX155,36f1LL63f1LM10f1LNSf1DT43,44,45,46f2

Tel. 0123 456789>                8 Station Road,
Membership No. Z123456>>           Green Town,
Your ref. 010101>>>                  Sussex,
>>>>                                  WA1 3GD.
>                                4th November 1990.
BEEBUG RETAIL,
117 Hatfield Road,
St. Albans,
Hertfordshire,
AL1 4JS.
f1JOf2
Dear Sir,

f1TI8f2 Start of text ....


f1LS3f2
f1CEf2          Yours sincerely,


f1CEf2              David Else
f1ES64f2
```

Figure 1. Example of standard letter layout

The reason for each of the embedded commands used in the layout of figure 1 is as follows:

ES64 Printer reset code. This code is used to reset the printer to its default values. This command should *not* be used if user defined characters are in the printer RAM as these will be erased as well.

ES40 This code sets NLQ, f1ES80f2 sets standard format.

EP Enable Paging. This is required so that the code f1LNSf2 can be used to include the line numbers on the screen in preview mode (option 7).

EM Enable Message. This is required so that printing stops at the end of each page.

TS0 Top Space 0. This is required to enable printing to start at the top of the page (the default value is 6).

BS0 Bottom Space 0. This is required to prevent the default message 'Page' from

being printed at the bottom of each sheet (the default value is 6). An alternative method to prevent 'Page' from being printed is not to include the code f1BS0f2 but to add the code f1DFf2 as the last code on a line of the initial embedded commands.

PL68 Page Length 68. This sets the number of lines on a page. The number (68) is set in conjunction with the number of lines shown on the page in preview mode and the next code. For short letters ensure this number is greater than that shown in preview. For longer letters, the number will govern where one page ends and the next starts (the default value is 66).

ES51,48 Line spacing at 48/216 of an inch. This code sets the distance between lines. Assuming A4 paper (11 inches long) is used, multiply 216 by 10 and divide by the number of lines given on the screen in preview mode. Round this number down to give the number of 216ths of an inch per line to make the text fill the page. If this code, or one of the other line space codes, is not included the line spacing will be 1/6 of an inch.

***FX155,36** This code is used to give white text on a blue background in the preview mode.

LL63 Line Length 63. This sets the number of characters per line from the margin (the default value is 70).

LM10 Left Margin 10. This sets the width of the margin in characters (the default value is 0).

LNS Line Number Start. This gives the line numbers at the left hand edge of the screen in the preview mode.

DT43,44,45,46 Define Tabs. This sets the 'Tab' positions from the left margin. The Tab positions are used to space the address and date neatly on the right hand side at the top of the page. These numbers ascend in twos with a number for each line of the address. Care should be taken that the sum of the number for each line plus the number of characters on the right of that line does not exceed the number used for the Line Length. This method of locating the address and date is used so that telephone numbers, reference indicators etc. can be added or removed without affecting the position of the date or address. A '>' is shown to indicate the pressing of the Tab key.

JO Justification On. This spaces the words out on the line so that both the right and left margins are a straight line. The code to turn justification off is f1NJf2.

TI8 Temporary Indent 8. This code is used to indent the start of a paragraph by 8 characters. Provided this code is the first entry on a line, it starts that one line 8 characters in from the left hand margin. If it is placed after any character or characters on a line it forces a new line to start and the first character on that new line will be 8 characters from the left hand margin.

LS3 Line Spacing 3. This sets the number of blank lines between lines of text to give a large space for the signature. In the example above the space will be 48 x 3/216 = 2/3 of an inch. Any subsequent lines will be at this spacing unless reset by another code e.g. f1LS1f2 would return the line spacing to the original distance. Alternatively, the same size of space could be left for the signature by using the code f1ES51,144f2 in place of

Continued on page 56

BEEBUG Function/Procedure Library (4)

by Stefano Spina and P.A.Bolton

This month we present the final collection of functions and procedures by Stefano Spina, and a further procedure contributed by P.A.Bolton.

The first procedure provides useful conversions between different formats of numbers (binary, decimal, hexadecimal etc.). Note that the numbers must be specified in string format, and that the converted number is also returned as a string.

Then we have a number of disc-related routines, some of which are restricted in use to the ADFS (marked in the description where appropriate). Finally, a number of miscellaneous routines are listed. There are two window drawing routines, one by Stefano Spina which allows either a text or a graphics window to be drawn, one by P.A.Bolton which permits both text and graphics windows to be drawn of a similar size with an enclosing border. Take your choice depending on your requirements.

Future instalments of the library will be published as and when further collections of routines become available. Contributions from readers are very welcome, but should conform to the style which you see below. All contributions published will be paid for.

THE FUNCTION/PROCEDURE LIBRARY (PART 4)

Routine 22: Conv

Type: PROCEDURE

Syntax: PROCconv(N$,M%)

Purpose: Performs numerical conversion between decimal, hexadecimal, octal and binary strings.

Parameters: N$ Number to be converted (passed as a string)

M% Flag for base number
- 0 N$ is decimal
- 1 N$ is hexadecimal
- 2 N$ is octal
- 3 N$ is binary

Notes: Numbers must be passed in string format; for decimal and hexadecimal types the STR$ function can be used. Four global string variables hold the results:
- dec$ Decimal value
- hex$ Hexadecimal value
- oct$ Octal value
- bin$ Binary value

Related: None

```
10 num%=255
20 PROCconv(STR$num%,0)    bin$="255"
                           hex$="FF"
                           oct$="377"
                           bin$="11111111"

10 num$="FF"
20 PROCconv(num$,1)        bin$="255"
                           hex$="FF"
                           oct$="377"
                           bin$="11111111"

10 num$="377"
20 PROCconv(num$,2)        bin$="255"
                           hex$="FF"
```

```
                        oct$="377"
                        bin$="11111111"
10 num$="11111111"
20 PROCconv(num$,3)     bin$="255"
                        hex$="FF"
                        oct$="377"
                        bin$="11111111"
```

Routine 23: DirCheck
Type: FUNCTION
Syntax: FNdirck(F$)
Purpose: Check for a valid ADFS pathname
Parameters: F$ Pathname to be checked
Notes: This function gives a TRUE value if the pathname is legal else the exit value is FALSE. The pathname is not checked against an actual disc, but only for its syntax.
Related: None

```
10 INPUT"Pathname: "path$
20 IF NOT FNdirck(path$) THEN
   PROCalm(7): GOTO10
30 OSCLI("DIR "+path$)
```

Routine 24: Found
Type: FUNCTION
Syntax: FNfound(F$)
Purpose: Gives a TRUE value if the required file exists on the actual disc else the exit value is FALSE
Parameters: F$ File/Pathname to be checked
Notes: None
Related: None

```
 90 file$="MagStore"
100 path$="$.DATA"
110 IF NOT FNfound(path$+"."+file$) THEN
    PROCalm(6)
```

Routine 25: Free
Type: FUNCTION
Syntax: FNfree
Purpose: Gives the free memory on the actual disc
Parameters: None
Notes: This value can be used in a Basic program to check if a file can be loaded or if a disc has been changed (the chance of two discs with exactly the same free memory is very unlikely).
Related: None

```
90 IF FNfree >= 50000 THEN
   PROCload(file$) ELSE PROCalm(2)
```

Routine 26: PrtCheck
Type: PROCEDURE
Syntax: PROCprtchk
Purpose: Checks for printer status, avoids characters being lost
Parameters: None
Notes: The alarm message is just an example and should be adapted to the context of the program in which it is used.
Related: None

```
10 PROCprtchk:
   REM here the program remains until
       printer is OK
20 PROCprint
```

Routine 27: Shadow
Type: FUNCTION
Syntax: FNshd
Purpose: Determines whether or not shadow memory is in use on a Master 128 or Master Compact.
Parameters: None

Notes: The result is TRUE for shadow memory otherwise FALSE
Related: None

```
100 IF NOT FNshd THEN OSCLI"SHADOW"
```

Routine 28: Size
Type: FUNCTION
Syntax: FNsize(F$)
Purpose: Gives the size of the specified file (in bytes).
Parameters: F$ File or path name
Notes: None
Related: None

```
 90 file$=":0.$.DATA.Store"
100 IF FNsize(file$) > 120000 THEN
    PRINT"Too Long"
```

Routine 29: Wait
Type: PROCEDURE
Syntax: PROCwt(T%)
Purpose: Waits until a key is pressed if T% is set to 0, else waits until the given times is up or a key is pressed.
Parameters: T% Time to wait (in centi-seconds)
Notes: None
Related: None

```
100 PRINT"Press Any Key"
120 PROCwt(0)
```

Routine 30: Window
Type: PROCEDURE
Syntax: PROCwind(A%,B%,C%,D%,M%)
Purpose: Opens text/graphic windows
Parameters: A% Left text/graphic column co-ordinate
B% Bottom text/graphic row co-ordinate
C% Right text/graphic column co-ordinate
D% Upper text/graphic row co-ordinate
M% Flag to set text/graphic window
0 Text window
1 Graphic window
Notes: If M% is set to zero the co-ordinates must be given in terms of column and rows, else the graphic co-ordinates must be given.
Related: None

```
100 MODE 129
100 PROCwind(0,25,39,0,0):
    REM text from col. 0, row 25 to
    col. 39, row 0
120 PROCwind(0,0,1279,200,1):
    REM graphic from pixel 0/0 to
    pixel 1279/200
```

Routine 31: Window_Text
Type: PROCEDURE
Syntax: PROCtext(XA,XB,YA,YB)
Purpose: Defines a text window. Draws a border around the window and defines the graphics area.
Parameters: XA Left hand edge of window
XB Bottom edge
YA Right hand side
YB Top edge
Notes: Min and Max numbers for XA/YA are 1-78. Min and Max numbers for XB/YB are 1-30. Only works in mode 0. Gives impression of window being 'lifted off' screen.
Related: None

```
23330 REM Conv
23340 :
23350 DEF PROCconv(N$,M%)
23360 LOCAL N%
23370 IF M%=0 N%=VAL(N$) ELSE IF M%=1 N%
=EVAL("&"+N$) ELSE IF M%=2 PROCconv1(N$,
8) ELSE IF M%=3 PROCconv1(N$,2)
23380 dec$=STR$(N%):hex$=STR$~(N%)
23390 IF M%=2 oct$=N$ ELSE PROCconv2(N%,
8)
23400 IF M%=3 bin$=N$ ELSE PROCconv2(N%,
2)
23410 ENDPROC
23420 :
23430 DEF PROCconv1(N$,M%)
23440 LOCAL B$,B%,L%
23450 L%=LEN(N$):B%=0:N%=0
23460 REPEAT
23470 B$=MID$(N$,L%-B%,1)
23480 N%=VAL(B$)*(M%^B%)+N%:B%=B%+1
23490 UNTIL B%=L%
23500 ENDPROC
23510 :
23520 DEF PROCconv2(N%,M%)
23530 IF N%<8 AND M%=8 oct$=STR$(N%):END
PROC
23540 LOCAL A$,R%
23550 R%=0:A$=""
23560 REPEAT
23570 R%=N% MOD M%:N%=N%/M%
23580 A$=STR$(R%)+A$
23590 UNTIL N%<M%
23600 A$=STR$(N%)+A$
23610 IF M%=8 oct$=A$ ELSE bin$=A$
23620 ENDPROC
23630 :
23640 REM DirCheck
23650 :
23660 DEF FNdirck(F$)
23670 LOCAL A$,B$,A%,B%,C%,D%:A%=LEN(F$)
23680 IF ((LEFT$(F$,1)=".") OR (RIGHT$(F
$,1)=".") OR (A%>1 AND LEFT$(F$,1)="$" A
ND LEFT$(F$,2)<>"$.")):=FALSE
23690 B%=1:C%=0:D%=TRUE
23700 REPEAT
23710 A$=MID$(F$,B%,1):IF A$="." C%=B%
23720 IF (A$<>"." AND (B%-C%)>10) OR (A$
="$" AND B%>1) OR (A$="." AND MID$(F$,B%
+1,1)=".") D%=FALSE ELSE B%=B%+1
23730 UNTIL B%>A% OR NOT D%
23740 =D%
23750 :
23760 REM Found
23770 :
23780 DEF FNfound(F$)
23790 LOCAL A%
23800 A%=OPENIN(F$):CLOSE#A%
23810 =(A%<>0)
23820 :
23830 REM Free
23840 :
23850 DEF FNfree
23860 LOCAL A%,B%,X%,Y%,Z%
23870 DIM B%4:A%=&71:X%=B%
23880 Y%=B%DIV256:Z%=USR(&FFF1)
23890 =!B%
23900 :
23910 REM PrtCheck
23920 :
23930 DEF PROCprtchk
23940 IF FNprtchk ENDPROC
23950 LOCAL B%
23960 REPEAT
23970 PRINTTAB(10,10)"Check printer and
press any key":B%=GET
23980 UNTILFNprtchk
23990 ENDPROC
24000 :
24010 DEF FNprtchk
24020 VDU2,1,31,1,127,3
24030 =(ADVAL(-4)=63)
24040 :
24050 REM Shadow
24060 :
24070 DEF FNshd
24080 LOCAL A%,X%,Y%,Z%
24090 A%=&EF:X%=0:Y%=255:Z%=USR(&FFF4)
24100 IF VAL(LEFT$(STR$~(Z%AND&0000FF00)
,1))=0:=TRUE ELSE=FALSE
```

Continued on page 51

Printing Scientific Characters with Word Processors (Part 2)

By Gareth Leyshon

Last month I looked at designing UDGs (user-defined graphics) for printers, and using them with WYSIWYG word processors. None of these methods, however, provides a particularly good screen display of the actual characters. Now I shall explore one alternative for Master users.

Numeric keypad codes: alone....			and with Shift		
Key	New ASCII code	Edit-Plus symbol	New ASCII code	Edit-Plus symbol	Key
+	239	+	255	Unprintable	+
-	241	-	225	±	-
/	243	÷	227	↧	/
*	238	×	254	⌘	*
#	231	As Shift-3	247	As 3	#
,	240	Not used	224	Not used	,
RET	209	↵	224	Not used	RET
.	242	·	226	•	.
0	244	₀	228	⁰	0
1	245	₁	229	¹	1
2	246	₂	230	²	2
3	247	₃	231	³	3
4	248	₄	232	⁴	4
5	249	₅	233	⁵	5
6	250	₆	234	⁶	6
7	251	₇	235	⁷	7
8	252	₈	236	⁸	8
9	253	₉	237	⁹	9

Table 1. Codes and suggested UDGs for a scientific character set

EDIT AND EDIT-PLUS

Being less familiar with View, I turned to Edit, the Master's built-in text processor. Edit can display any character, so showing the right symbols on screen is easy. But using Edit brings out other problems: underlining and other 'embedded' commands can only be performed using cumbersome three-character codes such as '.bu'; it has the annoying habit of automatically justifying everything it prints (though it doesn't do so on screen), and it does not have a pound patch. To overcome these problems I wrote the *Edit-Plus* program listed here, a Basic utility to make the use of Edit easier, if not truly WYSIWYG. Unfortunately, since Edit is not standard on the model B, this method only applies to the Master and does not lend itself to easy conversion.

Master computers have a numeric keypad in addition to the main keyboard. These keys provide the redundant keys necessary to enter new character codes directly (another obstacle to Beeb conversion). By setting a high base character for the keypad, and another base for the keypad with Shift, more than 30 extra character codes can be input. The function keys together with Ctrl and Shift provide the other ten to make it up to the P1081's maximum of forty (differences for other printers will be dealt with later). I tend to use the number keys and plus and minus for the subscripts of the same symbols, and the same keys, shifted, to give the equivalent superscripts. A function key strip can list the ten for the function keys. The other codes can be assigned to other keys on the numeric keypad, with the following constraints:

* The hash key works as Shift-3 and vice versa.
* The Delete key has a large offset compared to the other keys, and when the numeric keypad is rebased above character 128, its code will probably be within the standard character set - as I have set it, it comes up as 'c'.
* The Shift-Plus key (Shift-+) has code 255, which some printers (including the P1081) may not allow you to redefine.

The other keys - slash, asterisk, comma, Return and full stop - can be used as you see fit, bearing in mind that the maximum number of characters you can use is limited by the printer UDG buffer - 40 for the P1081. I have included some other generally useful characters in the Edit-Plus DATA lines, identified by REM statements so that you may easily change them. Table 1 shows both the codes and suggested UDGs for each character.

Edit places no restrictions on the display of characters or the keys used for entry, so this makes the new characters easy to type in, and you can appreciate roughly what the display will look like. However, it is not as user-friendly as some word processors. One thing which can be adjusted from the keyboard is the screen colour.

Normally Ctrl-key commands are put into the text and so VDU functions cannot be accessed this way. However, if you are in a text line input mode (e.g. by pressing f1 to type in a star command) you can use VDU functions, including colour redefinition. While in command-entry mode, hold down the Ctrl key and type (for instance) SGD@@@ - this changes the screen colour (standard white is colour 7, i.e. code G) to blue (colour 4, code D). Background colour can also be changed.

As described above, Edit annoyingly automatically formats and justifies the text, and requires three-letter codes for other functions. To avoid these problems, I incorporated a rudimentary printer driver into the Edit-Plus program, which takes an Edit file from disc and prints it out, preceding this with the relevant (user-definable) printer character definition codes.

Functions such as underlining are achieved by single-character codes. Codes 0 to 31 are not printable and are shown by Edit in inverse colours. These are entered from the keyboard using the Ctrl key. Thus Ctrl-U marks the beginning or end of an underlined region, Ctrl-B is used for bold, and so on. While these codes are still visible on the display, a lone inverse character is more noticeable than two lower case letters preceded by a dot. The full list of default recognised codes is given in Table 2. Note the inclusion of null code Ctrl-@. If you are trying to align several rows of text into a column where some rows are underlined, the lines preceded by the Ctrl-U character will look indented on the screen. Inserting the CTRL-@ allows the other lines to appear equally indented.

@	-	null code
U	-	underlining on/off
B	-	bold on/off
X	-	standard printer mode
D	-	Edit-Plus default printer mode
J	-	fully justify
L	-	left justify
R	-	right justify
C	-	centre
P	-	new page

Table 2. Control codes which may be inserted into Edit files

SETTING UP EDIT-PLUS

Type in the listing as provided. Lines 1000-1630 contain the main procedures, and 3000-3090 the error handling routine. 2000-2220 contain a data-checking routine of which I will have more to say later. Most of the parameters the user will want to change are contained in the DATA statements from line 10000 onwards, which are grouped into four distinct sections.

The first category, 10000-10450, contains what I refer to as my standard new character set, namely the superscript and subscript digits, plus the superscript plus and minus characters. These I use most often and am least likely to want to change, so put them first. The second group, 11000-11510, contains the 'auxiliary new character set' - they are

identical in structure to the previous set, only they are more likely to be changed. It is here that you are most likely to place the data for your preferred characters.

Each line of data is preceded by a REM statement reminding you what the data is for. The REM statements can be omitted, but I do not recommend this for your master copy of the program, since you may later wish to change selected definitions. You may, however, compact the REMs out of a working copy whose character set you are happy with. Each line must take one of four formats:

1. DATA n,a,b,c,d,e,f,g,h,A,B,C,D...Z
   This is the most common form. 'n' is the code of the character to redefine; 'a' through to 'h' are the eight codes for the screen UDG; and 'A' to 'Z' are the codes to be sent to the printer to define its UDG, however many it needs.

2. DATA n,a,b,c,d,e,f,g,h,-1
   This redefines screen character 'n' but leaves its printer definition intact. You may, for instance, be using an international character set which prints different symbols corresponding to the codes for []{}\~ etc. on paper, and wish to make those characters have matching screen appearances.

3. DATA n,-1
   This serves no practical purpose, but it is there to remind you of unused codes. Since there are slightly more codes on the keypad than I have used for the samples (constrained by my printer UDG buffer), I have noted the spare codes in this way. You may choose to leave others unused instead.

4. DATA -1
   This is the 'end of character data' marker. The only constraint on positioning is that the DATA statement for the first character must be the first DATA statement in the whole program, and must be on or after line 10000. No other types of DATA may be included before the 'end of character data' marker.

After adding DATA statements, you may wish to ensure that all are the right length. This can be checked by typing:

```
PROCcheck
```

from command mode. If all is well, you will see all the data codes listed on screen; if not, you should spot the anomalies in the list, so that you can correct them. PROCcheck (lines 2000-2220) need not be typed in if you do not require this function.

EDIT-PLUS IN USE

On running the program, you will be asked for a filename; if it doesn't already exist, you will be asked if you want to create a new file or not. If it already exists, you will be asked whether you want to (P)rint it or (E)dit it.

In the case of a new file, or editing an old one, the screen characters only will be defined, and the relevant file loaded into Edit, where you may proceed.

If printing an existing file, you will be asked to insert a test piece of paper into the printer. This is because when the printer characters are defined they are printed to test them - in each case, to the right of a letter X. This makes it easy to check the positioning of super- or subscript characters. To suppress this, change line 1200 to read:

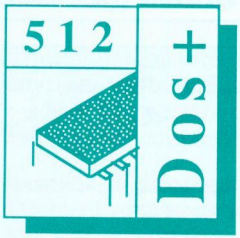
```
1200 PRINT N$;CHR$3:ENDPROC
```

Next, you will be asked to position the paper for the actual printout in the printer. You may press Return or Space to proceed, the former also defines the current paper position to be the 'top of page'. Note that Edit-Plus will not justify text when printed, but if your printer has word processing facilities, it may be able to do the job itself - configure this in the 'pre-print DATA'.

More technical information relating to this program, and changes which may be needed for other printers, will be dealt with in the conclusion to this article in the next issue of BEEBUG.

```
   10 REM Program Edit-Plus
   20 REM Version A1.3/M128
   30 REM Author  Gareth Leyshon
   40 REM BEEBUG  June 1991
   50 REM Program subject to copyright.
   60 :
  100 MODE 135
  110 ON ERROR GOTO 3000
  120 I%=0:REPEAT:INPUT'"" Please enter
filename:"F$
  130 IF F$>"" THEN F%=OPENIN(F$) ELSE F
%=-1
  140 IF F%=0 THEN F%=FNnofile
  150 UNTIL F%>0
  160 IF I%=0 THEN I%=FNPorE ELSE I%=3
  170 CLS:IF I%>2 THEN CLOSE#F%:PROCchar
(FALSE):OSCLI("EDIT "+F$) ELSE MODE 131:
PROCprint(F%)
  180 CLS:PRINT'" Edit/Print another fil
e?  Y/N"
  190 REPEAT:I%=INSTR("YyNn",GET$):UNTIL
I%>0
  200 IF I%<3 THEN RUN
  210 MODE 135
  220 *FX 254,1
  230 *FX 228,0
  240 *FX 238,48
  250 *FX 4
  260 END
  270 :
 1000 DEF PROCchar(P%):LOCAL X%,Y%,N%,I%
 1010 RESTORE 12000:READ T%,L%,Q%,Z$,Q$,
K$,P$,Y$,V$:Z$=EVAL(Z$):Y$=EVAL(Y$):V$=E
VAL(V$)
 1020 PROCenable:RESTORE 10000:IF P% THE
N PRINT CHR$2;EVAL(K$);CHR$3
 1030 REPEAT:READ I%:IF I%>-1 THEN PROCn
otnull
 1040 UNTIL I%=-1:IF P% THEN VDU 2,1,13,
3
 1050 ENDPROC
 1060 :
 1070 DEF PROCnotnull:READ K%:IF K%>-1 T
HEN PROCscr_udg
 1080 ENDPROC
 1090 :
 1100 DEF PROCscr_udg:VDU 23,I%,K%
 1110 FOR Y%=1 TO 7:READ J%:VDU J%:NEXT
 1120 IF P% THEN PROCcodes ELSE READ J%:
IF J%>-1 THEN FOR Y%=2 TO T%:READ J%:NEX
T
 1130 ENDPROC
 1140 :
 1150 DEF PROCcodes:READ J%:IF J%>-1 THE
N PROCprn_udg
 1160 ENDPROC
 1170 :
 1180 DEF PROCprn_udg:LOCAL N$:N$=CHR$2+
EVAL(Q$)+CHR$I%+CHR$1+CHR$J%
 1190 FOR Y%=2 TO T%:READ J%:N$=N$+CHR$1
+CHR$J%:NEXT
 1200 PRINTN$;:VDU 32,88,I%,3:ENDPROC
 1210 :
 1220 DEF PROCenable
 1230 REM REBASE NUMERIC KEYS
 1240 *FX 238,244
 1250 REM ACTIVATE CSf KEYS 0-9 = 180-18
9
 1260 *FX 228,180
 1270 REM ACTIVATE SHIFT-NUMERIC KEYS
 1280 *FX 254,0
 1290 ENDPROC
 1300 :
 1310 DEF PROCprint(F%):LOCAL A%,B%,C%,I
%,U%,V%,X%,A$,B$,C$,D$,E$,R$,X$
 1320 VDU 19,1,6|:RESTORE 12000:READ T%,
L%,Q%,Z$,Q$,K$,P$,Y$
 1330 Y$=EVAL(Y$):Z$=EVAL(Z$):RESTORE 13
000:DIM C$(Q%)
 1340 C$(0)="CHR$A%":FOR X%=1 TO Q%:READ
 V%,C$(X%):R$=R$+CHR$V%:NEXT
 1350 PRINT '"  Put the printer on-line
and insert a test piece of paper."'" Pres
s any key when you have done that."
 1360 OSCLI("FX 21,0"):IF GET
 1370 PROCchar(TRUE)
 1380 PRINT'"'"  Now insert your copy pap
er, press return to set ""top of page"""
'" otherwise, press the space bar. Print
out will then commence."
 1390 REPEAT:I%=INSTR(" "+CHR$13,GET$):U
NTILI%>0
 1400 PRINTCHR$2;Y$:IF I%=2 THEN PRINT E
VAL(P$);CHR$L%
 1410 REPEAT:A$="":B$="":C$="":D$="":E$=
"":C%=0:REPEAT:A%=BGET#F%:X$=FNadd(A%):A
$=A$+X$:C%=C%+LEN(X$):UNTILA%=13 OR C%=2
40 OR EOF#F%
 1420 IF A%<>13 AND NOT(EOF#F%) THEN REP
EAT:A%=BGET#F%:X$=FNadd(A%):B$=B$+X$:C%=
C%+LEN(X$):UNTILA%=13 OR C%=480 OR EOF#F
%
 1430 IF A%<>13 AND NOT(EOF#F%) THEN REP
```

Continued on page 54



512 Forum

by Robin Burton

Before going on to other topics, I'd like to thank those of you who have written in response to my request for your feelings and ideas about the content of the Forum.

No-one seems to have any major complaints and it appears that the overall feeling is that I should carry on much as before. In other words, each month's Forum will continue to be almost as much of a surprise to me as it is to you.

WHO-DUNNIT?

This month I've decided to tell you about something I recently discovered, and how I came to find out. There's also a lesson in this about how much time you can waste if you make assumptions about other people's software (clue, Acorn's).

Over the last couple of months I've been writing and testing a new program. Nothing unusual, I spend a good deal of my time doing that, but it proves it's never too late to learn something new. I've found some interesting limitations (that's the polite way of saying bugs) in the 512's XIOS.

As I've said previously, I won't advertise in the Forum, so I'll just say enough for you to understand what I was doing and why I was doing it. The program is for setting up variable hard disc partitions, so let's leave it at that. Even if you don't have a hard disc I think you'll still find the following item interesting and who knows, it might possibly also apply to some floppies. I don't know because I haven't tried it. By the time I'd sorted the problem out I'd had enough of this sort of thing.

You'll realise that I didn't find testing this program much fun. Basically, the first part of the exercise was to secure the contents of my entire hard disc, because creating a new partition always destroys all existing DOS files. In addition, if the size of the partition is to increase from the previous one, you almost certainly have to remove (or move) your ADFS files as well to free sufficient disc space.

So far so good, but having created a new partition how do you go about testing it? The answer of course is that you must first load it with data, and what could be better than real files? In my case this consisted of roughly twenty or thirty directories containing about 15 megabytes of data in several hundred files, plus about another five megabytes in ADFS.

You'll also guess without much effort that after the basic testing of the first few different partition sizes the novelty of this job began to wear a bit thin, even though I'd only had to secure the files once when I started. Still, this direct method was all right to establish that the program seemed to be working before investing more effort in setting up specific tests.

A further consideration was that although I had about 15 megabytes of DOS data which I could load, this would leave me somewhat short of both data and files for testing partitions of between 15 and 32 megabytes. I could have recovered the same files to extra, new directories with names like test1, test2 and so on, but the time taken to do this would be enormous for the larger partitions (as you'll know if you've ever had to recover a large 512 hard disc).

There had to be a better way, and so there was, but (very) indirectly this led to my

discoveries. Of course, to start each partition off I had to recover my DOS utilities and one or two other items needed during the tests, but this did reduce the basic recovery job to about five minutes. With a bit of work I was then able to carry out the most lengthy tests in a much more automated fashion. These are filling directories and filling the disc. The beauty of automating the process, of course, was that I could just set it off and then go and do something else.

THE EVIDENCE

The first test, checking what happens when a directory becomes full, is pretty basic stuff, but even so it had to be done, both for the root and then for sub-directories. There are several ways to tackle filling a directory with files, but the most obvious was to use BBCBASIC. The program was simple, using nested FOR-NEXT loops to increment the last two characters of a filename, opening the file, printing its name to the file then closing it. I duly set the program going and went away.

You'd probably expect, as I did, to come back to the machine some time later to find a 'Dir full' message, and naturally a directory full of files. If that's what you'd expect you'd have been as surprised as I was to find that the system had crashed. There was just a non-stop scrolling 'Invalid opcode' display on the screen.

I rebooted and checked the root directory. It was indeed full, but why had it crashed? I (manually) deleted one file and tried to copy one - instant hang-up! On rebooting I found that the directory should have held the extra file, but it wouldn't. I deleted a dozen files and copied new ones until it crashed again, this time finding that the 509th file hung the system. Obviously the directory was somehow damaged!

My first reaction was the obvious one, the hard disc partitioner was at fault. After all I'd been changing the number of root directory entries from standard amongst various other things. I therefore next created another partition, but with the standard number of 512 root entries. I then re-ran the test.

Same result! What next? Could it be BBCBASIC? It didn't take long to write a machine code program to do exactly the same as the BBCBASIC program. I then created another (standard root) partition and tried again. Guess what! The same result!

I was pretty confident that my partition program was all right, particularly using standard settings, so I persisted. The next step was to fill the directory using standard DOS COPY commands in a short batch file. This took ages to run, because each new entry now involved both reading and writing a file, but a couple of hours later I came back to find the machine had hung with no display. I rebooted and sure enough the directory was full again.

It now looked as though there was something wrong in my partitioning program after all, so to eliminate that I created yet another new partition, this time using the standard Acorn program. This limits the range of partition sizes to four and it doesn't mess about with the configuration, so surely I'd now get to the bottom of the problem.

THE GUILTY PARTY

Yes I did. Maybe I should have realised sooner and perhaps you have already. My excuse is that by this stage I'd been creating partitions, filling the root directory and crashing the system for nearly a day. The answer, it turns out, is the simple one (of course!).

If you fill the root directory of a 512 hard disc partition the system crashes.

I hope that little gem saves somebody some trouble, it took most of a day for me to prove it beyond doubt. Of course I'd had plenty of 'red-herrings' to chase in the form of more likely candidates for the cause, and proving it when you know the answer is much easier. I wasn't too pleased as you might expect. I'd spent all day chasing a bug which Acorn built into the system.

My conclusion, for what it's worth, is that Acorn knew about the problem and decided on 512 entries for the hard disc root directory simply because it's such a large number (no-one was ever likely to find the bug). It must be said that in normal use performance degrades so badly after the first few dozen files that the urge to create a new subdirectory does become overpowering, but it's hardly satisfactory.

BATCH TECHNIQUES

As part of the testing above I used batch files and they reminded me of a couple of questions I've been asked.

The first one is how can you prevent the output from a batch file being written to the screen? Using 'ECHO OFF' stops all the commands being displayed as they're executed (except the echo off itself) but everything else, such as files being copied, is displayed. There are only two methods I can think of and they both involve some restrictions.

The simplest technique is to set the screen foreground colour to the same as the background using the 'COLOUR' command at the start of the file; then include a 'CLS' before you reset the foreground colour at the end of the batch job (otherwise all the output will re-appear). This doesn't actually prevent screen output, so you'll see the COLOUR command for a moment, but it does hide the rest of the output. The disadvantage is that anything previously displayed is also lost.

An alternative method is to use two batch files, redirecting the screen output of the second one (the one that does the work) to a file in the call to it, then deleting this file at the end of the job.

An example will make this clearer, let's call the files 'BATCH1' and 'BATCH2'. Here's how BATCH1 looks:

```
BATCH2 >TEMPFILE
DEL TEMPFILE
```

That's all there is to it. It simply calls the second file and redirects any console output from it to the file 'TEMPFILE'. On return from BATCH2 the file is deleted. BATCH2 can do anything you like, but for illustration, just catalogue the disc with a 'DIR'. If you create these two files and try them you'll soon get the idea.

The disadvantages this time are also twofold. First, as before this method doesn't hide the initial command in BATCH1. Secondly, anything which should affect the screen, such as a PCSCREEN command, a COLOUR command and so on are actually implemented in the BBC as VDU strings. Because the VDU output (from the DOS program) is being re-directed it never gets to the BBC micro and so such commands won't work during re-direction. On the other hand, it is a technique which can be 'switched on and off' as many times as you like within any batch file, so you can largely get round the problem.

So far as I know, there isn't a perfect answer to this problem in the 512, but by using these two methods and ECHO OFF in various combinations you can certainly tidy up your batch file displays if you don't want someone to see everything that they're doing as they execute.

As a final point, if either (or both) of these techniques are used in your AUTOEXEC file the initial command doesn't appear on screen anyway, so if you want a totally

invisible start-up when you boot the 512 here are two ways in which you can do it.

SHARED INFORMATION

You'll remember I offered to include addresses in the Forum if anyone would like to correspond with other 512 users. Well one member, Ron Thompson, has taken up the offer. I've known Ron (by letter) for a couple of years or so and I can tell you that he's a very keen 512 user.

In fact I met Ron late last year in North Devon. I was there visiting relatives, but Ron had to drive for three hours from the far South-West of Cornwall, which might tell you why he's keen to correspond with 512 users.

Living where he does, Ron is very much isolated from fellow users, computer clubs and even from Acorn dealers and (PC) software shops. I know he writes to one other user already and he tells me that exchanging information about applications, shareware and the 512 in general has proved very valuable. Not surprising, after all, 512 Forum readers together represent a vast reservoir of information about using the 512 - why not share it. If anyone would like to join in write to:

Mr. R. Thompson, 2 Gibbons Field, Mullion, Helston, Cornwall TR12 7EA .

Who knows, it might be the start of a 512 users club by mail.

We regret that the telephone number given in last month's Forum for Shibumi Software was incorrect. The company cannot currently be contacted by phone. Please do **not** use the number published last month.

BEEBUG Procedure/Function Library (continued from page 43)

```
24110 :
24120 REM Size
24130 :
24140 DEF FNsize(F$)
24150 LOCAL A%,B%
24160 B%=OPENIN(F$)
24170 A%=EXT#B%
24180 CLOSE#B%:=A%
24190 :
24200 REM Wait
24210 :
24220 DEF PROCwt(T%)
24230 LOCAL D%
24240 IF T%=0 T%=GET ELSE D%=INKEY(T%)
24250 ENDPROC
24260 :
24270 REM Window
24280 :
24290 DEF PROCwind(A%,B%,C%,D%,M%)
24300 IF M%=0 VDU28,A%,B%,C%,D% ELSE VDU
24,A%;B%;C%;D%;
24310 ENDPROC
24320 :
24330 REM Window_Text
24340 REM Author P.A.Bolton
24350 :
24360 DEF PROCtext(XA,XB,YA,YB)
24370 LOCAL Xa,Xb,Ya,Yb
24380 VDU28,XA,XB,YA,YB
24390 Xa=(XA*16)-10
24400 XB=32-XB:Xb=(XB*32)-45
24410 Ya=(YA*16)+25
24420 YB=32-YB:Yb=(YB*32)+15
24430 VDU24,Xa+10;Xb-10;Ya+10;Yb-10;
24440 MOVEXa+10,Yb-10:MOVEXa+10,Xb-10:PL
OT81,Ya+10,0:MOVEXa+10,Yb-10:MOVEYa+10,Y
b-10:PLOT81,0,-(Yb-Xb)
24450 VDU24,Xa;Xb;Ya;Yb;
24460 CLG
24470 MOVEXa,Xb:DRAWXa,Yb:DRAWYa,Yb:DRAW
Ya,Xb:DRAWXa,Xb
24480 ENDPROC
```

held in E%. With large numbers, PROCIndex calls PROCProduct both for the repeated squaring and for multiplying together the resulting terms. It is the relative slowness of PROCProduct that makes the calculation of powers to large moduli by PROCIndex take a long time.

For references refer to Vol.10 No.1.

```
   10 REM Program Fermat
   20 REM Version B1.29
   30 REM Author  Michael Taylor
   40 REM BEEBUG  June 1991
   50 REM Program subject to copyright
   60 :
  100 ONERROR REPORT:PRINT" at Line ";ER
L:END
  120 MODE0
  130 VDU19,1,0,0;0;0;19,0,7,0;0;0;
  140 VDU23,0,10,96,0;0;0;23,0,11,7,0;0;
0;
  150 VDU23,240,0,&FF,0,&FF,0,&FF,0,0
  160 DIM D%(32)
  170 @%=0:M%=1000000000
  180 PROCAnnounce
  190 ONERROR IF ERR=26 PRINT;" INVALID
INPUT.":PROCShove(5,5):GOTO 200 ELSE REP
ORT:PRINT" at Line ";ERL:END
  200 PROCMainloop
  210 END
  220 :
 1000 DEF PROCAnnounce
 1010 PRINT'SPC29;"MODULAR  ARITHMETIC"
 1020 PRINT'" This program calculates";S
PC(8);"A + B  (mod m),"'';SPC(27);" or
A - B  (mod m),"'';SPC(27);" or  A x B
(mod m),"'SPC(27);" or  A / B  (mod m)
 if B relatively prime to m,"'''';SPC(27)
;" or"SPC(4);
 1030 PRINT" A";:VDU11:PRINT"B";:VDU10:P
RINT "  (mod m),"''''" where m ranges fro
m 2 to 1000000000, and A and B range fro
m 1 to m."'SPC39;"(Be patient with big
moduli and powers!)":VDU31,0,VPOS:VDU11
 1040 ENDPROC
 1050 :
 1060 DEF PROCMainloop
 1070 REPEAT
 1080 VDU10:A$="0":B$="0":N$="0":A%=0:B%
=0:N%=0
 1090 REPEAT:INPUT" What is m? "N$:N%=FN
Test(N$,M%):UNTIL N%<>0
 1100 REPEAT:INPUT" What is A? "A$:A%=FN
Test(A$,M%):UNTIL A%<>0
 1110 REPEAT:INPUT" What is B? "B$:B%=FN
Test(B$,M%):UNTIL B%<>0
 1120 W%=INT((2^31)/N%)
 1130 PROCShove(11,11)
 1140 PRINT'"  +  -  x  /  or  ^ ?
"'
 1150 REPEAT
 1160 INPUT" Operator?  "Op$
 1170 IF Op$=";" OR Op$="+" PROCAdd:Op$=
"Done"
 1180 IF Op$="-" OR Op$="=" PROCSubtract
:Op$="Done"
 1190 IF Op$="x" OR Op$="X" PROCMultiply
:Op$="Done"
 1200 IF Op$="/" OR Op$="?" PROCDivide:O
p$="Done"
 1210 IF Op$="^" OR Op$="~" PROCPower:Op
$="Done"
 1220 IF Op$<>"Done" PROCClear
 1230 UNTIL Op$="Done"
 1240 UNTILFALSE
 1250 ENDPROC
 1260 :
 1270 DEF PROCAdd
 1280 PROCTabs
 1290 PRINT;A%;"  +  ";B%"  ";CHR$240;CH
R$240;"  ";
 1300 PRINT (A%+B%)MODN%;"  (mod  ";N%;"
)"
 1310 ENDPROC
 1320 :
 1330 DEF PROCSubtract
 1340 PROCTabs
 1350 PRINT;A%;"  -  ";B%"  ";CHR$240;C
HR$240;"  ";
 1360 L%=(A%-B%)MODN%:IF L%<0 L%=L%+N%
 1370 PRINT L%MODN%;"  (mod  ";N%;")"
 1380 ENDPROC
 1390 :
 1400 DEF PROCMultiply
 1410 PROCTabs
 1420 PRINT;A%;"  x  ";B%"  ";CHR$240;CH
R$240;"  ";
```

```
 1430 PRINTFNProduct(A%,B%);"  (mod  ";N
%;")"
 1440 ENDPROC
 1450 :
 1460 DEF PROCDivide
 1470 IF B%=N%:PROCClear:VDU10,10:PROCCl
ear:PRINT" Division is not defined (sinc
e B = m).":ENDPROC
 1480 PROCTabs
 1490 PRINT;A%;"  /  ";B%"  ";CHR$240;CH
R$240;"  ";
 1500 I%=FNInverse(B%,N%)
 1510 IF I%=0 PROCClear:VDU10,10:PROCCle
ar:PROCMessage:ENDPROC
 1520 PRINTFNProduct(A%,I%);" (mod  ";N%
;")"
 1530 ENDPROC
 1540 :
 1550 DEF PROCPower
 1560 PRINT'TAB(46-LENSTR$A%-LENSTR$B%)A
%;:VDU11:PRINTB%;:VDU10:PRINT"  ";CHR$24
0;CHR$240;"   (Calculating)";
 1570 VDU 31,POS-14,VPOS:PRINTFNIndex(A%
,B%);" (mod  ";N%;")";SPC5
 1580 ENDPROC
 1590 :
 1600 DEF PROCTabs:PRINT'TAB(41-LENSTR$A
%-LENSTR$B%);
 1610 ENDPROC
 1620 :
 1630 DEF PROCMessage
 1640 PRINT" ";B%;" and ";N%;" are not r
elatively prime (their gcd is ";gcd%;"),
"'" and so there is no multiplicative in
verse and division cannot be defined."
 1650 ENDPROC
 1660 :
 1670 DEF PROCShove(U%,V%)
 1680 FOR Z%=1 TO U%:VDU10:NEXT
 1690 FOR Z%=1 TO V%:VDU11:NEXT
 1700 ENDPROC
 1710 :
 1720 DEF PROCClear:VDU13,11:PRINTSPC79:
VDU13,11
 1730 ENDPROC
 1740 :
 1750 DEF FNIndex(U%,V%)
 1760 U%=U%MODN%:S%=1:Z%=-1
 1770 REPEAT
 1780 Z%=Z%+1:E%=V%MOD2:V%=V%DIV2
 1790 IF Z%=0 Y%=U% ELSE IF N%<46340 OR
Y%<46340 Y%=(Y%*Y%)MODN% ELSE Y%=FNProdu
ct(Y%,Y%):REM If Y%*Y%(MOD N%) might be
more than 2^31-1, finds FNProduct(Y%,Y%)
 1800 IF E%=1 AND (S%<46340 AND Y%<46340
) S%=(S%*Y%)MODN% ELSE IF E%=1 S%=FNProd
uct(S%,Y%): REM If S%*Y%(MOD N%) might b
e more than 2^31-1 finds FNProduct(S%,Y%
)
 1810 UNTIL V%=0
 1820 =S%
 1830 :
 1840 DEF FNProduct(U%,V%): LOCAL Z%,K%,
R%,G%
 1850 K%=U%MODN%:V%=V%MODN%:Z%=-1
 1860 REPEAT
 1870 Z%=Z%+1:D%(Z%)=K%MODW%
 1880 K%=K%DIVW%
 1890 UNTIL K%=0
 1900 R%=Z%:G%=0
 1910 FOR Z%=R% TO 0 STEP -1
 1920 G%=((W%*G%)MODN%+D%(Z%)*V%)MODN%
 1930 NEXT
 1940 =G%
 1950 :
 1960 DEF FNInverse(U%,V%):LOCAL A%,B%,C
%,D%,X%,Y%,Q%
 1970 A%=1:B%=0:C%=0:D%=1
 1980 Y%=V%:X%=U%
 1990 REPEAT
 2000 Q%=Y%DIVX%:Y%=Y%MODX%
 2010 A%=A%-Q%*C%:B%=B%-Q%*D%
 2020 Z%=A%:A%=C%:C%=Z%
 2030 Z%=B%:B%=D%:D%=Z%
 2040 Z%=X%:X%=Y%:Y%=Z%
 2050 UNTILX%=0
 2060 IF Y%=1 = (B%+V%)MODV% ELSE gcd%=Y
%: =0
 2070 :
 2080 DEF FNTest(T$,U%)
 2090 IF VALT$=0 OR VALN$=1 PROCClear: =
0 ELSE T=EVALT$
 2100 IF N%=0 AND(T>U% OR T<1) PROCClear
: =0
 2110 IF N%<>0 AND(T>N% OR T<1) PROCClea
r: =0
 2120 IF INT(T)<>T PROCClear: =0
 2130 =T
```

```
EAT:A%=BGET#F%:X$=FNadd(A%):C$=C$+X$:C%=
C%+LEN(X$):UNTILA%=13 OR C%=720 OR EOF#F
%
 1440 IF A%<>13 AND NOT(EOF#F%) THEN REP
EAT:A%=BGET#F%:X$=FNadd(A%):D$=D$+X$:C%=
C%+LEN(X$):UNTILA%=13 OR C%=960 OR EOF#F
%
 1450 IF A%<>13 AND NOT(EOF#F%) THEN REP
EAT:A%=BGET#F%:X$=FNadd(A%):E$=E$+X$:C%=
C%+LEN(X$):UNTILA%=13 OR C%=1200 OR EOF#
F%
 1460 PRINTA$;B$;C$;D$;E$;CHR$(-13*EOF#F
%)
 1470 UNTIL EOF#F%
 1480 PRINTV$;EVAL(K$);CHR$3:CLOSE#F%
 1490 VDU 7:PRINT''" PRINTOUT CONCLUDES.
 PRESS ANY KEY."
 1500 OSCLI("FX 21,0"):IF GET
 1510 ENDPROC
 1520 :
 1530 DEF FNadd(A%):LOCAL R%
 1540 R%=INSTR(R$,CHR$A%):B%=B%EOR(-(R%=
3)):U%=U%EOR(-(R%=2))
 1550 =EVAL(C$(R%))
 1560 REM Any variables to be toggled sh
ould be changed here.
 1570 DEFFNPorE:PRINT''" (P)rint or (E)d
it file?"
 1580 REPEAT:I%=INSTR("PpeE",GET$):UNTIL
I%>0:=I%
 1590 DEFFNnofile:PRINT''" Not found!"CH
R$7''"Create new file? Y/N"
 1600 REPEAT:I%=INSTR("YynN",GET$):UNTIL
I%>0
 1610 IF I%<3 THEN F%=OPENOUT(F$)
 1620 =F%
 1630 :
 2000 REM The following procedure may be
 called from command mode.
 2010 REM It ensures the DATA lines are
the right length.
 2020 DEFPROCcheck:VDU 12,14
 2030 RESTORE 12000:READ T%
 2040 RESTORE 10000:REPEAT:READ A%
 2050 IF A%>-1 THEN PROCone
 2060 UNTIL A%=-1
 2070 PRINT '"Listing concludes.":VDU 15
 2080 ENDPROC
 2090 :
 2100 DEF PROCone
 2110 PRINT'"Screen char:";CHR$A%
 2120 READ B%:IF B%>-1 THEN PROCtwo
 2130 ENDPROC
 2140 :
 2150 DEF PROCtwo
 2160 PRINT;" ";B%;:FOR X=1 TO 7:READ B%
:PRINT;" ";B%;:NEXT
 2170 READ D%:IF D%=-1 THEN PRINT'"No pr
inter character!" ELSE PROCthree
 2180 ENDPROC
 2190 :
 2200 DEF PROCthree:PRINT'"Printer: ";D%
;:FOR X=2 TO T%:READ D%:PRINT;" ";D%;:NE
XT
 2210 PRINT:ENDPROC
 2220 :
 3000 REM Error handling routine
 3010 VDU 3:*CLOSE
 3020 MODE 135
 3030 IF ERR<>17 THEN PRINT:REPORT:PRINT
" at line ";ERL''
 3040 *FX 254,1
 3050 *FX 228,0
 3060 *FX 238,48
 3070 *FX 4
 3080 END
 3090 :
10000 REM CHARACTER DEFINITIONS - digits
, plus and minus
10010 REM ZERO TO SUB0
10020 DATA 244,0,0,0,0,24,36,36,24,6,9,0
,9,6,0,0,0,0
10030 REM ONE TO SUB1
10040 DATA 245,0,0,0,8,24,56,24,60,9,0,3
1,0,1,0,0,0,0
10050 REM TWO TO SUB2
10060 DATA 246,0,0,0,0,48,8,16,60,1,8,17
,10,5,0,0,0,0
10070 REM THREE TO SUB3
10080 DATA 247,0,0,0,48,8,48,8,48,21,0,2
1,10,0,0,0,0,0
10090 REM FOUR TO SUB4
10100 DATA 248,0,0,0,48,80,80,60,16,6,8,
2,29,2,0,0,0,0
10110 REM FIVE TO SUB5
10120 DATA 249,0,0,0,120,64,32,16,96,1,2
4,5,18,0,0,0,0,0
10130 REM SIX TO SUB6
10140 DATA 250,0,0,0,24,32,56,36,24,15,1
6,5,16,7,0,0,0,0
10150 REM SEVEN TO SUB7
10160 DATA 251,0,0,0,60,4,8,16,32,17,2,2
0,8,16,0,0,0,0
10170 REM EIGHT TO SUB8
```

```
10180 DATA 252,0,0,0,24,36,24,36,24,10,1
7,4,17,10,0,0,0,0
10190 REM NINE TO SUB9
10200 DATA 253,0,0,0,24,36,28,4,24,28,1,
20,1,30,0,0,0,0
10210 REM SH0 TO SUPER0
10220 DATA 228,48,72,72,48,0,0,0,0,0,96,
144,0,144,96,0,0,0
10230 REM SH1 TO SUPER1
10240 DATA 229,8,24,56,24,60,0,0,0,72,0,
248,0,8,0,0,0,0
10250 REM SH2 TO SUPER2
10260 DATA 230,48,8,16,60,0,0,0,0,8,64,1
36,80,40,0,0,0,0
10270 REM SH3 IS SUPER3
10280 DATA 231,48,8,48,8,48,8,48,0,0,0,168,0,
168,80,0,0,0,0,0
10290 REM SH4 TO SUPER4
10300 DATA 232,48,80,80,60,16,0,0,0,48,6
4,16,232,16,0,0,0,0
10310 REM SH5 TO SUPER5
10320 DATA 233,120,64,32,16,96,0,0,0,8,1
92,40,144,0,0,0,0,0
10330 REM SH6 TO SUPER6
10340 DATA 234,24,32,56,36,24,0,0,0,120,
128,40,128,56,0,0,0,0
10350 REM SH7 TO SUPER7
10360 DATA 235,60,4,8,16,32,0,0,0,136,16
,160,64,128,0,0,0,0
10370 REM SH8 TO SUPER8
10380 DATA 236,24,36,24,36,24,0,0,0,80,1
36,32,136,80,0,0,0,0
10390 REM SH9 TO SUPER9
10400 DATA 237,24,36,28,4,24,0,0,0,224,8
,160,8,240,0,0,0,0
10410 REM PLUS TO SUPERSCRIPT
10420 DATA 239,8,28,8,0,0,0,0,0,32,0,248
,0,32,0,0,0,0
10430 REM MINUS TO SUPERSCRIPT
10440 DATA 241,0,28,0,0,0,0,0,0,0,32,0,3
2,0,32,0,0,0
10450 :
11000 REM Other character data.
11010 REM Shift-plus CANNOT BE USED
11020 DATA 255,-1
11030 REM Shift-minus to plus/minus
11040 DATA 225,24,24,126,24,24,0,126,0,0
,0,34,0,250,0,34,0,0
11050 REM Slash to division sign
11060 DATA 243,0,24,0,126,0,24,0,0,16,0,
16,68,16,68,16,0,16
11070 REM Shift-slash to minus/plus
11080 DATA 227,126,0,24,24,126,24,24,0,0
,0,136,0,190,0,136,0,0
11090 REM Asterisk to five-pointed star
11100 DATA 238,8,28,127,62,28,62,119,65,
64,34,20,8,240,8,20,34,64
11110 REM Shift-asterisk to six-pointed
star
11120 DATA 254,99,54,28,127,28,54,99,0,1
6,130,84,40,16,40,84,130,16
11130 REM Comma NOT USED
11140 DATA 240,-1
11150 REM Shift-comma NOT USED
11160 DATA 224,-1
11170 REM Full stop to decimal point
11180 DATA 242,0,0,0,24,24,0,0,0,0,0,0,2
4,0,24,0,0,0
11190 REM Shift-full stop to bullet
11200 DATA 226,0,0,56,124,124,56,0,0,0,0
,24,36,24,36,24,0,0
11210 REM Return to return-arrow
11220 DATA 209,12,12,12,44,124,252,120,3
2,4,10,21,8,244,8,240,0,0
11230 REM Shift-Return NOT USED
11240 DATA 193,-1
11250 REM KEY 0 TO block
11260 DATA 180,0,62,62,62,62,62,0,0,0,12
4,0,124,0,124,0,124,0
11270 REM KEY 1 TO lower case alpha
11280 DATA 181,0,1,59,110,102,111,59,0,0
,12,18,0,18,12,18,33,0
11290 REM KEY 2 TO lower case beta
11300 DATA 182,30,51,51,62,51,51,62,96,0
,1,126,128,2,144,108,0,0
11310 REM KEY 3 TO lower case gamma
11320 DATA 183,0,0,102,54,28,24,48,48,0,
12,146,65,34,28,64,0,128
11330 REM KEY 4 TO capital delta
11340 DATA 184,28,20,54,38,99,67,127,0,0
,4,24,100,128,100,24,4,0
11350 REM KEY 5 TO back apostrophe
11360 DATA 185,48,24,12,0,0,0,0,0,40,0,2
54,0,40,0,254,0,40
11370 REM KEY 6 TO greater than or equal
 to
11380 DATA 186,224,56,14,56,224,7,28,112
,68,0,109,0,59,0,22,0,4
11390 REM KEY 7 TO not equal to
11400 DATA 187,3,4,127,24,255,32,192,0,0
,40,2,44,16,40,64,168,0
11410 REM KEY 8 TO less than or equal to
11420 DATA 188,7,28,112,28,7,224,56,14,4
,0,22,0,59,0,109,0,68
```

```
11430 REM KEY 9 TO pi
11440 DATA 189,0,126,36,36,36,36,36,0,96
,0,126,0,96,0,126,0,96
11450 REM End of Character Data
11460 DATA -1
11510 :
12000 REM Printer UDG information and co
ntrol codes
12010 REM Bytes per UDG, Lines per page,
 Number of control characters
12020 DATA 9,66,13
12030 REM Universal Escape sequence head
er, code for "Define new character", pri
nter reset code, code for "Set page leng
th"
12040 DATA CHR$1+CHR$27+CHR$1,Z$+CHR$121
+CHR$1,Z$+CHR$64,Z$+CHR$67+CHR$1
12050 REM Set,Clear default mode (Fully
justified, emphasised, proportionally sp
aced)
12060 DATA CHR$1+CHR$27+CHR$1+CHR$97+CHR
$1+CHR$3+CHR$1+CHR$27+CHR$1+CHR$112+CHR$
1+CHR$1,CHR$1+CHR$27+CHR$1+CHR$97+CHR$1+
CHR$0+CHR$1+CHR$27+CHR$1+CHR$112+CHR$1+C
HR$0
12070 :
13000 REM CONTROL CODES
13010 REM @ is the null character
13020 DATA 0,CHR$0
```

```
13030 REM U gives underlining toggle
13040 DATA 21,Z$+CHR$45+CHR$1+CHR$(U%)
13050 REM B gives boldening toggle
13060 DATA 2,Z$+CHR$(72-B%)
13070 REM Patch screen pound to printer
code
13080 DATA 96,CHR$35
13090 REM Patch screen hash to printer (
UDG) code
13100 DATA 35,CHR$185
13110 REM Patch screen back-apostrophe t
o printer code
13120 DATA 185,CHR$96
13130 REM C gives auto-centralisation
13140 DATA 3,Z$+CHR$97+CHR$1+CHR$1
13150 REM L gives left justification
13160 DATA 12,Z$+CHR$97+CHR$1+CHR$0
13170 REM R gives right justification
13180 DATA 18,Z$+CHR$97+CHR$1+CHR$2
13190 REM J gives full justification
13200 DATA 10,Z$+CHR$97+CHR$1+CHR$3
13210 REM P ejects a page (FORM FEED)
13220 DATA 16,CHR$1+CHR$12
13230 REM D returns to default mode
13240 DATA 4,Y$+CHR$0
13250 REM X clears default mode to print
er original
13260 DATA 24,V$+CHR$0
```

Wordwise User's Notebook (continued from page 39)

the code f1LS3f2. To return to the original spacing use the code f1ES51,48f2. This latter method is not recommended as it will upset the calculation used to determine the optimum line spacing to fill the page.

CE Centre. Provided this code is the first entry on a line it will place the text, which follows it, in the centre of the line. If it is placed after any character or characters on a line, it forces a new line to start and the text which follows it to be placed in the centre of the next line.

ES64 Printer reset code. This code is used to reset the printer to its default values. It is placed at the start of the last line and is the only entry on that line. This command should not be used if 'user defined' characters are in the printer RAM as these will be erased.

The above notes give the reasons why the embedded commands are used in the example letter format. They are not necessarily the primary function of the commands. The default values are the values which Wordwise or the printer use if the relevant code is not included.

If any other Wordwise/Wordwise Plus users would like to contribute to this series then they should contact the Editor.

HINTS and tips HINTS and tips HINTS and tips HINTS and tips HINTS and tips

This month we have an excellent hint for View users - I wonder how many other readers were unaware of this information. More hints and tips for this page are always welcome, and all hints published are paid for.

SEARCH AND REPLACE IN VIEW

Andrew Rowland

Mike Williams is not alone in believing that View cannot replace double spaces (BEEBUG Vol.9 No.8 page 25). This is true for early versions of View, but View 3 can, using the special code ^S. This is particularly useful as View seems to add spaces all by itelf when re-formatting paragraphs, which have to be removed with:

```
CHANGE/^S^S/^S/
```

Note that SEARCH, REPLACE and CHANGE don't like spaces left after them.

Other special codes which can be used are:

^T - Tabs
^R - Carriage Returns (which mark every end of line)
^^ - the ^ character itself
^? - any single character
^Z - soft Returns (inserted by justifying)

The latter enables the safest way to unjustify text:

```
CHANGE/^Z//
```

which is useful when you can't trust FORMAT not to mess up your layout of tables etc.

^? is the "wildcard", which will match any letter (including Tabs, Cariage Returns etc, which can have some surprising results). Interestingly, it can also be used in the replace string to stand for the letter thus matched. For example:

```
CHANGE/S^?T/W^?N/
```

would change SAT to WAN, SIT to WIN, SET to WEN and so on. If more than one is used in the search string, they are taken in turn: for example:

```
CHANGE/S^?^?^?S/H^?^?^?S/
```

would replace SALTS with HALTS and SANDS with HANDS.

TWO INTO ONE ON A B

T.D.Parsons

For those with an EPROM programmer capable of handling 32K EPROMs and a BBC model B, here is a method of getting two "Switchable ROM Images" into one 32K EPROM. using an EPROM type 27256, program it with two images (i.e.high and low). Then do the following: bend up pin 27, solder a 10K resistor between pin 14 and the bent up pin 27. Solder a link between pin 1 and pin 28. Finally, solder a switch between pin 28 and the bent up pin 27.

NOTE: Although we can verify that this information is correct, any readers who undertake this action do so at their own risk.

MAKING MORE OF SIDEWAYS RAM

James Randall

On a Master 128, large amounts of sideways RAM often go unused. Using the function key definitions below it is possible to use two banks of sideways RAM as a simple RAM disc for use when developing programs:

```
*KEY 0 *SRWRITE E00 8000 0|M
*KEY 1 *SRREAD E00 8000 0|M
```

To use these function key definitions, press function key f1 to save any program currently in memory to sideways RAM, and function key f2 to reload the program back into user RAM. However, before you can use these two function keys you should type:

```
*SRDATA 4
*SRDATA 5
```

This routine provides a quick method of temporarily saving the contents of user RAM. Do remember not to switch the computer off without making a disc copy of anything saved to sideways RAM. B

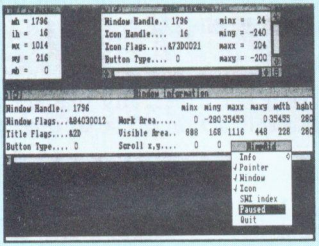
RISC USER

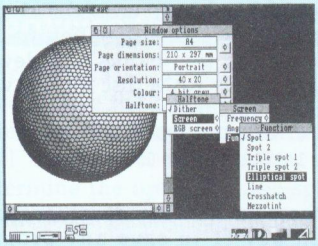
The Archimedes Magazine & Support Group

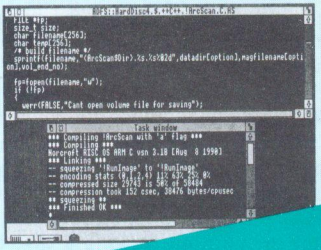
Now in its fourth year of publication, Risc User continues to enjoy the largest circulation of any magazine devoted solely to the Archimedes range of computers. It provides support for all Archimedes users at work (schools, colleges, universities, industry, government establishments) and home.

Existing Beebug members, interested in the new range of Acorn micros, may either transfer their membership to the new magazine or extend their subscription to include both magazines.

A joint subscription will enable you to keep completely up-to-date with all innovations and the latest information from Acorn and other suppliers on the complete range of BBC micros. RISC User has a massive amount to offer to enthusiasts and professionals at all levels.







Here are some articles and series published in the most recent issues of RISC User:

DEBUGGING THE WIMP
A look into the use of a number of readily available tools for debugging Wimp programs written in Basic or C.

SOUNDTRACKER KEYBOARD
A colourful multi-tasking musical keyboard display, driven automatically by the SoundTracker module.

SHOWPAGE
A look into the new PostScript interpreter from Computer Concepts.

ICONBAR SHELL
A useful application which provides the building blocks for creating simple multi-tasking programs without extensive knowledge of the Wimp.

QUITE A REVELATION
A review of Revelation, the sophisticated image processing application from Longman Logotron.

FASTWIPE
A utility which enables you to wipe a floppy disc very efficiently and quickly.

INTO THE ARC
A regular series for beginners currently treating the subject of using Draw for text.

USING ANSI C
A new series of articles on programming Desktop applications in C.

MASTERING THE WIMP
A major and very popular series on the Wimp programming environment.

WP/DTP
A regular column which offers advice on using different DTP and WP packages.

PROGRAMMER'S WORKSHOP
A major series covering all advanced aspects of the Archimedes and incorporating the Assembler Workshop.

ARCADE
An occasional column offering a round-up of the latest games for the Archimedes.

SUBSCRIPTION DETAILS

As a member of BEEBUG you may extend your subscription to include RISC User for only:

Destination	Additional Cost
UK,BFPO &Ch Is	£ 9.10
Rest of Europe and Eire	£14.00
Middle East	£17.00
Americas an d Africa	£19.00
Elsewhere	£20.00

LITHUANIAN RESPONSE

Readers may recall the letter we published in ***Postbag****, BEEBUG Vol.9 No.10 from Kriukelis Saulius, a Lithuanian student, who was seeking help in using a BBC micro which he had obtained.*

As a result, Mr.P.N.Vincenza from Colchester phoned and offered to pay for a year's subscription to BEEBUG for Mr. Saulius. This is a most generous gesture, and a year's subscription has now been taken out in the name of Mr.Saulius.

In addition, a further letter has been received from Mr.Saulius:

Thank you for the magazines which I found very interesting and useful for me. I must say that BEEBUG is much more useful and interesting than BBC Acorn User, especially for BBC owners. Thank you for putting my letter in BEEBUG - I am sure that correspondence with other BBC owners is the only way for me to get some information on hardware and to exchange software. Also, thank you for the BEEBUG indexes and magazines, in which I found several items of importance to me.

I think I am the only BBC B user in Lithuania - other people usually use IBM computers or clones, and to read PC discs on my BBC B is the next task for me. Thank you once more.

Kriukelis Saulius

As a result we have sent some further copies of BEEBUG to Mr.Saulius relating to PC to BBC file transfer, and some photocopies of some now out of print issues relating to CAD in which Mr.Saulius also expressed interest. We are sure that Mr.Saulius would be interested in hearing from other BBC micro owners.

PERSONAL ADS AN EYE OPENER

Through the advert pages in the last issue of BEEBUG I got hold of 8-9 years of Acorn User magazines. Previously I had only known about Beeb history for the last two years - the time I have had my 2nd hand BBC B. My eyes have been opened to a lot of things. I now appreciate BEEBUG (and Watford Electronics) even more than I did before. I can assure you I will be a member and Beeb owner for a very long time to come.

T.D.Parsons

We have always maintained that recent purchasers of a BBC micro will find a wealth of information in back issues. Unfortunately, many of the earliest issues of BEEBUG are now out of print, and our present offer on back issues will terminate on 30th June 1991 (see inside back cover). After that date we shall maintain back issue stocks of magazines and discs only from volume 6 onwards. No more copies of earlier issues will then be available.

DFS BUG IN THE 'NEW' MASTER OS

The 'alternative' operating system ROM for the Master 128 has an irritating bug in that when trying to use DFS from the keyboard after previously using ADFS, a spurious "Disc write protected" error prevents any writing operations from DFS. This condition persists over Break or Ctrl-Break, or any manipulation of ADFS or DFS commands so far as I can discover. However, the problem can be cleared by setting location &C2DE to zero after selecting DFS (?&C2DE=0 in immediate mode).

This appears to offer a practical solution although I am not sure of the mechanisms involved. Fortunately, the bug appears in immediate mode only, and does not seem to affect operations between ADFS and DFS within programs. Maybe someone else can throw more light on this.

C.W.Robertson

We have been unable to reproduce this ourselves - have any other readers encountered this problem? B

Personal Ads

BEEBUG members may advertise unwanted computer hardware and software through personal ads (including 'wants') in BEEBUG. These are completely free of charge but please keep your ad as short as possible. Although we will try to include all ads received, we reserve the right to edit or reject any if necessary. Any ads which cannot be accommodated in one issue will be held over to the next, so please advise us if you do not wish us to do this. We will accept adverts for software, but prospective purchasers should ensure that they always receive original copies including documentation to avoid any abuse of this facility.

We also accept members' Business Ads at the rate of 40p per word (inclusive of VAT) and these will be featured separately. Please send us all ads (personal and business) to MEMBERS' ADS, BEEBUG, 117 Hatfield Road, St. Albans, Herts AL1 4JS. The normal copy date for receipt of all ads will be the 5th of each month.

WANTED: Instructions/manual for BEEBUGSOFTs Program Builder, can copy and return if necessary. Tel. (0602) 723914.

BEEBUG magazines Vols. 1-9 complete - 90 issues the lot £30 plus postage. Tel. (0983) 525036.

512 board for Master with DR DOS 2.1 £90, Music 500 £15, Z80 2nd processor £65, ATPL board with battery backup £30, Softlife numeric keypad £15, Technomatic EPROM blower £30, Microvitec colour monitor £95, Model B with 1770 and Acorn speech system £150, also loads of software available. Tel. (0372) 749678.

Interchart ROM and manual £15. Tel. 081-445 7875.

BBC issue 7, 16 ROM expansion board (sideways), 40T single sided drive, DFS (the original DFS), CUB monitor. Tel. (0935) 27689.

Archimedes 310 base unit, RISC OS, 2 slot backplane, fan, software £500, 40Mb internal hard disc £250. Tel. (0895) 30826.

Archimedes 410/1 upgraded to 440, 4Mb RAM, 40Mb hard disc, NEC multisync monitor, software and games (inc. PC Emulator, DTP, Word Processor and Spellmaster, E-Type, Arcpinball, Quazer, Pacmania) £1700 o.n.o. Tel. (0602) 811600.

65C102 co-processor plus Watford co-pro adaptor £90, Watford Quest mouse & free teletext mouse course £10, BEEBUG Printwise £15. Tel. (0751) 75058 extn.22.

512/1024 (solidisk PC+) board and mouse etc. £120, CMC 20Mb hard disc for Master £220, Morley Advanced teletext decoder & ROM £40, Pace Linnet V21/23 modem £50. Tel. (0454) 311062 eves.

The following ROMs are complete with original manuals; AMX Design and Pagemaker £15 each, Watford Electronics ROMSpell, Beebfont, Wordaid, and NLQ £8 each, Pear Trees The Artist and BEEBUG Help £6 each, Watford's 16k sideways RAM chip £12. Tel. (0993) 841859 after 6pm.

Telesoftware: Can any member please help with copy of the utility to extract data (weather etc.) from Teletext pages daily and plot graphs, described by J Fletcher in BEEBUG 5(3) 1986. Tel. 092-575 4783.

Technomatic twin disc drive 5.25" (suit any BBC) £120, Z80 second processor and manuals (suit any BBC) £130, Master Modem & manual (for BBC Master series only) £110, BEEBUG C ROMs & 18 books (suit any BBC) £110. Tel. 081-961 5735 anytime.

6502 second processor & Hi Basic/DNFS ROMs & 6502 development package £50, AMX Super Art boxed with mouse £25, Fleet Street Editor £15, PCB Design (disc & u.g.) £10, Starbase (ROM, disc & u.g.) £10, Acornsoft "C" (3.5 + 5.25" disc & u.g.) £15, Forth (ROM & manual) £10, ISO-Pascal (2 ROMs, 2 manuals, & disc) £30, Viewsheet (ROM & u.g.) £10, LISP (tape & manual) £5, BEEBCALC (ROM, tape & u.g.) £5. Write to; Mr Gadd, 50 Humber Avenue, South Ockendon, Essex, RM15 5JN.

Tracker Ball for sale, equivalent connections to RB2 style and so plugs into the user port, T/Ball compatible drawing package on 5.25" disc, swap over box to make T/Ball connections equivalent to mouse, all for £15 (collect) or £18 (posted), also 40 column amber 4000 dot matrix printer, bargain at £25 posted, also lightpen with software £5 posted. Tel. (0635) 297701 eves.

Master Compact owners! soft /firmware for sale, PAL TV adaptor, 4 viglen cartridges and holder, Overview (View, Viewsheet, Viewstore, Viewspell, Viewplot, Index, Drivers) Dumpout and Command ROMs, 15 discs inc. originals of Fairy Tales, Typing Tutor, Funschool x 2 (under 5 & 5-8), Little Red Riding Hood, Numbercopter, Nursery Rhymes & manuals, some BEEBUG's and books £60 o.n.o. Tel. (0622) 858476 (Lenham, Kent).

Electron with Plus 1 (+View and Viewsheet ROMs), Plus 3 (+ over 20

Points Arising....Points Arising....Points Arising....Points Arising....

MINI WALL PLANNER (Vol.10 No.1)

The value of &7C00 in line 1720 in the magazine must be altered to &5800. If this is not done, the wall planner will appear correctly on screen, but only garbage will be printed.

MEWS (Vol.10 No.1)

The correct phone number for *Resource* is (0302) 340331.

HINTS AND TIPS (Vol.10 No.1)

In *Automated Macro Entry in View* the second code line should read:

```
*KEY0 "|!$xx|M|!|M"
```

discs containing various programs), Slogger ROM box (+ utility ROMs, Elkman, Toolkit, Addcomm, T2P3), 2 joysticks, various commercial software (Pascal, Repton 3, Elite, Citadel, etc.) All equipment in excellent condition and fully documented. Offers! Tel. (0734) 751875.

WANTED: Has anyone adapted or built a modem to run at 9600 baud and has used the BBC B as a fax? Furthermore any software and ROM images that are not wanted, will pay or swap. Please write to; Mr R Tyrer, 1 Conifer Road, Tokai 7945, South Africa. (There is no support for the Beeb here).

Hybrid Music 5000 synthesiser, Music 3000 16 channel expander and Music 4000 keyboard, Ample ROM, plus manuals, Ample programming guide, system and issue discs and two music discs (Cosmix & Notes) £300. Tel. (0582) 882081 between 10am and 5pm weekdays.

BBC Master 512, mouse, Gem software and manuals, Dabs 512 guide and shareware collection, dual 40/80T drives with PSU in plinth, Microvitec 1451 colour monitor, Panasonic KXP1081 printer hardly used, 2 cartridges, DA computers GDump Dabs MOS+, Viewstore, Spellcheck II, reference manuals 1&2, Dabs Guides to view, viewsheet, viewstore and Master Operating system, book on Assembly Language programming, disc storage box and some blank discs complete system ready to use £650. Tel. (0663) 733092.

FOR SALE: Good copies of the following; BEEBUG Vol. 2 No. 8 to Vol. 7 No. 8 inclusive, RISC User Vol. 1 No. 1 to date. Offers? Tel. (0304) 853267.

A3000 colour system, 1Mb, RISC OS, E-Type, Fun School 2, as new £500. Tel. 081-360 8076.

Magic Modem and Command (auto-dial/answer) £25, Ricochet £4, AMX Pagemaker £12, Accelerator (BASIC compiler) £10, Acorn User seven disc compilation of utilities and games £10, P.I.A.S. 13 £5, the whole lot for £30. Tel. (066479) 525.

Archimedes 400/1 with 2Mb RAM, hard drive and scanner together with original software (Impression, First Word Plus, ANSI C and others), half price or less, will split. Tel. (0923) 855853 after 6pm.

512 co-processor, in Watford co-pro adaptor (runs well with BBC B), Essential Software 1Mb memory expansion, Including mouse and software GEM, Essential's RAM disc and other utilities inc. BBC Basic £175 the lot. Tel. 051-228 2402 anytime.

Acorn A3000 computer, Acorn 2Mb memory upgrade, Philips CM8833 stereo colour monitor, Zarch, UIM, White Magic, Soccer (immaculate) £700. Juki 6100 Daisywheel printer, two daisywheels, spare ribbon, only light home use £150 o.n.o. Tel. 051-606 0289.

WANTED for ELECTRON: Plus 3 add-on, single disc drive and DFS. Tel. (0322) 525652.

Master 128, 16" monitor/TV, LC10 printer, Cumana CSX 40/80T disc, cassette recorder, A.A. board, ROMs, BEEBUG Master, ADT, ADI, Hyperdriver, Brom Plus, Floppywise plus, publisher, Wordpower, MOS Plus, PMS Multi font, Edikit, 3 ROM cartridges, Mastering View and disc, Dabs Press View and disc, sidewriter, Anchor, all original manuals £600 cash buyer collects. Tel. (0924) 372003.

BBC issue 7, Acorn DFS/DNFS, ROM board, dual 40/80 drives, Microvitec 1431 colour monitor, Acorn 6502 second processor with Hi BASIC and manual, Dabs Hyperdriver, Mini Office II, Dabs Guide to Mini Office II, Fleet Street Editor, Edword, Wordwise Plus, Wordaid, Slave+, User Dump, Keyword, some games, twin joysticks, several books on Assembly Language etc. disc storage box with some discs, Solidisk DDFS but no manual £400. Tel. (0663) 733092.

M128 with Viglen 40/80T DS DD, twin joysticks, welcome and reference manuals, leads £250 + delivery or buyer collects. Tel. 081-949 6625 eves.

WANTED: A copy of BBC BASIC for Beginners by David Smith - Melbourne House Publications (ISBN 0-86161-126-8). Tel. 081-777 1887.

Brother HR35 daisywheel printer, fast (40 cps), superb quality, excellent condition, serial connection, complete with supply of high-quality carbon ribbons, two print wheels and serial lead for BBC, cost over £800 new, bargain at £275. Also 100 computer cassettes C15/20, branded, unused £20. Tel. (0438) 833575 eves.

Modem: Tandata TD1400, 1200/75bps operation, with manual, little used £50. Tel. (0928) 722454.

BBC Master 128, user guide, reference manuals 1&2, original introductory disc, all leads included, Acorn colour monitor (like Philips 8833) RGB cable included, 80T DS DD (no PSU) 40/80T DS DD (no PSU) power and data duct cabling for these items, Acorn joysticks, 512k add-on board including, DOS+ 2.1, GEM software, mouse etc. Master 512k User Guide and software, Master 512k Shareware Vol.1, Smith Corona D200 dot matrix printer - FX80 compatible but with NLQ font as standard, (draft 160 cps, NLQ 40 cps) parallel cable to BBC included, Viewstore, Pascal (for BBC), Comal (for BBC), Logo (for BBC), BEEBUG Spellcheck III, Database (for BBC), Fleet Street Editor, AMX Pagemaker, ROM cartridges 2 by 2 slot, 1 by 4 slot (for BBC Master), BEEBUG Vol. 1 to date, 2 x 16k EPROMs - 27128-25. Tel. (0634) 710972 after 6.30pm for details.

A310, 4Mb upgrade, MEMC1a upgrade, 4-slot IFEL backplane with fan, 2nd internal 3.5" drive, switched external 5.25" drive interface, VIDC enhancer, all boxed complete with manuals, software, cables etc. excellent condition £1050 o.v.n.o. **A540 as new**, boxed complete with manuals, software, cables etc. Offers over £3000. Tel. (0225) 428662.

BBC model B, issue 7 computer, fitted with Watford Shadow RAM, Watford DFS and View (version B3) word processor, complete with all manuals £210 inclusive of carriage. View Professional word processor /spreadsheet £30, Watford Electronics "Adder" EPROM programmer £35. Tel. (0481) 45866 extn 351, 1-4pm.

Integrated circuit tester, tests all digital IC's and displays waveforms on the screen, plugs into user and 1MHz bus sockets on BBC B, 128 or Master, great for experiments, schools etc. c/w handbook and software £30. Tel. (0635) 297701.

WANTED: Interword word-processor for BBC. Tel. (0463) 83 658 after 5pm.

BEEBUG MEMBERSHIP

Send applications for membership renewals, membership queries and orders for back issues to the address below. All membership fees, including overseas, should be in pounds sterling drawn (for cheques) on a UK bank. Members may also subscribe to RISC User at a special reduced rate.

BEEBUG SUBSCRIPTION RATES

BEEBUG		BEEBUG & RISC USER
£18.40	1 year (10 issues) UK, BFPO, Ch.I	£27.50
£27.50	Rest of Europe & Eire	£41.50
£33.50	Middle East	£50.50
£36.50	Americas & Africa	£55.50
£39.50	Elsewhere	£59.50

BACK ISSUE PRICES (per issue)

Volume	Magazine	5"Disc	3.5"Disc
5	£0.50	£2.00	£2.00
6	£0.50	£2.00	£2.00
7	£1.00	£2.00	£2.00
8	£1.20	£3.00	£3.00
9	£1.60	£4.75	£4.75
Binders	£4.20		

All overseas items are sent airmail. We will accept official UK orders for subscriptions and back issues, but please note that there will be a £1 handling charge for orders under £10 which require an invoice. Note that there is no VAT in magazines.

POST AND PACKING

Please add the cost of p&p when ordering individual items. See table opposite.

Destination	First Item	Second Item
UK, BFPO + Ch.I	£1.00	£0.50
Europe + Eire	£1.60	£0.80
Elsewhere	£2.40	£1.20

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 40303, FAX: (0727) 860263
Manned Mon-Fri 9am-5pm
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: Alan Wrigley
Technical Assistant: Glynn Clements
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

We are always seeking good quality articles and programs for publication in BEEBUG. All contributions used are paid for at up to £50 per page, but please give us warning of anything substantial that you intend to write. A leaflet 'Notes to Contributors' is available on receipt of an A5 (or larger) SAE.

Please submit your contributions on disc or cassette in machine readable form, using "View", "Wordwise" or other means, but please ensure an adequate written description is also included. If you use cassette, please include a backup copy at 300 baud.

In all communication, please quote your membership number.

Printed by Arlon Printers (0923) 268328 ISSN - 0263 - 7561

Magazine Disc

June 1991
DISC CONTENTS

THE GAME OF BLOCKADE - an entertaining board game for two players, one of which can be the computer, where the aim is to move your counters in such a way that you block the opponent's move.

ROUTE PLANNER REVISITED - an enhanced version of the Route planning program (Vol.10 No.1) which caters for ADFS and allows a choice of pricing in litres or gallons; and a second program for creating route files. Some sample route files are also included.

RECREATIONAL MATHEMATICS: MODULAR ARITHMETIC FOR PLEASURE - this month's program 'Fermat' allows modular arithmetic to be carried out with moduli as big as a billion.

VOLUME CONTROLLER - a short program which provides variable volume control for sound on your computer.

BEEBUG WORKSHOP: NUMERICAL INTEGRATION - a program with some useful practical applications which helps you calculate the area under a particular curve and displays visually the way this is done.

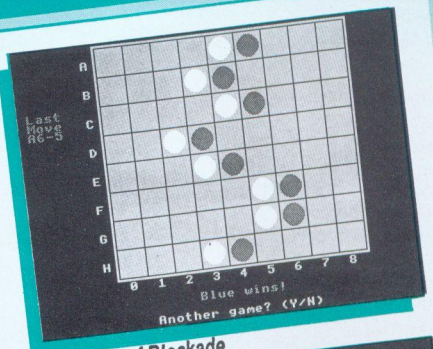
MERGE, PART-MERGE AND PART-SAVE - this assembly language utility offers two star commands *PMERGE and *PSAVE which allow parts of a Basic file from a disc to be merged into a program in memory, or parts of a program in memory to be saved to disc.

BEEBUG FUNCTION/PROCEDURE LIBRARY (PART 4) - a Basic program containing the latest collection of functions and procedures from this series.

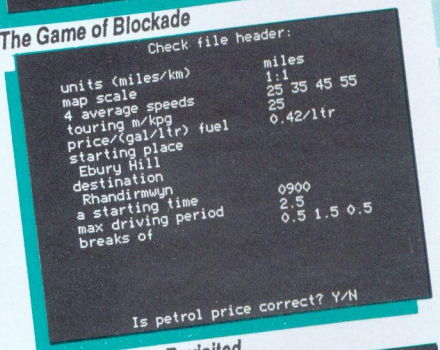
PRINTING SCIENTIFIC CHARACTERS WITH WP (2) - this month's program allows you to design scientific characters for use with Edit, the Master's built-in text processor.

MAGSCAN DATA - bibliography for this issue (Vol.10 No.2).

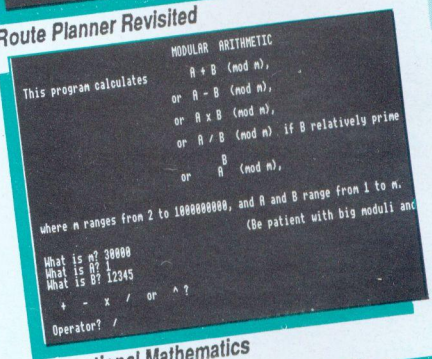
version of this popular jigsaw puzzle.



The Game of Blockade



Route Planner Revisited



Recreational Mathematics

ALL THIS FOR £4.75 (5.25" & 3.5" DISC) + 60P P&P (30P FOR EACH ADDITIONAL ITEM)
Back issues (5.25" disc since Vol.3 No.1, 3.5" disc since Vol.5 No.1) available at the same prices.

DISC (5.25" or 3.5") SUBSCRIPTION RATES	UK ONLY	OVERSEAS
6 months (5 issues)	£25.50	£30.00
12 months (10 issues)	£50.00	£56.00

Prices are inclusive of VAT and postage as applicable. Sterling only please.

BEEBUG, 117 Hatfield Road, St.Albans, Herts AL1 4JS

Special Offers to BEEBUG Members June 1991

BEEBUG'S OWN SOFTWARE

Code	Product	Members Price inc Vat
1407a	ASTAAD3 - 5" Disc (DFS)	5.95
1408a	ASTAAD3 - 3.5" Disc (ADFS)	5.95
1404a	Beebug Applics I - 5" Disc	4.00
1409a	Beebug Applics I - 3.5"Disc	4.00
1411a	Beebug Applics II - 5" Disc	4.00
1412a	Beebug Applics II - 3.5" Disc	4.00
1405a	Beebug Utilities - 5" Disc	4.00
1413a	Beebug Utilities - 3.5" Disc	4.00
1450a	EdiKit 40/80 Track	5.75
1451a	EdiKit EPROM	7.75
1452a	EdiKit 3.5"	5.75
0005b	Magscan Vol.1 - 8 40 Track	12.50
0006b	Magscan Vol.1 - 8 80 Track	12.50
1457b	Magscan Vol.1 - 8 3.5" ADFS	12.50
0011a	Magscan Update 40 track	4.75
0010a	Magscan Update 80 track	4.75
1458a	Magscan Update 3.5" ADFS	4.75

Code	Product	Members Price inc Vat
PAG1a	Arcade Games (5.25" 40/80T)	5.95
PAG2a	Arcade Games (3.5")	5.95
PBG1a	Board Games (5.25" 40/80T)	5.95
PBG2a	Board Games (3.5")	5.95
1600a	Beebug magazine disc	4.75
0077b	C - Stand Alone Generator	14.56
0081b	Masterfile ADFS M128 80 T	16.86
0024b	Masterfile DFS 40 T	16.86
0025b	Masterfile DFS 80 T	16.86
0074b	Beebug C 40 Track	45.21
0075b	Beebug C 80 Track	45.21
0084b	Command	29.88
0073b	Command(Hayes compatible)	29.88
0053b	Dumpmaster II	23.76
0004b	Exmon II	24.52
0087b	Master ROM	29.88
1421b	Beebug Binder	4.20

OTHER MEMBERS' OFFERS

These offers are available for a limited period only. To avoid disappointment please order early, as offers are only available while stocks last and demand is always high. Orders are dispatched on a first come first served basis. To order phone (0727) 40303 or write to BEEBUG, 117 Hatfield Rd, St.Albans, AL1 4JS.

WHITE KNIGHT MK2 The Chess Master

£13.50 (inc VAT) + p&p. *Existing price kept for 6 weeks only!*
REVISED Members price £16.95 (ex VAT)

"White Knight is by far the best computer chess program available for the BBC micro and the Master 128." *BBC Publications*

Stock code 1125b

SPEECH!

£7.95 (inc VAT) + p&p
RRP £11.95

SPEECH! from Superior Software is a program which enables your computer to speak words you input or text files.Stress and intonation can be added and even foreign languages can be spoken.

Stock code 5541b

Post & Packing	UK	Europe	Americas, Africa Middle East	Elsewhere
a	£ 1.00	£ 1.60	£ 2.40	£ 2.60
b	£ 2.00	£ 3.00	£ 5.00	£ 5.50
c	£ 3.10	£ 6.50	£10.50	£11.50
d	£ 3.60	£15.50	£24.50	£25.50

ViewSheet

£12.77 (inc VAT) +p&p
Normal Members price £42.49 (inc VAT)

Excellent spreadsheet for the BBC micro and Master, supplied on a 16K ROM. A very powerful and extremely useable product which will produce results ready to print or for direct merging into View text files.

Stock code 1001b

ViewSpell

£7.16 (inc VAT) + p&p
Normal Members price £28.75 (inc VAT)

An automatic spelling checker with a built-in 75 000 word dictionary. Supplied on ROM with full manual, examples disc and reference card. Ideal for View or ASCII text files, and can use updateable user dictionaries.

Stock code 1043b

ViewStore

£14.10 (inc VAT) + p&p
Normal Members price £42.49 (inc VAT)

ViewStore is Acorn's database manager program. It offers 40 and 80 column display, multiple indexes, detailed report generation, variable field lengths and much more. Excellent value for money.

Stock code 1019b