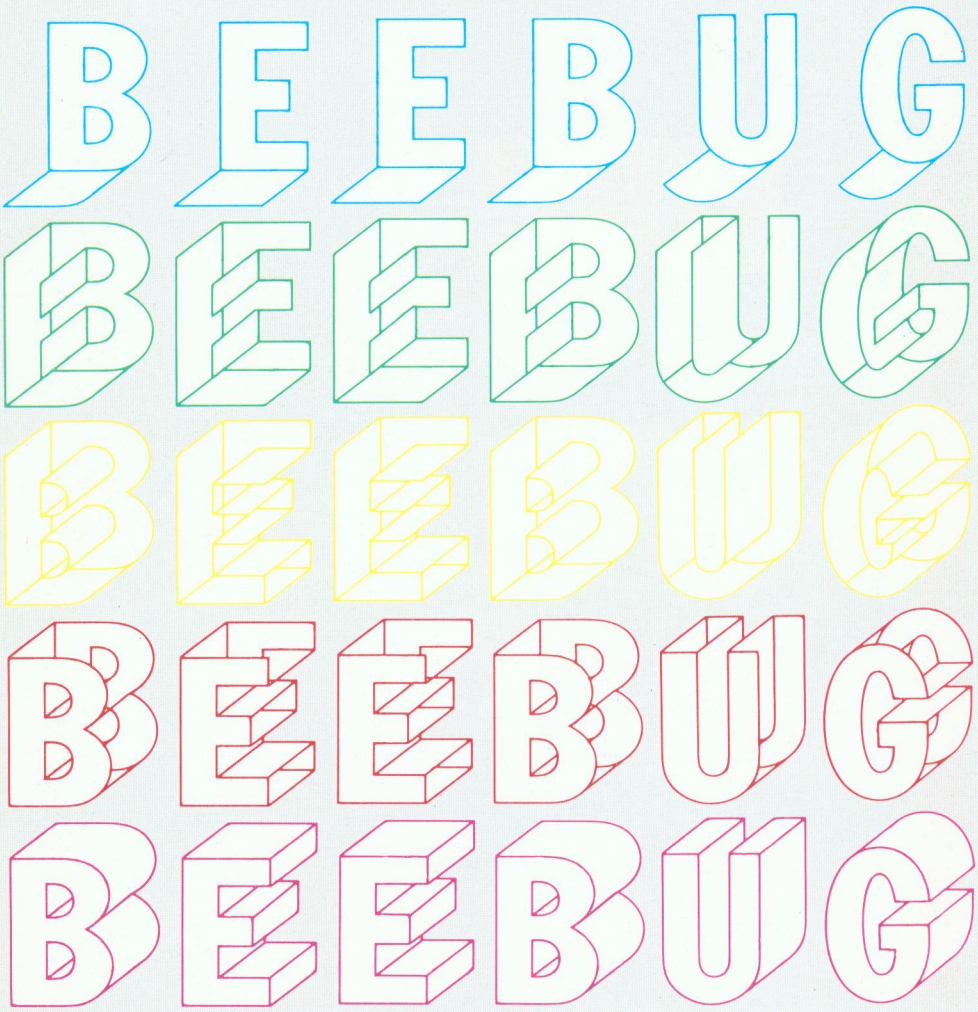
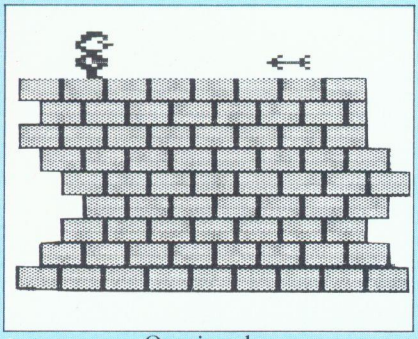
VOLUME 4 NUMBER 6 NOVEMBER 1985 PRICE £1.20

BEEBUG

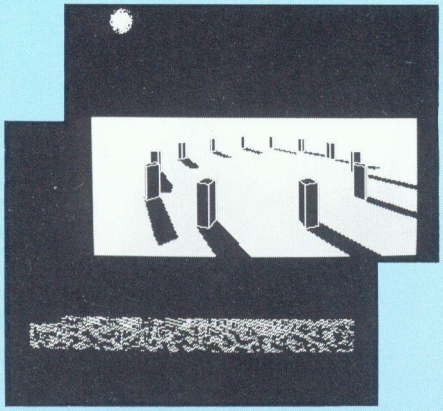
FOR THE BBC MICRO

HIDDEN LINE REMOVAL

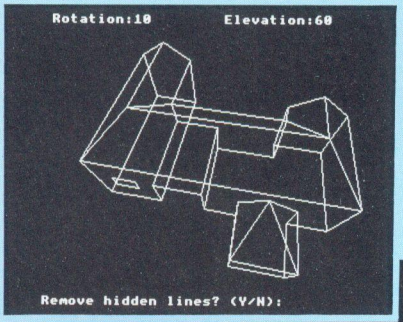


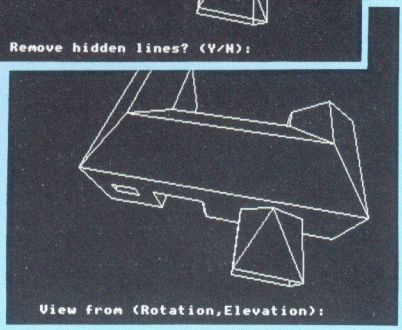


Quasimodo



Picture Compression





Hidden Line Removal

BEEBUG

VOLUME 4 NOVEMBER 6
NOVEMBER 1985

GENERAL CONTENTS

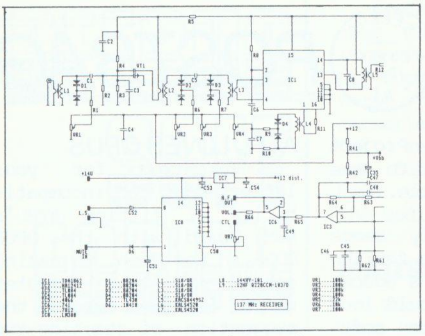
3 Editorial Jottings
4 Postbag
5 BEEBUGSOFT Forum
6 News
7 Build Your Own Database Manager
BEEBUG Filer – a database for all
10 Compressing Screen Displays
12 Wordwise Plus Program Compactor
15 Writing Your Own Compiler (Part 1)
20 ROM Utilities for All
23 Programming with Wordwise Plus (Part 2)
26 Hidden Line Removal
31 Computer Concepts Speaks Out
32 BEEBUG Workshop
Virtual Arrays
34 Viewstore
37 Programming Sideways RAM and ROM
42 First Course
Print Formatting (Part 2)
44 Generating Diagrams and Drawings
46 Quasimodo

PROGRAMS

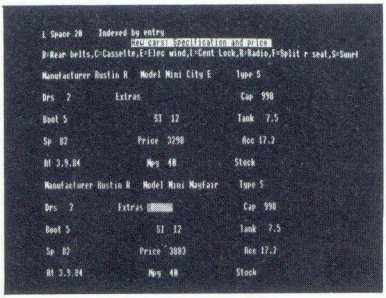
7 BEEBUG Filer
10 Picture Compression
and Decompression
12 Wordwise Plus Program Compactor
15 Basic Compiler (Part 1)
23 Wordwise Plus Examples
26 Hidden Line Removal
32 Workshop Procedures
37 Programming ROM and RAM
42 First Course
Print Using Function
46 Quasimodo Game

HINTS, TIPS & INFO

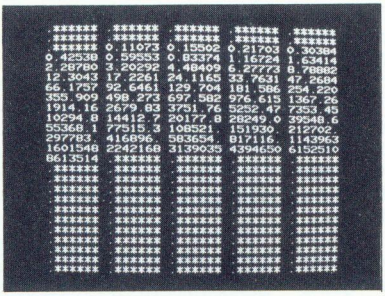
14 Filling Up
25 Easier Bytes
41 *FX Values
41 Quick Lines
41 Default Vector Values



Diagrams and Drawings

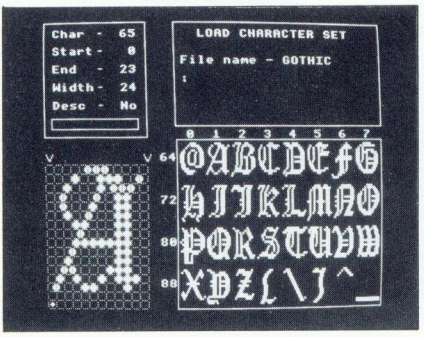


Viewstore



First Course

ROM Utilities



EDITORIAL JOTTINGS

You will again find more helpful information on BEEBUGSOFT products under the heading BEEBUGSOFT Forum. This month we give details of two useful routines to use with Wordease and Hershey Characters. The code for both new routines is included on the magazine cassette/disc. We expect to publish further useful routines for BEEBUGSOFT programs in this way in the future.

Another BEEBUGSOFT package which has proved very popular with members since its launch is Magscan, the disc-based bibliography for all issues of BEEBUG magazine. When published, Magscan was complete up to the end of volume 3. Starting with this issue we shall be including each month on the magazine cassette/disc, the complete Magscan entry for that issue. In this way you will be able to keep your Magscan bibliography up to date as each new issue appears. To catch up with volume 4, all the Magscan entries for issues 1 to 5 (first half year) are available separately on disc.

Another special offer for BEEBUG members is the Computer Concepts Speech ROM reviewed in this issue. This is available to members at only £31.00 including VAT and p & p. Full order details in the supplement.

MICRO USER SHOW

We hope to see as many of you as possible at the Micro User Show (stand 80) from the 14th to 17th November at the New Horticultural Hall, London. We shall not be attending the recently arranged Acorn User Show the following week. In our view this new show is unnecessary at this time and not in the best interests of BBC micro users. Over the last two years the Micro User Show has established itself as THE pre-Christmas show for the Beeb.

PROGRAM CLASSIFICATION

All programs in the magazine, and on the magazine cassette/disc, are marked with the symbols shown below. An unmarked symbol indicates full working, a single line through a symbol shows partial working (normally some modifications will be required), and a cross through a symbol indicates a program that will not work on that type of system. There is also a symbol for the B+ which includes the 128K version.

Basic I	I	Electron	
Basic II	II	Disc	
Tube		Cassette	
Model B+	+		

POSTBAG

SHEILA IN TROUBLE AGAIN

B.D.Cocksedges' letter, which you published in the August/september issue, has solved a problem which has been frustrating me for some time now ever since I first tried out the RS423/ cassette programs in the Advanced User Guide without success. Also, Amcom's game "Space Hi-Way", in which the main section loads via the RS423 interface, refused to work. On checking I found that I too have the special Acorn chip. So far all attempts to obtain a replacement Ferranti chip have proved unsuccessful.

However, changing the *FX7,3 and *FX7,4 commands to *FX7,8 in the A.U.G. programs enables them to work perfectly, while changing the m/c byte in the Amcom game at &5211 to &81 (setting bits 3, 4 & 5 to zero) has cured the loading problem, enabling me to appreciate this as one of the best games I have seen of its type.

D.Kerr Jamieson

Acorn state that they are unable to exchange chips as both meet their specification, and that any problems that may arise from using undefined values are the responsibility of the software writer.

HARDLY SOFTWARE

I am a radio amateur (G3UDA), and it is quite incredible the number of Beebs being used on the air for slow scan TV, reception of Meteosat & RTTY as well as the 'common or garden' station logbook. I particularly enjoyed the latest EPROM Programmer Project and look forward to the follow-up RAM article.

The Beeb really seems the serious machine for the hobbyist/scientist & educationalist so I would look forward to any more software/hardware projects in this field.

Ken Linney

ACORN CLAIM COMPATIBILITY

Following your table showing the compatibility of various ROMs with the B+ in Vol.4 No.3, I list below the Acornsoft ROMs with their usability on the B+.

View 1.4	****
View 2.1	***(*)
Viewsheet	****
ViewStore	****
Logo	****
BCPL	****
Basic Editor	****
Forth	****
LISP	****
Comal	****

All the Acornsoft ROMs can take advantage of the shadow memory, which is not generally true of the four star ROMs in the review. View 2.1 will work correctly and take advantage of the extra memory using the patch routine from Acorn.

Rob Macmillan
Acornsoft

We were perhaps a little harsh, on reflection, in our original classification of View 1.4. The Acornsoft patch referred to for use with View 2.1 is included on this month's magazine cassette/disc. We can also provide a photocopy of the listing on receipt of an A5 SAE (marked VIEW PATCH).

ATPL LOVES OPUS

Thank you for your letter about the incompatibility of fitting ATPL's ROM Board with the OPUS DDOS in the Beeb. On asking around my local computer shops I came across the suggestion of stacking two 40 pin sockets under the OPUS DDOS Board. This is not, as they say, very good if your Beeb is always on the move, but I have found no problems so far.

G.A.Smith

BEEBUGSOFT MAGSCAN

I use your Masterfile and the BEEBUG bibliography utility Magscan and find both excellent. I was wondering whether it was on the cards for you to publish in BEEBUG or on disc a bibliography program that can be customised to user requirements.

C.Dunn

There has been considerable interest in Magscan, and as detailed in this month's editorial jottings, we are now including a Magscan update on each magazine cassette/disc. There is no reason, in principle why you should not create other bibliographies to use with the same program, though you are restricted to the basic Magscan format. For the best results, you are recommended to delete lines 1260 to 1280 and change line 1290 to read IF I%>0 AND... Then call your new datafiles VOL1, VOL2 etc and use either Wordwise or View to prepare entries as described in the Magscan manual.

BEEBUG SOFT FORUM

WORDEASE Canon/Kaga Printer Codes

One of the attractive features of Wordease is its ability to insert printer codes into a Wordwise Plus file at the touch of a key. The routines are supplied with codes for Epson printers as a default, and the manual explains how to insert codes for other printers.

On this month's magazine cassette/disc you will find the necessary modifications to use this Wordease routine with the popular Canon PW1080 and Kaga KP810 printers. Our thanks to Brian Quentin of Chelmsford for sending in this modification.

HERSHEY CHARACTERS

We must again thank one of the members - Peter Miller from Ilford - for adding a most useful facility to one of our products. Peter has sent in a modification to the Hershey Characters program which enables this package to dynamically load in the Hershey character data as and when it is required by the user's program. This allows wider choices of characters and even larger programs to be written.

Unfortunately, the program modifications are too large to be described here, but they are included on this month's magazine cassette/disc along with full instructions on their use. Tube users should set PAGE to &1900 before using the modified program.

MAGSCAN

The BEEBUG Bibliography Disc (Magscan) has proved to be of great interest and service to many members. Thank you for the many letters that we have received on the subject.

As sold, the Magscan disc provides a full index of BEEBUG hints, tips, reviews, programs and all the other articles for volumes 1, 2 and 3 of the magazine. It also includes details on how to update the disc for volume 4.

Starting with this issue of BEEBUG, we are including the index file for each month's issue of BEEBUG, on that month's magazine cassette/disc. You will find this month's Magscan index contained in the file MSCN406.

If you wish to bridge the gap between the first three volumes and this issue, the Magscan entries for Vol.4 Nos.1-5 are available on disc from BEEBUGSOFT at a cost of £4.50 plus 50p postage.

DOUBLE DENSITY

Every month we receive a large number of letters from members who have purchased a double density filing system of one make or another, and who are disappointed to find that many commercially produced programs will not run on them.

This is often the case, even in so called 'Acorn Compatible' modes. Contrary to popular thought, it is often nothing to do with software protection. Frequently users find that any program using random access, such as a database, disc utility program or spelling checker, will not run on their double density system.

Our advice to potential purchasers of double density systems would be to think very carefully before committing yourself to any non-standard equipment.

Of all the makes that we have tested, only the Watford DDFS 1.53 and Opus DDOS 3.45 double density disc systems work fully with Masterfile. If you wish to find out more about double density compatibility with any of our programs, please telephone us on the BEEBUGSOFT hotline - 0727-40303.

EXHIBITIONS

We will be exhibiting the complete range of BEEBUGSOFT products at the London Micro User Show on the 14th to 17th of November. The show is at the New Horticultural Hall in Westminster and BEEBUGSOFT will be there to demonstrate the new range of BBC products. We look forward to the opportunity of meeting members at what looks like being a most exciting exhibition.

Please note that this will be the only exhibition that we will be attending before Christmas.

News News News News News News....

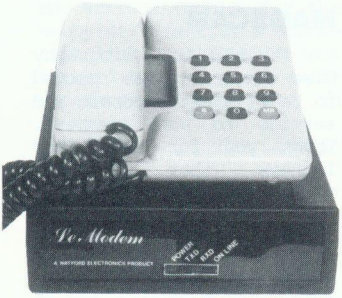
B DOUBLE PLUS

Acorn has been busy and released three new products at the PCW show at Olympia. First and foremost is the 128K model B+. This is based on the same circuit board as the 64K B+ but with the addition of a 64K RAM module. The extra RAM is in the form of four 16K paged RAM banks. The new machine has a full 64K available for Basic and costs £499. Owners of the 64K B+ can upgrade their machines to 128K with an upgrade kit that will cost in the region of £25.

Owners of the model B can also now upgrade their machines to imitate at least one function of the B+. The 1770 disc upgrade kit costs £49.95. The kit consists of a small circuit board containing a 1770 disc controller chip, as used in the B+, instead of the 8271 that the old Beeb is designed to take, and a new DFS which includes format, 40/80 switching, and other commands. For a further £29.95 you can add the ADFS upgrade kit. This will give any Beeb fitted with a 1770 the Advanced Disc Filing System used in the ABC micros and the Electron Plus 3. Acorn, as ever, is on 0223-214411.

LE WATFORD

Watford Electronics has a new modem to tempt you with. Le Modem is a £102 package comprising an auto-answer, auto-dial, auto-baud selection multi-standard modem to connect to the 1MHz bus, along with software on ROM to provide easy access to Prestel, BT Gold, The Source and other international systems, bulletin boards, and so on. Terminal emulation is also included. Needless to say, Le modem has not (yet) received BT approval.



Watford has also released a new version of its ROM expansion board. The new board holds just eight ROMs but can also take eight banks of 16K RAM to load sideways software into. Prices start at £44 for just 16K of RAM fitted and rise to £125 for a board with the full 128K. Watford Electronics is on 0923-37774.

NEW BOOKS

The 'BBC Micro and Electron Book' (£11.95 from McGraw-Hill) is not going to turn any heads for originality of subject matter. However, this large volume does seem to pack in more everyday data and useful routines than most others. Another useful reference work is the '6502 Reference Guide' from Melbourne House (£9.95). Written for all computers that use this processor, the guide contains information on the timing, instruction set, addressing modes, hints, routines, and lots more.

If the BEEBUG spreadsheet (Vol.4 No.s 9 and 10) impressed you but you still don't know what to do with one, then 'Computer Spreadsheets' (£6.95 form BBC soft) may be the answer. The book covers small spreadsheets such as Ultracalc 2 and also the more professional Visicalc and Lotus 1-2-3.

Following on from the reviews of interfacing books in BEEBUG Vol.4 No.5) yet another book on this subject has appeared. 'Microcomputer interfacing' from Collins (£8.95) is aimed mainly at students of computer science, though, it contains much to appeal to any Beeb owner.

NEW SOFTWARE

The software scene seems to be having a summer recess at the moment. However, a few new faces have appeared. Martech, the champion of the armchair sportsman have released 'Geoff Capes Strongman' for £8.95. A little different is 'Zoids' (£8.95) also from Martech - a zap the aliens game based on the Tomy Zoid toys. BBC Soft has released yet another version of its chess program. 'White Knight 12' is an enhanced version of the game, on disc only, for £13.25 direct from BBC Soft. Described as, 'as epic as the Norse sagas and as tiresome as any book by L. Ron Hubbard', Silver Soft has produced an adventure based on the spoof book 'Bored of the Rings'. Followers of Fordo, Aragant, Giblet, and Grandalf can join their quest for £6.95.

Build your own Database Manager Beebug Filer

— a database for all

Editor Mike Williams launches an exciting new software project for all BEEBUG readers in which he describes how to build your own flexible database program.

One of the major uses of computers today is the storing of large quantities of information, and many home micro users find this an attractive and worthwhile use for their micro. To match this interest we are presenting our own BEEBUG database.

Beginning this month, we present a disc based database system that will meet many practical needs. We start with the basic program, showing how to create a simple database file. Further articles will add to the basic program to provide selective retrieval of records, formatted output, sorting and other facilities. The program will be entirely in Basic, and we will show you how to modify and extend the program to cater better for your own requirements if you wish.

Of course, a program published in a magazine cannot hope to provide all the features and power of commercial packages, like our own Masterfile and Acornsoft's ViewStore (reviewed in this issue), but for those considering purchase, our own program will provide a useful testbed for your ideas, before parting with your cash.

THE SOFTWARE PROJECT

The program listed this month is complete and fully working as far as it goes. It provides the basic facilities for creating a database file of your choice, entering, displaying and deleting records. The program is written so that further functions can be readily added and existing functions improved as may seem desirable for this project. Although we have planned the additional features to be added to the program, we would welcome any feedback or ideas, and the best of these may be incorporated in a further article.

The database has been designed specifically for disc systems allowing direct, rather than just sequential, access to records. The program is command-driven, rather than menu-driven. Although menu-driven programs are very popular, commands provide much more direct and efficient ways of controlling an application, once you have learnt their meanings.

The program is designed for mode 3 to allow the maximum screen space for displaying records and other information. The screen is divided up into two main areas, the principal section for presenting records and other information, with four lines reserved at the bottom of the screen for commands and other user responses. The initial set of commands and their functions are as follows:

CREATE	Create a new file on disc
OPEN	Open an existing file for use
CLOSE	Close a file in use
ADD	Add a new record to the file
DISPLAY	Display records from the file
DELETE	Delete a record from the file
END	Close any file in use and exit

All commands can be abbreviated to a minimum of two letters, and may be entered in upper or lower case. Some commands can use parameters as described below. Parameters must always be separated from the command (or its abbreviation) by a single space. All the usual *commands may be used as well.

FILE MANAGEMENT

The first three commands listed above are concerned with file management. Both CREATE and OPEN require a filename to be specified. When you use CREATE to set up a new file you will need to specify a file description as follows:

Number of records
Number of fields per record
Fieldname For each
Fieldwidth field

You will have a chance to either confirm or reject your file description.

RECORD MANAGEMENT

Records are entered using the ADD command. An empty record is displayed on the screen and is filled in field by field before you confirm whether it is to be written to disc. All records may be displayed using the DISPLAY command, or any individual record displayed by specifying its record number. Similarly a record to be deleted is displayed on the screen so that you can confirm your action. Deleted records initially remain in the file and appear as blank records on the screen. However, when you close the file or exit from the program, the file is compacted to lose the deleted records.

Next month we will give a more detailed description of the file structure and the program as well as adding further to the existing program. When you enter the program please keep to the line numbering given so that later additions will fit with no problems.

BASIC I

Basic I users should replace OPENUP by OPENIN at lines 2860 and 3260.

```
   10 REM BEEBUG FILER version B1.9
   20 REM Author Mike Williams
   30 REM BEEBUG November 1985
  100 MODE3:ON ERROR PROCerror:END
  120 PROCsetup:PROCtitle:PROCwindow2
  140 REPEAT:PROCcommand:UNTIL exit%
  165 PROCclose:VDU26,12:*FX4,0
  170 END
  180 :
 1000 DEF PROCsetup
 1020 LOCAL I:exit%=FALSE:open%=FALSE
 1040 maxf=12:X=0:Y=0:*FX4,2
 1060 DIM com$(20),record$(maxf),field$(
maxf),width%(maxf),os 40
 1080 READ N
 1100 DATA 7
 1120 FOR I=1 TO N:READ com$(I):NEXT I
 1140 DATA CREATE,OPEN,CLOSE,ADD,DISPLAY
 1160 DATA DELETE,END
 1180 ENDPROC
 1200 :
 1300 DEF PROCtitle
 1320 PRINTTAB(28,1)"B E E B U G   F I L
 E R"
 1340 PRINT STRING$(80,"_")
 1360 PRINTTAB(0,20)STRING$(80,"_")
 1380 ENDPROC
 1400 :
 1500 DEF PROCcommand
 1520 LOCAL command$,pm$
 1540 REPEAT
 1560 INPUT"-> " command$
 1580 C=FNvalidate(command$)
 1600 IF C=0 THEN PRINT"Unrecognised com
mand"
 1620 UNTIL C
 1630 IF C=1 THEN PROCcreate(pm$)
 1640 IF C=2 THEN PROCopen(pm$)
 1650 IF C=3 THEN PROCclose
 1660 IF C=4 THEN PROCadd(pm$)
 1670 IF C=5 THEN PROCdisplay(pm$)
 1680 IF C=6 THEN PROCdelete(pm$)
 1690 IF C=7 THEN exit%=TRUE
 1870 IF C=99 THEN PROCstar(pm$)
 1880 ENDPROC
 1890 :
 2200 DEF FNvalidate(C$)
 2220 LOCAL found%,I,P:found%=0
 2240 FORI=1 TO N
 2260 IF (ASC(MID$(C$,1))AND223)=ASC(MID
$(com$(I),1)) AND (ASC(MID$(C$,2))AND223
)=ASC(MID$(com$(I),2)) THEN found%=I
 2280 NEXT I
 2300 P=0:IF C$>"" THEN P=INSTR(C$," ")
 2320 IF P=0 THEN pm$="" ELSE pm$=MID$(C
$,P+1)
 2340 IF ASC(C$)=42 THEN pm$=MID$(C$,2):
found%=99
 2360 =found%
 2380 :
 2500 DEF PROCcreate(p$):LOCAL I
 2520 IF p$="" THEN PRINT"No file given"
:ENDPROC
 2540 IF open% THEN PRINT"File already o
pen":ENDPROC
 2560 PROCwindow1:CLS
 2580 PRINTTAB(5,0)"Creating file ";p$
 2600 PRINTTAB(5,2)"Number of records:":
recn=VAL(FNinput(24,2,4,"."))
 2620 PRINTTAB(45,2)"Number of fields:"
 2640 REPEAT:f=VAL(FNinput(63,2,2,".")):
UNTIL f<=maxf
 2660 FOR I=1 TO f
 2680 PRINTTAB(5,3+I)"Field";STR$(I)
 2700 PRINTTAB(16,3+I)"Name";
 2720 field$(I)=FNinput(21,3+I,12,".")
 2740 PRINTTAB(45,3+I)"Fieldwidth"
 2760 REPEAT:width%(I)=VAL(FNinput(56,3+
I,2,".")):UNTIL width%(I)<=64
 2780 NEXT I:PROCwindow2
 2800 IF FNask("Confirm (Y/N): ")>2 THEN
 ENDPROC
 2820 recs=2*f:FOR I=1 TO f:recs=recs+wi
dth%(I):NEXT I
 2840 PROCoscli("SAVE "+p$+" 0+"+STR$~(2
56+recn*recs))
 2860 rec=1:F=OPENUP(p$):PTR#F=0:PRINT#F
,rec,recn,recs,f
 2880 FOR I=1 TO f:PRINT#F,field$(I),wid
th%(I):NEXT I:CLOSE#F
```

```
 2900 PROCwindow1:CLS:PROCwindow2
 2920 PRINT"File ";p$;" created"
 2940 ENDPROC
 2960 :
 3200 DEF PROCopen(p$):LOCAL I
 3220 IF p$="" THEN PRINT"No file given"
:ENDPROC
 3240 IF open% THEN PROCclose
 3260 F$=p$:F=OPENUP(F$)
 3280 IF F=0 THEN PRINT"No such file":EN
DPROC ELSE open%=TRUE
 3300 PTR#F=0:INPUT#F,rec,recn,recs,f
 3320 FOR I=1 TO f:INPUT#F,field$(I),wid
th%(I):NEXT I
 3330 IF f>8 THEN L=1 ELSE L=2
 3340 PROCheader:ENDPROC
 3360 :
 3600 DEF PROCadd(p$):LOCAL I
 3620 IF NOT open% THEN PRINT"No file op
en":ENDPROC
 3640 PROCwindow1:PROCrecord
 3660 PROCinv(1):PRINTTAB(64,0)"Record:
";SPC(4-LEN(STR$(rec)));rec:PROCinv(0)
 3680 FOR I=1 TO f:record$(I)=FNinput(13
,L*(I-1)+1,width%(I),"."):NEXT I
 3700 PROCwindow2:IF FNask("Confirm (Y/N
): ")<3 THEN PTR#F=256+recs*(rec-1):FOR
I=1 TO f:PRINT#F,record$(I):NEXT I:rec=r
ec+1:PTR#F=0:PRINT#F,rec
 3720 PROCwindow1:PROCinv(1):PRINTTAB(47
,0)SPC(4-LEN(STR$(rec-1)));rec-1:PROCinv
(0):PROCwindow2
 3740 ENDPROC
 3760 :
 4000 DEF FNinput(x,y,w,p$)
 4020 LOCAL ch$,c,p:ch$="":p=1
 4040 PRINTTAB(x,y)STRING$(w,p$):VDU31,x
,y
 4060 REPEAT:c=GET
 4080 IF c=127 THEN PROCd ELSE PROCa
 4100 UNTIL c=13
 4120 ch$=ch$+STRING$(w-LEN(ch$),p$)
 4140 =ch$
 4160 :
 4180 DEF PROCd
 4200 IF p>1 THEN VDU8,ASCp$,8:ch$=LEFT$
(ch$,LEN(ch$)-1):p=p-1
 4220 ENDPROC
 4240 :
 4260 DEF PROCa
 4280 IF p<=w AND c<>13 THEN ch$=ch$+CHR
$(c):VDU c:p=p+1
 4300 ENDPROC
 4320 :
 4500 DEF PROCrecord:LOCAL I
 4520 FOR I=1 TO f
 4540 PRINTTAB(0,L*(I-1)+1)field$(I)TAB(
13)STRING$(width%(I),".")
 4560 NEXT I:ENDPROC
 4580 :
 4800 DEF PROCdisplay(p$)
 4820 LOCAL G,I,start,end,n:n=VAL(p$)
 4840 IF NOT open% THEN PRINT"No file op
en":ENDPROC
 4860 IF rec<2 THEN PRINT"No records in
file":ENDPROC
 4880 IF n<0 OR n>rec-1 THEN PRINT"No su
ch record":ENDPROC
 4900 PROCwindow1:PROCrecord
 4920 IF n THEN start=n:end=n ELSE start
=1:end=rec-1
 4940 FOR I=start TO end
 4960 PROCdisplay1(I)
 4980 IF I<end THEN G=GET
 5000 NEXT I:PROCwindow2
 5020 ENDPROC
 5040 :
 5200 DEF PROCdisplay1(n)
 5220 LOCAL I
 5240 PROCinv(1):PRINTTAB(64,0)"Record:
";SPC(4-LEN(STR$(n)));n:PROCinv(0)
 5260 PTR#F=256+recs*(n-1)
 5280 FOR I=1 TO f:INPUT#F,record$(I):PR
INTTAB(13,L*(I-1)+1)record$(I):NEXT I
 5300 ENDPROC
 5320 :
 5500 DEF PROCdelete(p$)
 5520 LOCAL A,I,n:n=VAL(p$)
 5540 IF NOT open% THEN PRINT"No file op
en":ENDPROC
 5560 IF rec<2 THEN PRINT"No records in
file":ENDPROC
 5580 IF n<1 OR n>rec-1 THEN PRINT"No su
ch record":ENDPROC
 5600 PROCwindow1:PROCrecord
 5620 PROCdisplay1(n):PROCwindow2
 5640 A=FNask("Delete (Y/N): ")
 5660 IF A<3 THEN PTR#F=256+recs*(n-1):F
OR I=1 TO f:PRINT#F,STRING$(width%(I),".
"):NEXT I
 5680 ENDPROC
 5700 :
 6000 DEF PROCclose:LOCAL I,J,K
 6020 IF NOT open% THEN PRINT"No file op
en":ENDPROC
 6030 IF rec=1 THEN 6160
 6040 PRINT"Please wait - closing file "
;F$
 6060 J=1:FOR I=1 TO rec-1
 6080 PTR#F=256+recs*(I-1)
 6100 FOR K=1 TO f:INPUT#F,record$(K):NE
XT K
 6120 IF ASC(record$(1))<>46 THEN J=J+1:
PTR#F=256+recs*(J-2):FOR K=1 TO f:PRINT#
F,record$(K):NEXT K
 6140 NEXT I:rec=J:PTR#F=0:PRINT#F,rec
 6160 PRINT"File closed - ";rec-1;" reco
rds in use"
 6180 CLOSE#F:open%=FALSE
 6200 ENDPROC
```

→ 36

COMPRESSING SCREEN DISPLAYS

Screen displays use up much valuable storage space and, for cassette users, can be very slow to save and load. Geoff Bains and Alan Webster show how compressing your pictures can greatly improve matters.

A major problem with the BBC micro is that its graphics displays take up an awful lot of memory. Cassette users know well the hours needed for a 20K screen to load from tape. Disc users also suffer - you can only fit four such screens onto a 40 track disc. This is where this pair of programs will prove useful. The first program will compress a 20K screen to around 6K, the exact figure depending on the picture itself. The second program consists of a procedure to add to your own program that loads in the compressed screen and expands it back to full size.

COMPRESSION

The compression technique relies on the fact that most pictures have large areas of the same colour. If an area of the screen is composed of the same bytes of data we can more efficiently store it as the byte and the number of repetitions of these bytes. This program extends this idea to account for stippled or striped areas and stores the screen in the form:

Byte1 Byte2 Number of byte pairs

The compressed data is stored initially in RAM. This is then saved with a reload address calculated so that the data loads into RAM at the bottom of the screen. This avoids the compressed data occupying any memory additional to that required by the expanded picture (20K).

Using the program is simple enough. Just answer the prompts for the name of the uncompressed screen and the compressed data and the program does all the rest.

```
  10 REM PROGRAM SCREEN COMPRESSION
  20 REM VERSION B0.1
  30 REM AUTHOR  GEOFF BAINS
  40 REM BEEBUG  NOV 1985
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 MODE 1
 110 INPUT "Filename of screen"'"to co
mpress ",N$
 120 INPUT '"Filename for compressed"'
'"screen ",C$
 130 VDU28,1,25,19,20
 140 *OPT 1,0
 150 A$="LOAD "+N$+" 3000":PROCos
 160 store%=&2000
 170 HIMEM=store%
 180 B1%=?&3000:B2%=?&3001:N%=1
 190 FOR I%=&3002 TO &8000 STEP2
 200 D1%=?(I%):D2%=?(I%+1)
 210 ?(I%)=NOT D1%:?(I%+1)=NOT D2%
 220 IF D1%<>B1% OR D2%<>B2% OR N%=255
THEN ?(store%)=B1%:?(store%+1)=B2%:?(sto
re%+2)=N%:N%=0:B1%=D1%:B2%=D2%:store%=st
ore%+3
 230 N%=N%+1
 240 length%=store%-HIMEM
 250 NEXT I%
 260 *OPT
 270 A$="SAVE "+C$+" "+STR$~HIMEM+" +"+
STR$~(length%)+" 0000 "+STR$~(&8000-leng
th%):PROCos
 280 END
 290 DEFPROCos:$&900=A$:X%=0:Y%=9
 300 CALL&FFF7:ENDPROC
```

DECOMPRESSION

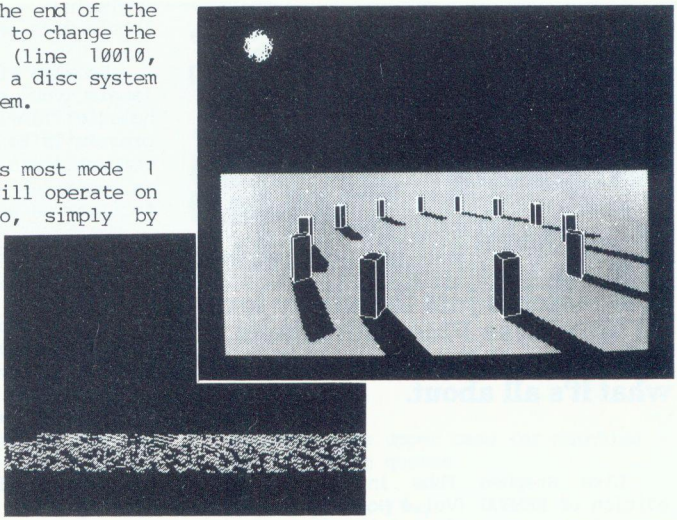
Expanding the compressed data back into its full picture form is a little more complicated. The picture is loaded into position at the end of screen memory by the machine code equivalent of *LOAD. The start address of the data is found (as a result of the load operation) and the decoding proceeds as a simple loop, poking the expanded data directly into screen memory.

To use the expansion routine in your own programs, simply add the procedure, PROCscreenload onto the end of your program and call it with the filename of the compressed data (the name you chose in the first program to store the data) as the parameter (see the example in lines lines 100-120). For example: you could use this procedure in the Jigsaw program (Vol.4 No.3) to save on time/space to store the puzzle picture. Compress your picture and replace line 1580 of Jigsaw with PROCscreenload(F$) and append the

procedure listed here onto the end of the program. You will also have to change the address used for the code (line 10010, below) to either &0A00 for a disc system or &0D00 for a cassette system.

DEGREE OF COMPRESSION

The program will compress most mode 1 screens to around 6K. It will operate on mode 0 and mode 2 screens too, simply by altering the mode in line 100 of each of the routines, though as it is designed for mode 1 screens, the degree of compression will not be as great in these cases. 10K modes are not accomodated (though they could be) as the savings are less.



If the compressed data is too large (from a very detailed picture, for example) it will be overwritten by the reconstituted picture as it is decoded. This is not normally a problem as such large data is not really worth using in a compressed form. If a screen is very detailed, with nearly every pixel pair different from its neighbour, the compressed data can get to be bigger than the original screen!

```
   10 REM PROGRAM SCREEN DECOMPRESSION
   20 REM VERSION B0.1
   30 REM AUTHOR  ALAN WEBSTER
   40 REM BEEBUG  NOV 1985
   50 REM PROGRAM SUBJECT TO COPYRIGHT
   60 :
  100 MODE 1
  110 PROCscreenload("filename")
  120 END
  130 :
10000 DEF PROCscreenload(N$)
10010 name%=&0900
10020 FOR pass=0TO2STEP2:P%=name%+12
10030 [OPTpass
10040 .load
10050 LDA#&FF:LDY#6:STA pblock,Y
10060 LDX #pblock MOD 256
10070 LDY #pblock DIV 256
10080 LDA #255:JSR &FFDD
10090 JSR subt2:RTS
10100 .pblock
10110 BRK:BRK
10120 JMP0:JMP0:JMP0:JMP0:JMP0:BRK
10130 .subt2:LDA &3BE:STA&80
10140 LDA &3BF:STA&81:SEC:LDA &80
10150 SBC#3:STA &80:LDA &81:SBC #0:STA &81
10160 :
10170 .decode
10180 LDA #0:STA&84:LDA#&30:STA&85
10190 .loop
10200 JSR inc2
10210 CLC:LDA &81:CMP #&80:BEQend
10220 LDY#0
10230 LDA(&80),Y:STA &8E
10240 INY:LDA(&80),Y:STA &8F
10250 INY:LDA(&80),Y:TAX
10260 .loop2
10270 LDY#0
10280 LDA&8E:STA(&84),Y
10290 INY:LDA&8F:STA(&84),Y
10300 JSR inc
10310 DEX:CPX#0:BNE loop2
10320 JMP loop
10330 .end
10340 RTS
10350 .inc
10360 CLC:LDA&84:ADC#2:STA&84
10370 LDA&85:ADC#0:STA&85:RTS
10380 .inc2
10390 CLC:LDA&80:ADC#3:STA&80
10400 LDA&81:ADC#0:STA&81
10410 RTS
10420 ]:NEXT pass
10430 ?pblock=name%:pblock?1=name%DIV256
10440 VDU 12,28,1,10,19,2
10450 $name%=N$
10460 CALL load
10470 VDU26
10480 ENDPROC
```

Wordwise Plus Program Compactor

This segment program compactor is just the utility for all Wordwise Plus enthusiasts. Leila Burrell-Davies explains what it's all about.

Like Stephen Ibbs in the June 1985 edition of BEEBUG (Vol.4 No.2), I am an enthusiastic user of Wordwise Plus segment programs. Like him I also have problems with fitting into memory all the segments that I want and having any memory left for text.

The solution I have adopted is to write my segment programs with plenty of spaces and REMs and explanatory label names and then, when they are thoroughly tested, to compress them into working versions which are as compact as possible. I keep the original (full) version in file F.segname and the working version in file +.segname. When I want to modify the program, I go back to the full version, which is much more readable and easy to understand.

Needless to say, the compacting is done with a segment program! It compacts the program in the currently selected area of memory by replacing all the keywords with their minimum abbreviations, deleting all spaces which are not in quotes, removing all REMs (except on the first line), removing all blank lines and replacing possibly long explanatory labels with short numeric ones.

Because of this method of dealing with labels you should avoid using any purely numeric labels if your Wordwise Plus program is to be compacted. The BEEBUGSOFT compactor in Wordease avoids this problem as well as offering a decompaction facility as well.

The keywords and their abbreviations are stored in a file W.ABBREVS in the format:

KEYWORD→abbrev.

where the symbol '→' means 'press the Tab key'. The keywords are ordered by length, longest first. This is to ensure, for instance, that REPEAT is replaced by r. before the AT which it contains is replaced by 'a'. You will need to enter this as a Wordwise Plus segment and save as a separate file.

The hash character (#), has to be replaced temporarily by a character which does not occur in the text; @ is used here. This is because # acts as a wild card in search strings, so that a search for CLOSE# will find CLOSE followed by any character. The compactor checks that the '@' character does not appear in the text to be compacted. If it does you are requested to replace it temporarily while compaction takes place.

The program has to do quite a lot of work and is, unfortunately, rather slow. Because of this, it does prints dots on the screen while it is running to show the user that something is happening.

The program will work with a cassette system so long as the cassette player has motor control. It should be entered as a Wordwise Plus segment program and saved as F.WW+COMP (or similar). You can then load and call this program to compact any other program you write. As a start, you can use it to compact itself!

DEFAULTS→def.	VARFREE→v.	PARAMS→par.	PRINT→p.
DISPLAY→di.	BOTTOM→b.	REPEAT→r.	RIGHT→r.
ENDPROC→e.	CLOSE@→close@	SELECT→s.	SPOOL→sp.
MARKERS→marker.	CURSOR→c.	BGET@→b.	TIMES→t.
OPENOUT→openo.	DELETE→de.	BPUT@→bp.	UNTIL→u.
PREVIEW→pre.	DOLINE→dol.	FALSE→fa.	WORDS→w.
RECOUNT→rec.	DOTHIS→d.	GCF$@→gcf.	CALL→ca.
REPLACE→repl.	MARKED→m.	GLF$@→glf.	CHR$→chr.
SEGMENT→s.	OPENIN→op.	OSCLI→o.	DOWN→d.

EOF@>eof.	LOAD>lo.	WORD>w.	MOD>m.
EXT@>ex.	PAGE>p.	AND>a.	REM>rem
FILE>f.	PROC>pro.	ASC>as.	SOT>so.
FIND>f.	PTR@>pt.	CLS>cl.	TOP>t.
FKEY>fk.	SAVE>sa.	DIV>div	TTC>tt.
FREE>fr.	STR$>st.	END>end	VAL>val
GCK$>gck.	SWAP>sw.	EOR>e.	VDU>v.
GCT$>gct.	TEXT>te.	EOT>eot	AT>a.
GLK$>glk.	THEN>t.	GET>ge.	IF>i.
GLT$>glt.	TIME>time	LEN>len	OR>o.
GOTO>g.	TRUE>tr.	LET>l.	UP>u.
LEFT>l.	TYPE>ty.		

```
REM "Wordwise Plus program compactor"
REM Leila Burrell-Davis, November 1985

REM W$ can be changed from @ if
REM required so long as it is also
REM altered in file V$.

CLOSE#0
DELETE MARKERS

REM name of abbreviations file
V$="W.ABBREVS"

REM char which mustn't appear in text
W$="@"
CURSOR TOP
FKEY4,W$
IF EOT THEN GOTO W$ok
VDU7
CLS
PRINT"Special character "+W$+" appears"
PRINT"in your text. Please change it"
PRINT"into something else temporarily."
PRINT
PRINT"Press any key to continue...";
A%=GET
DISPLAY
END

.W$ok
REM open file V$
CLS
PRINT"Trying to open "+V$+" file..."
X%=OPENIN(V$)
IF X%=0 THEN GOTO nofile
CLS
PRINT"Working ";

REM now change #'s to W$'s because
REM # is wild in replace strings
PRINT".";
CURSOR TOP
REPEAT
  FKEY4,"#"
  IF EOT THEN GOTO atend
  DELETE at
  type W$
 .atend
until eot

REM make file upper case for matching
REM except in quotes
cursor top
REM in quotes flag
Q%=false
REPEAT
  A$=gct$
  REM if quote, toggle quote flag
  IF A$=chr$34 THEN Q%=Q% EOR &FFFF
  REM if eol, set quote flag false in
  REM case unbalanced quotes in line
  IF A$=chr$13 THEN Q%=false
  IF Q% THEN GOTO next
  IF A$<"a" OR A$>"z"THEN GOTO next
  CURSOR LEFT
  SWAP
  PRINT".";
 .next
UNTIL EOT

REM now replace keywords by abbrevs.
REPEAT
PRINT".";
  A$=""
  B$=""
  REM Get keyword
  REPEAT
    A$=A$+B$
    B$=gcf$#X%
    REM lenB$ will be 0 when TAB read
    REM as it has ASCII value > 127
  UNTIL LENB$=0
  REM Get abbreviation
  B$=glf$#X%
  CURSOR TOP
  REPEAT
    REPLACE A$,B$
  UNTIL EOT
UNTIL EOF#X%
CLOSE#X%
```

```
REM get rid of spaces except in quotes
CURSOR TOP
REM in quotes flag
Q%=FALSE
REPEAT
  A$=GCT$
  REM if quote, toggle quote flag
  IF A$=chr$34 THEN Q%=Q% EOR &FFFF
  REM if eol, set quote flag false in
  REM case unbalanced quotes in line
  IF A$=chr$13 THEN Q%=FALSE
  IF A$<>" " THEN GOTO nextchar
  IF Q%=FALSE THEN DELETE LEFT
  PRINT".";
 .nextchar
UNTIL EOT

REM remove rems unless on first line
REM (first line rem will have been
REM tokenised unless in quotes and
REM lowercase)
PRINT".";
CURSOR TOP
REPEAT
  FIND CHR$13+"rem"
  FKEY6,CHR$13
  DELETE AT
UNTIL EOT

REM remove blank lines
PRINT".";
CURSOR TOP
REPEAT
  REPLACE "|R|R","|R"
UNTIL EOT

REM now deal with labels
A%=0
CURSOR TOP
REPEAT
PRINT".";
  FIND "|R."
  IF EOT THEN GOTO nextlabel
  CURSOR RIGHT
  A$=GLT$
  CURSOR TOP
  REPEAT
    REPLACE A$,"."+STR$A%
    IF EOT=FALSE THEN CURSOR RIGHT
  UNTIL EOT
  CURSOR TOP
  FIND "|R"+"."+STR$A%
  CURSOR RIGHT
  A%=A%+1
 .nextlabel
UNTIL EOT

REM now change W$'s back to #'s
PRINT".";
CURSOR TOP
REPEAT
  REPLACE W$,"#"
UNTIL EOT

CURSOR TOP
DISPLAY
END

.nofile
REM abbrevs. file not on current drive
VDU 7
VDU 14
PRINT
PRINT"File "+V$+" not found."
PRINT
*CAT
PRINT
PRINT"Press any key ..."
A%=GET
DISPLAY
END
```

Sample of compacted file

```
p."Trying to open "+V$+" file..."
X%=op.(V$)
i.X%=0t.g.5
cl.
p."Working ";
p.".";
c.t.
r.
fk.4,"#"
i.eott.g.1
de.a.
ty.W$
.1
u.eott
c.t.
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS

FILLING UP - Roger Burg

Using a graphics window and the CLG command is a well known method of filling rectangles on the screen. However this method will only fill from the top. If you want to fill from the bottom of the rectangle, for a different effect, define a graphics window as before then use PLOT85 to draw a very large triangle that encompasses the whole window.

Writing your own compiler (part 1)

David Pilling begins a new series in which he describes the steps involved in constructing a compiler.

In this short series of articles, I will describe how to design and build your own compiler. This will provide a practical introduction to those who would like to try their hand at compiler writing. In addition, an insight into the art of compiler construction can help you to write better programs by understanding how the computer works. The three articles will, stage by stage, build up to a complete working compiler for a useful subset of Basic.

WHAT IS A COMPILER?

Everyone who has used a micro, will be familiar with an interpreter. This is a program which takes lines of Basic and executes them one at a time. A compiler on the other hand takes an entire program written in a high level language (the source code) and translates it into a program in another language (the object code). The classic example is the conversion of a Basic program to machine code. The usual reason for wanting to translate from one language to another is to obtain a program which is in a language closer to that of the processor in the computer and which will therefore run faster, often using less memory as well.

Compilers are complicated programs and there is a certain amount of mythology attached to them, not least that they are very difficult to write. Certainly this was true thirty years ago. In the time since then, much effort has been put into finding out how to write compilers, and as a result, if you stick to certain principles, it is fairly simple.

Compilers are often assumed to give a huge increase in run-time speed over using an interpreter. This is not universally true. For instance, it takes a finite amount of time for a 6502 to multiply two numbers together. No compiler can improve on this. What the compiler can do, is remove the time the interpreter takes to find the locations of the numbers in memory and the destination of the result. In addition, compiled programs will often run faster just because the code is translated once and not repeatedly re-interpreted.

Although many books have been written on the theory of compiler writing, the object here is to be very practical. As a result, the implementation of a Basic compiler (ABC) for the subset of BBC Basic shown in figure one will be described, with the first part of the program this month. The only point of interest about this subset, is that the WHILE WEND construct and PEEK and POKE have been included to show how easy it is to add features to a language.

FIRST PRINCIPLES

To begin then, the fundamental idea of a compiler is to translate from one language to another. The process is sketched out in figure two. The first parameter which allows the comparison of

IF THEN ELSE	PRINT	SIN	INKEY
WHILE WEND	INPUT	COS	END
REPEAT UNTIL	TAB(X,Y)	SQR	PEEK
FOR TO STEP NEXT	VDU	RND(X)	POKE
DEFPROC PROC ENDPROC	MOVE	PI	+,-,*,/
GOSUB RETURN	DRAW	TRUE	<,>,>=,<=,<>
GOTO	SOUND	FALSE	()

All variables are reals. No variable is allowed after NEXT. The delimiters DEFPROC ENDPROC, FOR NEXT, REPEAT UNTIL, WHILE WEND must appear in pairs. Procedures may have parameters and be called recursively.

Figure One: Basic Subset



Figure Two: Compilation

different compilers, is how many passes have to be made through the program. One speaks of a one pass, two pass etc. compiler. Obviously, there are practicalities involved here. If the source program is in a file on tape then a compiler that makes as few passes as possible is needed. However, the structure of the language to

be compiled determines how easy it is to compile programs in a single pass.

It is possible to compile a language which has been designed appropriately, by scanning programs from left to right without ever going backwards. For some languages such 'backtracking' is essential and leads to great complications. By keeping to the single left right scan, things are made much easier. It is no surprise that BBC Basic can be translated in just this way.

Given that one is setting out to compile Basic or another high level language, it may seem a trivial question as to what it is to be translated into. Most people would say machine code. The problem is that a processor like the 6502 lacks instructions to do almost everything that one would like. There are no instructions for handling floating point variables, for example. This means that there must be a large group of subroutines to provide all the facilities compiled programs need. This is referred to as a run-time support system. If programs are compiled directly to machine code, then for the most part the object code will consist of calls to these subroutines. Such a machine code compiler may suffer as although the programs it produces are fast they tend often to be very large, because of the run-time support system.

The compiler described here, produces 6502 machine code and mostly uses the mathematical routines in the Basic ROM for its run time system.

COMPILER STRUCTURE

The actual compilation process can be divided into a number of phases. The classic design of compiler has three: lexical analysis, syntax analysis and code generation. Figure three shows this structure. Sometimes, these three processes are carried out by completely separate programs and the output of the syntax and lexical analysis parts stored in an intermediate form. Our compiler will cope with all three together.

Lexical analysis takes the source code, and splits it up into the smallest significant items commonly called tokens or, appropriately, atoms. The idea is to protect the syntax analysis part of the compiler from the unnecessary complexity of how programs are stored. To give an example: the syntax analyser is only interested in whether the next item from the source program is a number, not in constructing it from a string of digits.

Syntax analysis derives the meaning of the program to be compiled and finally, acting on the results of the syntax analysis, the code generation phase produces the object code. The remaining item in figure three is the 'symbol table'. This is a table which is accessed by all stages of the compiler. It contains all the objects that the compiler knows about, for example names of variables and their memory locations.

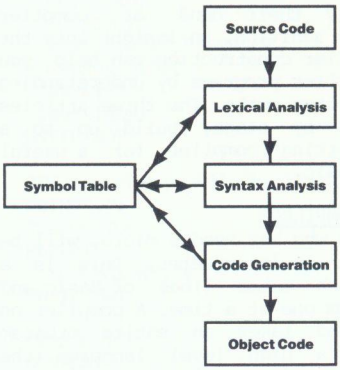


Figure Three: Compiler Structure

LEXICAL ANALYSER

Logically therefore, the first piece of the compiler should consist of the lexical analyser and symbol tables. What is needed is a way of holding a piece of source code and then a procedure which, each time it is called, returns the next token from it. In addition, every time the procedure finds a new object, it should enter it in the symbol table.

The tokens which can be found in the source text, comprise keywords, numerical constants (numbers), string constants, operators, spaces and identifiers. An identifier is like the name of a variable or procedure (an alphanumeric sequence started by a letter). Such identifiers are entered in the symbol table. The lexical

analyser, should strip off spaces and assemble composite operators like <> into single tokens. This section of the compiler is listed this month and appears on the magazine cassette/disc. The other parts of the compiler will appear similarly in the following two issues.

PROGRAM NOTES FOR THE LEXICAL ANALYSER

The program to be compiled is entered between lines 10 and 1000 and terminated with a STOP statement. The source code must not contain procedures with the same names as those in the compiler. Lines 1000 to 1300 form the main compiler program. PROCB1 and PROCB2 set up the locations of the routines in ROM which the compiler uses depending on whether you have Basic I or II. Only the procedure for your machine need be typed in. PROCSETUP is, naturally, concerned with setting up the values of various variables and dimensioning arrays.

The lexical analyser is PROCLX. It has the following specification. Every time PROCLX is called, it sets the variable T% to the value of the next token. If the token is a number, the variable N is given its value. If the token is a string or an identifier then S$ is set to it. Finally, if a line number is found, N% is given its value (a line number is what appears after a GOTO or GOSUB statement). In addition, SN% is set to the number of any object in the symbol table.

As well as the straightforward left right scan of the program at a token level, we also impose the condition that the lexical analyser should scan the characters that compose the source code program in a single left right scan, and that the variable A% always contains the next character to be read. Thus the lexical analyser has a single token single character look-ahead.

Anyone familiar with the way that BBC Basic programs are stored, should have little difficulty in fulfilling the above specification. Consider the version of PROCLX in the listing. This always reads characters by calling FNCH. The task of implementing a lexical analyser, is made rather simpler by the fact that all Basic keywords are represented in programs by tokens (a subset of the tokens above) in the range 128 to 255. So part of the lexical analysis job has been done. This means that keywords can give T% the value of their usual token. For other things, token values can be invented. This is done in lines 1730 and 1740. For example the combinations <=, >= and <> have tokens 1,2, and 3 and the new keywords WHILE and WEND the tokens 8 and 9.

On entering PROCLX, the first thing that is done is to look at the current value of T%. If this is equal to an end of line character (line 1950), PROCNL is called. This procedure sorts out the problem of getting the next line from the source text. If T% is equal to the end-of-program token (i.e. STOP), PROCLX does nothing and ends. Next, A% is tested and characters are read until something which is not a space is met.

At line 1970, FNAL is used to test if A% is an alphabetical character. If it is, then it represents the beginning of an identifier. PROCIG is used to assemble it into S$. Having done this, the program looks in line 1990 to see if the identifier is one of the new keywords. If it is, T% is given the appropriate value. Otherwise, the identifier is a variable name and PROCS is called. Obviously one could put a big table of new keywords at this point. Picking out the new keywords here does mean that they must be terminated with a space if they are not to be confused with variables.

PROCS handles the symbol table. The symbol table consists of the arrays st(), sa() and stp(). These hold the name of the object, its address and its type. For variables, the address is the actual location where the variable is stored. For procedures, it would be the address of the start of the procedure. It is necessary to keep track of the type of an object so that, for instance, procedures can be distinguished from variables of the same name. If a variable is entered into the symbol table, then an address in memory is allocated to it in line 2460.

If A% is not a letter, PROCLX branches to line 2010 where, if A% is the PROC token, the procedure name is assembled and entered in the symbol table. Otherwise, control branches to line 2040 where four more types of token are sorted out. If A% is a number, PROCDNG takes care of assigning it to N. A double quote value of A% calls PROCSG into action to get a constant string. If A% is the BBC Basic

line number token, PROCLG retrieves the line number. Finally, if A% is in none of the above categories, PROCTG is called. This checks to see if A% is part of one of the <, > special sequences and also strips off any unnecessary colons. T% is given the appropriate value.

There are a few complications to note concerning the handling of line numbers. When a line number is found after a GOTO, it is put as a string into the symbol table. This is logical, since in the object code, a line number corresponds to an address. Secondly, there is another table, consisting of the arrays lino() and lina(). These hold line numbers and corresponding object code addresses. Every time PROCNL is called, it enters the new line number in lino() and the value of the location at which code is being written in lina().

As the source code is scanned, it is displayed on the screen. This is done by making use of PROCA which prints the character value of A% and then gives A% the value of the next character using FNCH. A call to a ROM routine makes sure that keyword tokens are printed properly.

The processes described above are more general than they may seem. For example, all conventional languages have very similar specifications for identifiers. Therefore, all compilers for these languages contain a section like that in the listing for handling identifiers. The same point applies to numerical constants and so on. The lexical analyser, is perhaps the simplest part of any compiler, and as such is not greatly interesting. However, it provides one of the major bottlenecks as regards the speed at which programs are compiled and efficient design is essential for a fast compiler.

The discipline of always reading characters via FNCH confers the great advantage that it is also possible to compile programs from disc or tape thus making more memory available for the object code. To do this the following lines should be added:

```
1151 INPUT"FILE TO COMPILE "S$
1152 CH=OPENINS$
1171 CLOSE#CH
1795 Z%=0
1910 DEFFNCH:Z%=Z%+1:=BGET#CH
```

Disc users will probably find this modification very useful; tape users will need patience to use it. Programs must still be terminated with a STOP statement.

In part two, I will turn to the subject of syntax analysis and present the next section of the compiler.

```
  10 GOTO1000
 1000 REM ============================
 1010 REM A BASIC COMPILER
 1020 REM VERSION B1.1
 1030 REM D. J. Pilling
 1040 REM BEEBUG November 1985
 1050 REM Program subject to copyright
 1060 REM ============================
 1070 REM:
 1080 REM Section #1
 1090 REM Lexical Analysis.
 1100 MODE7
 1110 ns=18:nl=25
 1120 himem=&5800
 1130 msize=3100
 1140 page=himem-msize:HIMEM=page
 1150 PROCSETUP:CLS
 1160 T%=FNCH:A%=FNCH:PROCLX
 1170 PROCprog:top=P%
 1180 PROCFIX
 1190 PROCS1
 1200 PRINT'''"page = ";page
 1210 PRINT"himem = ";himem
 1220 PRINT"msize = ";msize
 1230 PRINT"variables ";himem-vmem-5
 1240 PRINT"object code ";top-GO
 1250 PRINT"source code ";Z%-W%
 1260 PRINT'''"execute code (Y/N)? ";
 1270 G%=GET:MODE 7
 1280 IF G%=ASC"Y" THEN CALL GO
 1290 END
 1300 :
 1310 DEFFNL(B1%,B2%,B3%)
 1320 B3%=B3%*256+B2%-&4040
 1330 IFB1%MOD&10=0 B3%=B3%+16384:B1%=B1
%+4
 1340 IFB1%=&44 B1%=64 ELSEIFB1%=&54 B1%
=0 ELSEIFB1%=&64 B1%=192 ELSEIFB1%=&74 B
1%=128
 1350 =B1%+B3%
 1360 :
 1370 DEFPROCB1
 1380 TK=&B53A:sqx=&A7B7:f2i=&A3F2
 1390 A2f=&A2DE:f2s=&9ED0:neg=&ADA0
 1400 ldz=&A691:sub=&A50B:mul=&A661
 1410 div=&A6B8:add=&A50E:ldf=&A3A6
 1420 stf=&A37E:cpf=&9A37:sig=&A1CB
 1430 six=&A997:cox=&A98C:pix=&ABF0
 1440 i2f=&A2AF:rnx=&AF53
 1450 asc2=&AC5A:ins=&BC17
```

```
1460 ENDPROC
1470 DEFPROCB2
1480 TK=&B50E:sqx=&A7B7:f2i=&A3E4
1490 A2f=&A2ED:f2s=&9EDF:neg=&AD7E
1500 ldz=&A686:sub=&A4FD:mul=&A656
1510 div=&A6AD:add=&A500:ldf=&A3B5
1520 stf=&A38D:cpf=&9A5F:sig=&A1DA
1530 six=&A99B:cox=&A990:pix=&ABCB
1540 i2f=&A2BE:rnx=&AF24
1550 asc2=&AC34:ins=&BBFC
1560 ENDPROC
1565 :
1570 DEFPROCSETUP
1580 IF?&8015=50 PROCB2 ELSEPROCB1
1590 PROCruntime
1600 DIM st$(ns),sa(ns),stp(ns)
1610 DIM lino(nl),lina(nl)
1620 S$=STRING$(40," "):S$="":NL%=0
1630 for=&E3:to=&B8:step=&88:next=&ED
1640 vdu=&EF:print=&F1:proc=&F2:def=&DD
1650 endproc=&E1:rep=&F5:unt=&FD
1660 goto=&E5:end=&E0:inkey=&A6:if=&E7
1670 else=&8B:lino=&8D:then=&8C:peek=&A
1680 eq=&3D:gt=&3E:lt=&3C:cls=&DB
1690 lb=&28:rb=&29:tr=&B9:fl=&A3:poke=& B
1700 plus=&2B:minus=&2D:slash=&2F
1710 times=&2A:squote=&27:tab=&8A
1720 dquote=&22:comma=&2C:scol=&3B
1730 col=&3A:spc=32:eoln=&D:eop=&FA
1740 geq=1:leq=2:neq=3:id=4:lbl=5
1750 const=6:string=7:whi=8:wnd=9
1760 sound=&D4:gosub=&E4:return=&F8
1770 inp=&E8:sin=&B5:cos=&9B:rnd=&B3
1780 draw=&DF:move=&EC:sqr=&B6:pi=&AF
1790 Z%=PAGE+?(PAGE+3):SN%=0:jm=0
1800 X1%=0:X2%=0:Y1%=0:Y2%=0:W%=Z%
1810 vmem=himem-5
1820 ENDPROC
1830 DEFPROCS1
1840 X2%=POS:Y2%=VPOS
1850 VDU28,0,10,39,0,31,X1%,Y1%
1860 ENDPROC
1870 DEFPROCS2
1880 X1%=POS:Y1%=VPOS
1890 VDU28,0,24,39,12,31,X2%,Y2%
1900 ENDPROC
1910 DEFFNCH:Z%=Z%+1:=Z%?-1
1920 DEFPROCA:PROCS1:CALL TK:PROCS2:A%=
FNCH:ENDPROC
1930 DEFPROCLX
1940 IFT%=eop:ENDPROC
1950 IFT%=eoln PROCNL
1960 IFA%=spc:REPEAT PROCA:UNTILA%<>spc
1970 IFNOTFNAL(A%) GOTO2010
1980 PROCIG
1990 IFS$="WHILE"T%=whi ELSEIFS$="WEND"
T%=wnd ELSEIFS$="PEEK"T%=peek ELSEIFS$="
POKE"T%=poke ELSET%=id:PROCS
2000 ENDPROC
2010 IFA%<>proc GOTO2040
2020 T%=A%:PROCA:IFNOTFNAL(A%) T%=0:END
PROC
2030 PROCIG:PROCS:ENDPROC
2040 IF FNDN(A%) PROCDNG ELSEIF A%=dquo
te PROCSG ELSEIF A%=lino PROCLG ELSEPROC
TG
2050 ENDPROC
2060 DEFFNAL(A%):=(A%>64ANDA%<91)OR(A%>
96ANDA%<123)
2070 DEFFNDN(A%):=(A%>47ANDA%<58)
2080 DEFPROCLG
2090 N%=FNL(FNCH,FNCH,FNCH)
2100 T%=lino:A%=FNCH:S$=STR$N%
2110 PROCS:PROCS1:PRINT;N%;:PROCS2
2120 ENDPROC
2130 DEFPROCIG
2140 S$="":REPEAT:S$=S$+CHR$A%:PROCA:UN
TIL NOT(FNDN(A%)ORFNAL(A%))
2150 ENDPROC
2160 DEFPROCDNG
2170 S$="":PROCDNGX
2180 IFA%=ASC"."S$=S$+".":PROCA:IFFNDN(
A%)PROCDNGX
2190 IFA%<>ASC"E"GOTO2220
2200 S$=S$+"E":PROCA:IFA%=ASC"-"PROCA:S
$=S$+"-"
2210 IFFNDN(A%)PROCDNGX
2220 T%=const:N=VALS$
2230 ENDPROC
2240 DEFPROCDNGX:REPEAT:S$=S$+CHR$A%:PR
OCA:UNTIL NOTFNDN(A%):ENDPROC
2250 DEFPROCSG:S$="":PROCA
2260 REPEAT:S$=S$+CHR$A%:PROCS1:VDUA%:P
ROCS2:A%=FNCH:UNTIL A%=dquote ORA%=eoln:
IFA%=eoln PROCERR ELSE PROCA
2270 IFA%=dquote GOTO2260
2280 T%=string:ENDPROC
2290 DEFPROCTG
2300 T%=A%:PROCA
2310 IFT%=gt ANDA%=eq T%=geq:PROCA
2320 IFT%=lt ANDA%=eq T%=leq:PROCA
2330 IFT%=lt ANDA%=gt T%=neq:PROCA
2340 IFT%=col ANDA%=col REPEATPROCA:UNT
IL(A%<>col)AND(A%<>spc)
2350 IFT%=col ANDA%=eoln:T%=A%:PROCA
2360 ENDPROC
2370 DEFPROCNL
2380 LI%=A%*256+FNCH:LL%=FNCH
2390 lino(NL%)=LI%:lina(NL%)=P%
2400 NL%=NL%+1:PROCS1
2410 PRINT';LI%; TAB(4);:A%=32
2420 PROCS2:ENDPROC
2430 DEFPROCS
2440 SN%=-1:REPEAT SN%=SN%+1
2450 UNTIL (st$(SN%)=S$ AND stp(SN%)=T%
) OR st$(SN%)=""
2460 IFst$(SN%)="" st$(SN%)=S$:NS%=NS%+
1:stp(SN%)=T%:IFT%=id sa(SN%)=vmem:vmem=
vmem-5
2470 ENDPROC
```

ROM Utilities for all

Judging by the number of new releases, there is still a healthy market for ROM based utilities. Geoff Bains and Alan Webster have been looking at some of the latest on offer.

Product : Transferom
Supplier : Watford Electronics
250 High Street,
Watford WD1 2AN.
0923-37774
Price : £25.30
Reviewer : Geoff Bains

Product : Advanced Disc Toolkit
Supplier : Advanced Computer Products
6 Ava House, High Street,
Chobham, Surrey.
Telephone 0276-76545
Price : £34.50
Reviewer : Alan Webster

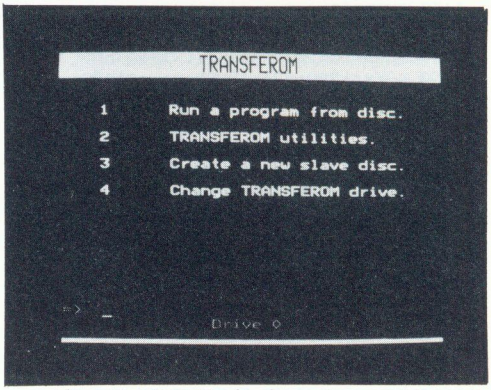
Product : Replay
Supplier : Vine Micros
Marshborough,
Sandwich, CT13 0PG
0304-812276
Price : £35.00
Reviewer : Geoff Bains

Product : NLQ Designer
Supplier : Watford Electronics
250 High Street,
Watford WD1 2AN.
0923-37774
Price : £28.75
Reviewer : Geoff Bains

Product : ROMAS
Supplier : Watford Electronics
250 High Street, Watford.
Telephone 0923-37774
Price : £51.75
Reviewer : Alan Webster

TRANSFEROM from Watford

Transferom is Watford's answer to Vine Micros' TDROM (see the review in BEEBUG Vol.4 No.1). Like that ROM, Transferom will put most of your tape programs conveniently onto disc.



Like the TDROM, it is not capable of transferring every tape program around. However, it does contain a few extras over and above TDROM. Files of any length can be built up. Even the longest adventure game can be accommodated. If there is no room left on one disc then you can save the rest onto another; Transferom looks after all the details. Another nice feature of this ROM is that you can change the name assigned to a game file on disc after it is created.

The whole package is menu driven and simple to use. As a tape-to-disc transfer utility Transferom scores at about the same level as TDROM, maybe a little higher. However, with items such as Replay (see below) around, for not much more, it is somewhat overshadowed.

ADVANCED DISC TOOLKIT from ACP

Advanced Computer Products (ACP) have been quietly making a name for themselves in the Electron market with a number of quality products including ROM software and RAM and ROM adaptors. Now they are branching into the BBC field with a second processor, DFS, ADFS and Electron compatible Disc Toolkit.

The toolkit adds over 30 new 'disc' commands to your Beeb or Electron, although ACP have fallen into the trap of filling surplus ROM space with a miscellany of general command functions. This is regrettable when you consider the many other disc commands which users might have preferred. As Beeb owners will see from the table, most of these new commands are similar to those in the Acorn or Watford DFS, and Computer Concepts' Disc Doctor. Electron owners who have not

BACKUP	**MAP**
BFIND	**MDUMP**
BUILD	**MENU**
CATALL	**MEX**
DCOMP	**MFIND**
DEX	**MLOAD**
DFIND	**MOVE**
DIRALL	**MRUN**
DUMP	**ROMS**
ENVELOPE	**SECTORS**
FCOMP	**SETADR**
FCOPY	**SPT**
FORM	**SWAP**
FREE	**TYPE**
FSN	**UNPLUG**
KEYL	**VERIFY**
LIST	**XFER**

benefitted from the above products, will find more to interest them.

Some of the better commands contained within the ROM are DCOMP which compares two discs and reports on any differences, FCOMP which is a verify command to compare two files, FCOPY to create a renamed copy of a file, and XFER to transfer a program between two filing systems.

For a disc toolkit such functions as an envelope lister, ROM lister, Basic program string search, function key lister, and memory dump seem out of place.

Overall, though, this is a very handy utility, combining all the best features of the DFS's and disc toolkits on the BBC micro, and at £34.50 for a 16k EPROM with a host of useful commands represents very good value for money.

REPLAY from Vine Micros

Replay is a very different kind of sideways ROM as it includes additional (and attached) hardware. Replay is a utility that will allow you to save the complete status of your Beeb to disc at any time. This can then be loaded into the machine again and the program continued from where you left off.

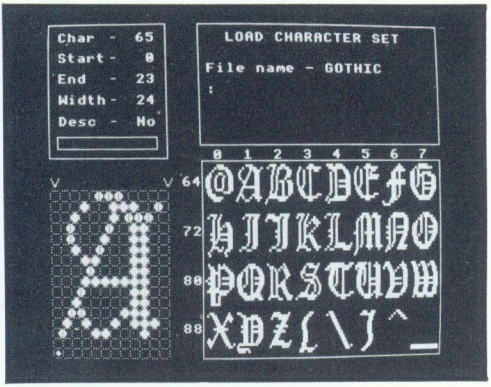
The prime use for Replay is to transfer tape programs to disc. First a special file on a disc is created. Then Break or Escape is pressed and the tape program (probably a game) is loaded in the



normal way. Next you wait until the game is fully loaded and press Replay's own trigger button. This initiates the saving, to disc, of the entire contents of the Beeb's 32K RAM and the status of the various control chips. Later on, you can re-enter Replay and choose to reload the file saved. Replay will then return your machine to exactly the state it was in when the button was pressed and you can play the game.

Replay can save the Beeb's contents during any program regardless of where it came from - tape or disc. This means that any game (or other program) can be transfered to your own disc. The ROM can also be used to just freeze a game and it can display the screen at the time that the RAM was saved. This makes dumping your favourite game's screen to the printer an easy task. You can even edit the file created on disc to make changes to the game. Vine Micros supply the relevant data to make a few changes, such as a blue background in Elite and infinite lives in Frak.

The only problem with Replay is the hardware. Apart from the ROM - along with with several flying leads - there is the trigger switch, a circuit board on the end of a flying lead, a replacement link for switch S21 on the board, and a two header plugs to jam over the relevant chips in your BBC. The methods chosen for the connection of Replay to your machine do mean that its installation is never permanent. It also means that the connections are never very reliable. I'd be inclined to solder mine into position, but that's up to you.



NLQ DESIGNER from Watford

The advent of the Kaga-Taxan (or Canon) NLQ printers has created a lot of interest amongst all computer users, especially those with a BBC micro. The high resolution characters produced by these machines are a worthy rival to the traditional daisy wheel printer. However, these machines also offer the facility of a downloadable alternative NLQ character set. To make use of this your printer must be fitted with a 'PCG RAM' kit - just a 6264 RAM chip. The NLQ Designer ROM from Watford Electronics is a utility to enable you to make full use of this facility.

The NLQ Designer is basically a character designer very similar to those well known to BBC owners. The NLQ characters are printed on paper using a 23 x 18 character grid in two passes. However, NLQ characters are designed on a 12 x 16 grid. The cursor keys move around the grid and the space bar toggles individual 'pixels' on and off. They can be designed on this smaller grid as each cell represents two pixels offset horizontally from one another and overlapping. The vertical discrepancy is accounted for by the characters with descenders (such as 'g' and 'p'). These are printed two pixels lower than the others.

As a character is defined its various properties are displayed. These are the horizontal start and end positions on the grid, the width (for the proportional spacing) and whether the character is to be printed as one with a descender. As a character is designed the accompanying characters in its group are displayed. Once designed the entire group of characters can be saved to cassette or disc. Similarly other groups can be loaded for further editing. Watford provide three character sets (Bold, Courier, and Gothic) on the disc accompanying the NLQ Designer ROM.

For anyone with a Beeb and a Kaga-Taxan KP810 or Canon PW1080 this ROM provides a valuable extension to the already impressive facilities of these printers.

ROMAS from Watford

ROMAS is a cross assembler and macro editor that will generate object code for the Mostek 6502 and 6511, Rockwell 65C02, Zilog Z8 and Z80, Intel 8085, 8041 and 8048 and Motorola 6809 processors.

ROMAS features a comprehensive screen editor which works in all 40 and 80 column modes. The editor is split into two windows with a status/command window occupying the first three lines on the screen and the editor taking up the rest.

In addition to the EPROM, the package consists of a disc, a comprehensive 160 page A5 manual and function key strip. ROMAS has to be used in conjunction with a disc drive as the assembler reads source code directly from disc. The EPROM contains the 3 versions of the assembler that support the 6502, 6511 and 8085 processors. The other 6 are held on disc.

All of the routines in ROMAS are invoked through * commands. The assembler is entered using *ASM and the editor is entered using *EDT or *BEDT. Further commands available are *LOCATE to relocate assembled 6502 code, *CLOSE to close all the currently open files, *INTEL and *HEXCONV to convert a hex code file into an INTEL standard hex file, and *TRANSMIT, *RECEIVE, *DATA and *DOWNLOAD to set the baud rate of the RS423 and download files to other machines.

The EPROM contains a useful cross-referencer (*XREF) while the disc contains the source code of a 6502 disassembler.

ROMAS is certainly a comprehensive cross assembler and at only £51.75 represents excellent value for the serious user.

Programming with

Wordwise Plus

(part 2)

Stephen Ibbs continues his series on programming in Wordwise Plus by looking at the role that loops have to play, again illustrated with useful hints and examples.

Last month a problem was posed concerning the small routine to delete the current line of text (reprinted below). The question was how to extend it so that the two lines, now joined together, are separated by one space, not two or none as can often be the case.

```
SELECT TEXT
CURSOR AT 39
DELETE LEFT ?&7E
DELETE AT
DISPLAY
```

If we imagine three text lines, A, B, and C, with line B now deleted, then calling this routine will leave the cursor underneath the first character of line C. We need to check if this character, and the one to its left (the last on line A) are spaces so the following lines are added:

```
CURSOR LEFT
A$=GCT$
B$=GCT$
CURSOR LEFT
IF A$=" " AND B$=" " THEN DELETE AT
IF A$<>" " AND B$<>" " THEN TYPE ""
DISPLAY
```

This places the cursor back underneath the first character of line C, with A$ and B$ holding the 2 characters. A single space will have been inserted (to the left of the cursor), if there was none, or deleted if there were 2. You may like to develop this further, to cope with deleting the top line of a piece of text, because when deleted, the routine can't obey the CURSOR LEFT because it will be at the very start of text (SOT).

HINT: The TYPE command inserts to the LEFT of the cursor position, the same as when inserting text when in EDIT mode.

Last month a hint was given about inserting the DISPLAY command into routines so that you can see them at work. Users of Wordwise Plus later than version 1.48 (as most are) have an alternative command, DISPLAY1. It is difficult to explain how this works, the best way being by the following example loaded into segment 0:

```
SELECT TEXT
REPEAT
A$=GCT$
DISPLAY
UNTIL A$=CHR$13
```

Place the cursor within some text, call the routine and the screen flickers as the routine repeats itself. Now change DISPLAY to DISPLAY1 and call the routine again. This time there's no flicker. DISPLAY1 will not always eliminate flicker (especially if the screen is being updated causing the screen to scroll). However, it is always well worth trying.

HINT: Try DISPLAY1 instead of DISPLAY, especially when developing routines. It (usually) avoids screen flicker.

The above routine introduces the REPEAT-UNTIL commands, very similar to their Basic counterparts in that the set of instructions enclosed by them will repeat until the terminating condition is met. Quite often this condition will be the start or end of text (SOT or EOT). A simple illustration of a loop, also using the commands so far discussed is given below. It will join together two paragraphs separated by one or more blank lines.

The routine consists of three sections. The first places the cursor at the end of the upper of the two lines to be joined together. If it finds a character at position 39 the cursor is moved one place to the right. The second section is the REPEAT-UNTIL loop that takes a character, checks to see if it is a space or a Return, then deletes it. Note that it deletes left, because the cursor moves one position to the right, as has been stressed before. The final section is the same as for the 'delete line' routine

- a check to ensure the lines are separated by one space.

```
SELECT TEXT
CURSOR AT 39
IF ?&7E=39 THEN CURSOR RIGHT
REPEAT
A$=GCT$
IF A$=CHR$13 OR A$=" " THEN DELETE LEFT
UNTIL A$<>CHR$13 AND A$<>" "
CURSOR LEFT 2
A$=GCT$
B$=GCT$
CURSOR LEFT
IF A$=" " AND B$=" " THEN DELETE AT
IF A$<>" " AND B$<>" " THEN TYPE " "
DISPLAY
```

This routine could also be developed further. At the moment it cannot cope if the upper of the two lines being joined ends (somewhat unusually) with more than one space.

Wordwise Plus does not have the FOR-NEXT commands of Basic. It is possible to use the REPEAT-UNTIL commands to repeat a routine, say, five times:

```
A%=0
REPEAT
(main routine)
A%=A%+1
UNTIL A%=5
```

However, it's more convenient to use the DOTHIS-TIMES commands which really speak for themselves. It is similar to REPEAT-UNTIL in that it can 'DOTHIS' until some criteria has been met, e.g. DOTHIS - TIMES A%, where A% has been given a value thus:

```
DOTHIS
(main routine)
TIMES 5
```

To illustrate their use further, we will write a very short routine to print multiple copies of the text. In addition, this will use the commands PRINT and, importantly, DEFAULTS. First we must establish how many copies are wanted, so the question needs to be presented on the screen:

```
CLS
PRINT "How many copies? "
A%=GET-48
```

The CLS is self-explanatory, but it is worth emphasizing that clearing the screen like this does NOT destroy any text. The question is displayed and the third line will wait for a number to be typed in. It will react to the first key pressed, so only single numbers can be inserted. The GET command returns the ASCII value (just like Basic), hence the GET-48 so that A% contains the value we actually want. If we want the routine to be able to print more than 9 copies at a time we would have to use the following:

```
CLS
PRINT "How many copies? "
A%=VAL(GLK$)
```

This waits for you to type in the number, e.g. 25, and doesn't do anything until Return is pressed. Then A% takes on the VALue returned by GLK$. The GLK$ means 'Get Line From Keyboard' and allows you to type in up to 255 characters and place it in a string variable, if required. Note that neither of these question options has any error trapping to prevent you typing in letters, nor a 'Y/N' check in case you type in the wrong value.

The routine would then continue with the lines:

```
SELECT TEXT
DOTHIS
DEFAULTS
PRINT TEXT
TIMES A%
DISPLAY
```

Note the DEFAULTS command. This resets all the formatting commands back to their original values. If this was omitted, the second printed copy of the text, assuming EP was set, would not start at page one and all your formatting could be upset.

HINT: Don't forget the DEFAULTS command when using PRINT or PREVIEW, otherwise the formatting could be upset.

The PRINT command is possibly one of the most useful Wordwise Plus commands, because it can be qualified by a second word, PAGE. So if for example you have a 13 page document and you slightly edit page 12 it is obviously wasteful to reprint the entire document. Similarly it is not easy to work out where the markers should go to just print page 12. What you can do is go to the menu and type:

```
:DEFAULTS <Return>
:PRINT PAGE 12 <Return>
```

and just that page will be printed. Note again the DEFAULTS command. If you just want to check what page 12 looks like you would type PREVIEW instead of PRINT, but unlike menu option 7, the screen will immediately clear when the preview has taken place.

As well as being useful when dealing with text already in memory, the PRINT and PREVIEW commands can cope with text on file, whether disc or tape, by following the command with FILE and the filename in quotation marks. Omitting DEFAULTS deliberately between separate PRINT statements can be useful if you want to print related files with the page numbers continuing in sequence e.g.:

```
DEFAULTS
PRINT FILE "filename 1"
PRINT FILE "filename 2"
PRINT FILE "filename 3"
DISPLAY
```

Earlier, the two commands SOT and EOT were mentioned, and a simple illustration of how EOT can be used effectively is given below. When typing in text, it is a chore to have to press Return twice and then Tab, to separate each paragraph. It would be easier to simply insert a little used character, say '{', wherever a new paragraph was needed. When all the typing is finished a routine could be called to do the job automatically:

```
SELECT TEXT
CURSOR TOP
REPEAT
FIND "{"
TYPE "|R|R|T"
DELETE AT
UNTIL EOT
DISPLAY
```

CURSOR TOP starts the routine from the start of the text. We now need to find the first occurrence of the symbol, using FIND, which places the cursor underneath the {, and then type in the Tab. A Return is typed in with |R and |T gives the Tab. Finally, the DELETE AT removes the '{' from the text. Remember that TYPE inserts characters to the left of the cursor so it's still under the {. This routine is repeated until the end of text. However, if you run this routine, you will find an error. An extra Tab has been inserted at EOT, so to stop this happening we must use the FALSE command, and modify the fifth line to:

```
IF EOT=FALSE THEN TYPE "|R|R|T"
```

In other words: 'if it isn't the end of text type in the Tab'.

HINT: When using REPEAT-UNTIL loops, make sure that you check for the SOT and EOT otherwise the routine may 'lock-up' and/or you may get unexpected results.

Wordwise Plus supports procedures. These work very much as their Basic counterparts, except that the name is not defined with DEFPROC. Instead, it is marked with a full stop. The procedure is called in the same way as in Basic, so 'PROC start' would call a routine marked with '.start' and run from there until ENDPROC, at which point it returns to the line following the PROC command. When using procedures, they are usually put after the end of the main routine to make 'debugging' easier. However, this does mean that the program end must be marked with an END command so that the computer doesn't accidentally run on into a procedure. Under other circumstances the END command is entirely optional.

As a problem for next month, have a look at the chapter-numbering routines published on page 17 of the June 1985 BEEBUG (Vol.4 No.2) and convert them to a single routine using procedures. You will also need to insert questions, as we have done here, so that the user can select which 'sub-routine' he wants.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS

EASIER BYTES - Jonathan Temple

When using VDU commands, or similar, it is easier, quicker, and takes up less program space to use -1 instead of 255.

Hidden Line removal

Turn your wireframe displays into solid-looking 3D objects with this technique for hidden line removal from Andrew Barnes and Simon Cooke.

Representing 3D objects on micros is usually restricted to wireframe drawings (see 3D ROTATION, BEEBUG Vol.1 No.10). Wireframe drawings serve their purpose but can be confusing. An obvious step nearer to reality would be the removal of hidden lines. However, the extra complexity of achieving this is usually dismissed as a task for mainframes.

This program will display a perspective wireframe projection of any suitable object from any angle about the vertical and horizontal axes and then optionally remove all the hidden lines from the new display.

USING THE PROGRAM

When run, the program spends a short time setting itself up to display the defined object and then the user is prompted to enter the orientation in which the object is to be drawn, in the form: angle of rotation about the object and angle above the flat plane (both in degrees). The object is then drawn - first as a wireframe diagram and then with hidden lines removed if desired, always with true perspective.

When the hidden line display is being created you can halt the image generation by pressing the Shift key and move on to the next step using the space bar. It will soon become obvious how much clearer even a simple object can become when it is drawn as a solid in this way. If you want a printout of the display at any time, Escape will activate the screen dump if one has been included (line 2890).

DEFINING AN OBJECT

The object is defined in DATA statements at the end of the program. Lines 2920 and 3000 must be included in any data, preceeding the point data and surface data. The first part of the data defines points on the shape in the form 'X,Y,Z'. Firstly, the number of points must be given, in line 2930 (39 in this example). This should be followed by the X,Y,Z co-ordinates for each point. Then comes the data to define how these points are linked to form surfaces. Each surface is entered as the point numbers, in order around the edge of the surface, separated by a hyphen. The data for each surface is separated by a comma. Note that the starting and finishing point of the surface need only be entered once.

A surface may be defined with any number of points, but to simplify the program, defined surfaces may not contain intruding points. However, such surfaces may still be catered for by entering them as two or more separate surfaces (which the program can handle) joined without the normal joining line (see diagram).

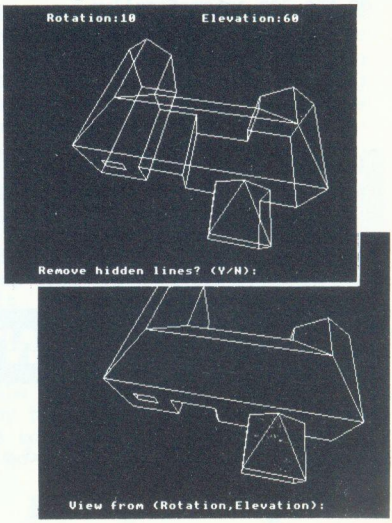
To specify such a join, the hyphen between the 'missing' line's two end points should be replaced by a full stop. This process may seem complex but should be made clearer by an example: The surface in the diagram could be represented by:

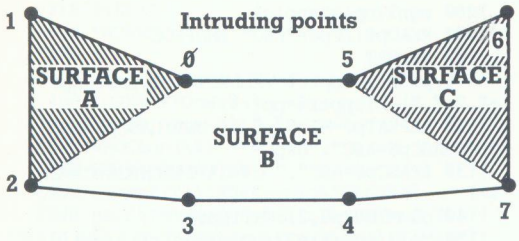
0-1-2. , 7.5-6- , 3-2.0-5.7-4-

When testing or debugging the program it is a good idea to add a 'DATA END' statement early on to reduce the amount of data which the program has to handle and speed up test runs. The data provided in the listing here will produce a display of a house.

ALGORITHM

Taken at its simplest, the program must draw the faces of an object which lie at the back first and the ones nearest last, to conceal the lines which should not be visible by filling in each face as





it is drawn. The obvious problem is how a computer can tell where surfaces lie in relation to the observer and each other in a useful way, and then decide on the order in which to draw them. The method described here roughly follows the pattern of a recursive sort where 'in front of' and 'behind' replace 'less than' and 'greater than'.

The algorithm used starts with an un-ordered list of the surfaces. From this list a surface is selected such that no other surface has points both in front and behind the chosen surface. If this is not possible, any surface can be chosen, but each other surface which has points lying on both sides of the chosen surface has to be split into two parts, one 'in front' and one 'behind'. These two new surfaces replace the original one in the list. The list is then rearranged to consist of first those behind the chosen surface, then the surface itself and those in the same plane, and finally those lying in front. This is effectively taking the whole shape and splitting it into two shapes, one of which is behind the chosen surface (drawn first) and the other in front (drawn last). The parts of the list corresponding to each 'new' shape must now be sorted using the same method as before. Since these new shapes can themselves be split to form new shapes the procedure lends itself to recursion. The whole process ends when each new 'shape' consists of only one surface. The surfaces are then drawn in the order given by this sorted list.

PROGRAM NOTES

Main procedures:

PROCgetdata reads the data. Definitions of points are read into arrays X,Y and Z, and definitions of surfaces are read into memory starting from pldef%.

PROCeq calculates coefficients of equations of planes, in arrays A%,B%,C%,D%.

PROCmc sets up two short machine code routines, rd and wr, which are used to quickly access the 'bit compressed' table (stored in memory arr%) of relationships between surfaces.

PROCarray sets up the table starting at arr%. The table is effectively a two dimensional array with which the numbers of two surfaces are used to look up a two bit element determining their relationship in space.

PROCtransform(th,ph) rotates all points by th degrees about the vertical axis of the display and ph degrees about the horizontal axis, and carries out a perspective transformation leaving the resulting co-ordinates in arrays X% and Y%.

PROCoutline(A%,col%) draws outline of surface A% in colour, col%.

PROCorder(F%,T%) sorts part of the array 'list' starting at F% and ending at T%. Using the surfaces between these limits it selects one and sorts the other surfaces into those in front and those behind. It then calls itself twice to sort out these sub-sections of the list.

PROCdraw(A%) draws surface A%.

PROCrel(X%,Y%) works out relation between two surfaces and stores it in an array.

FNside(A%,B%,D%) returns TRUE if surface B% is on the side of surface A% specified in D%.

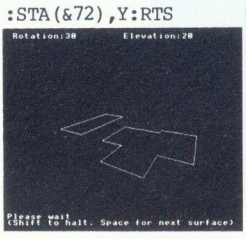
PROCdivideAbyB(A%,B%) splits surface A% in two along the plane of surface B%.

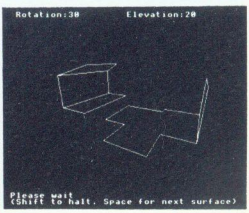
FNsplit(p1%,p2%,m,n) calculates a point which divides the line between the points p1% and p2% in the ratio m:n and transforms it.

PROCscreendump a call to a screendump routine (e.g. *GDUMP) inserted at line 2910 is initiated by pressing Escape.

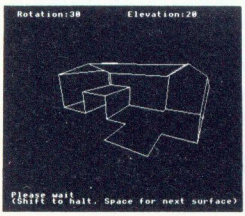
```
  10 REM Program 3D Hidden line removal
  20 REM Version B1.0
  30 REM Authors S. Cooke & A. Barnes
  40 REM BEEBUG  Nov 1985
  50 REM Program subject to copyright
  60 :
 100 MODE4
 110 scale=700:dist=200
 120 mp%=100:mpl%=100
 130 DIMX%(mp%),Y%(mp%)
 140 DIMX(mp%),Y(mp%),Z(mp%)
 150 DIMpldefs%500,pp%(mpl%):pptr%=0:np
ls%=0
 160 DIMlist%mpl%,list2%mpl%
 170 DIMVrel%mpl%
```

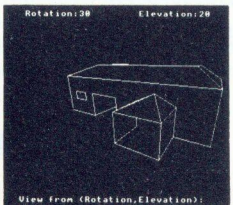
```
  180 DIMeqn%mpl%,col%(mpl%)
  190 :
  200 VDU29,640;512;28,0,31,39,29,24,-64
0;-416;639;479;
  210 PROCgetdata
  220 pnpls%=npls%:pnpts%=npts%
  230 scale=700:dist=200
  240 PROCtransform(30,40)
  250 FORA%=0TOnpls%-1:PROCoutline(A%,1)
:NEXT
  260 :
  270 neq%=npls%
  280 DIMA%(neq%-1),B%(neq%-1),C%(neq%-1
),D%(neq%-1)
  290 PROCeq
  300 DIMarr%mpl%*(neq%-1)DIV4
  310 DIMmc%200
  320 PROCmc
  330 PROCarray
  340 :
  350 ONERROR IF FNerror END ELSEGOTO350
  360 REPEAT
  370 CLS:INPUT"View from (Rotation,Elev
ation):"rot,elev
  380 VDU28,0,0,39,0,12:PRINT;" Rotation
:";rot,"Elevation:";elev;:VDU28,0,31,39,
29,10
  390 npls%=pnpls%:npts%=pnpts%
  400 PROCtransform(rot,elev)
  410 CLG
  420 FORA%=0TOnpls%-1
  430 list%?A%=A%:PROCoutline(A%,1)
  440 Vrel%?A%=(SGN(A%(A%)*Vx+B%(A%)*Vy+
C%(A%)*Vz+D%(A%))+1)DIV2
  450 NEXT
  460 CLS:PRINT"Remove hidden lines? (Y/
N):";:REPEATA$=GET$:UNTILA$="N"ORA$="Y":
IFA$="N" GOTO370
  470 PRINT''"Please wait"''"(Shift to ha
lt. Space for next surface)"
  480 lt%=npls%
  490 TS=0
  500 PROCorder(0,npls%)
  510 CLG:*FX15
  520 FORA%=0TOlt%-1:PROCdraw(list%?A%):
IF INKEY(-1) THEN REPEAT UNTIL GET=32
  530 NEXT A%
  540 UNTILFALSE
  550 :
 1000 DEF PROCgetdata
 1010 PRINTTAB(15)"Please wait"
 1020 READpt$:IFLEFT$(pt$,10)<>"Point da
ta" STOP
 1030 READnpts%
 1040 FORA%=0TOnpts%-1:READX(A%),Y(A%),Z
(A%):NEXT
 1050 READpl$:IFLEFT$(pl$,12)<>"Surface
data" STOP
 1060 pp%(npls%)=pptr%
 1070 C%=0
 1080 eqn%?npls%=npls%
 1090 READp$:IFp$="END" ENDPROC
 1100 REPEAT
 1110 pldefs%?pptr%=VALLEFT$(p$,INSTR(p$
+".",".")-1):pptr%=pptr%+1
 1120 REPEATp$=MID$(p$,2):UNTILASCp$=ASC
"-"ORASCp$=ASC"."ORp$=""
 1130 IFASCp$=ASC"." C%=C%+C%ELSEC%=C%+C
%+1
 1140 p$=MID$(p$,2):UNTILp$=""
 1150 col%(npls%)=C%:npls%=npls%+1:pp%(n
pls%)=pptr%
 1160 GOTO1070
 1170 :
 1180 DEF PROCeq
 1190 FORN%=0TOnpls%-1
 1200 p%=pp%(N%)
 1210 x%=X(pldefs%?p%):y%=Y(pldefs%?p%):
z%=Z(pldefs%?p%)
 1220 A%(N%)=(Y(pldefs%?(p%+1))-y%)*(Z(p
ldefs%?(p%+2))-z%)-(Y(pldefs%?(p%+2))-y%
)*(Z(pldefs%?(p%+1))-z%)
 1230 B%(N%)=(Z(pldefs%?(p%+1))-z%)*(X(p
ldefs%?(p%+2))-x%)-(Z(pldefs%?(p%+2))-z%
)*(X(pldefs%?(p%+1))-x%)
 1240 C%(N%)=(X(pldefs%?(p%+1))-x%)*(Y(p
ldefs%?(p%+2))-y%)-(X(pldefs%?(p%+2))-x%
)*(Y(pldefs%?(p%+1))-y%)
 1250 D%(N%)=-A%(N%)*x%-B%(N%)*y%-C%(N%)
*z%
 1260 NEXT
 1270 ENDPROC
 1280 :
 1290 DEFPROCmc
 1300 FORpas%=0TO2STEP2:P%=mc%:[OPTpas%
 1310 .rd
 1320 LDA&448:STA&70:LDA&449:LSRA:ROR&70
:LSRA:ROR&70:STA&71:LDA&70:CLC:ADC #arr%
MOD256:STA&72:LDA&71:ADC#arr%DIV256:STA&
73:LDY#0:LDA(&72),Y:STA&74
 1330 LDA&448:AND#3:TAY:LDA&74
 1340 CPY#0:BEQrts:.loop:LSRA:LSRA:DEY:B
NEloop:.rts:RTS
 1350 .wr
 1360 LDA&448:STA&70:LDA&449:LSRA:ROR&70
:LSRA:ROR&70:STA&71:LDA&70:CLC:ADC#arr%M
OD256:STA&72:LDA&71:ADC#arr%DIV256:STA&7
3:LDY#0:LDA(&72),Y:STA&74
 1370 LDA#3:STA&70:LDA&444:STA&71
 1380 LDA&448:AND#3:TAY
 1390 CPY#0:BEQmsk:.loop:ASL&70:ASL&70:A
SL&71:ASL&71:DEY:BNEloop:.msk
 1400 LDA&70:EOR#&FF:AND&74:ORA&71:LDY#0
:STA(&72),Y:RTS
```





```
 1410 ]:NEXT
 1420 ENDPROC
 1430 :
 1440 DEF PROCarray
 1450 FORX%=0TOneq%-1
 1460 PRINT;neq%-X%;" ";
 1470 PROCeqn(X%)
 1480 FORY%=0TOnpls%-1
 1490 P%=pp%(Y%):R%=X%+neq%*Y%:Q%=0
 1500 REPEATN%=pldefs%?P%:Q%=Q%OR((SGN(cA%*X(N%)+cB%*Y(N%)+cC%*Z(N%)+cD%)+3)MOD3):P%=P%+1:UNTILQ%=3ORP%=pp%(Y%+1)
 1510 CALLwr
 1520 NEXT:NEXT:CLS:ENDPROC
 1530 :
 1540 DEF PROCrel(X%,Y%)
 1550 LOCALP%,Q%
 1560 P%=pp%(Y%)
 1570 Q%=0
 1580 REPEAT:Q%=Q%OR((SGNFNeq(pldefs%?P%)+3)MOD3):P%=P%+1:UNTILQ%=3ORP%=pp%(Y%+1)
 1590 R%=X%+neq%*Y%:CALLwr:ENDPROC
 1600 ENDPROC
 1610 :
 1620 DEFPROCorder(F%,T%)
 1630 LOCALTl%
 1640 IFF%+1>=T% ENDPROC
 1650 E%=F%
 1660 REPEATX%=eqn%?(list%?E%):E%=E%+1
 1670 P%=F%:REPEATR%=X%+neq%*list%?P%:P%=P%+1:UNTIL(USRrd AND3)=3ORP%=T%
 1680 UNTIL(P%=T%AND(USRrd AND3)<>3)ORE%=T%
 1690 FORP%=0TOT%-F%-1:list2%?P%=list%?(P%+F%):NEXT
 1700 T2%=T%-F%
 1710 IF(USRrd AND3)<>3 Fl%=F%:Tl%=F%+T2%:GOTO1770
 1720 FORP%=0TOT%-F%-1:Y%=list2%?P%:R%=X%+neq%*Y%
 1730 IF(USRrd AND3)=3 PROCdivideAbyB(Y%,X%):list2%?T2%=npls%-1:T2%=T2%+1:list2%?P%=npls%-2
 1740 NEXT
 1750 Fl%=F%:Tl%=F%+T2%
 1760 FORP%=lt%-T%-1TO0STEP-1:list%?(P%+Tl%)=list%?(P%+T%):NEXT:lt%=lt%+Tl%-T%:T%=Tl%
 1770 FORP%=0TOT%-F%-1:Y%=list2%?P%:R%=X%+neq%*Y%
 1780 IF(USRrd AND3)=Vrel%?X%+1 list%?Fl%=Y%:Fl%=Fl%+1
 1790 IF(USRrd AND3)=2-Vrel%?X% Tl%=Tl%-1:list%?Tl%=Y%
 1800 NEXT
 1810 N%=Fl%
 1820 FORP%=0TOT%-F%-1:Y%=list2%?P%:R%=X%+neq%*Y%
 1830 IF(USRrd AND3)=0 list%?N%=Y%:N%=N%+1
 1840 NEXT
 1850 IFN%<>Tl% STOP
 1860 Tl%=Tl%-lt%:T%=T%-lt%
 1870 IFFl%-F%>1:PROCorder(F%,Fl%)
 1880 IFT%-Tl%>1:PROCorder(lt%+Tl%,lt%+T%)
 1890 ENDPROC
 1900 :
 1910 DEF FNside(A%,B%,D%)
 1920 LOCALN%,P%
 1930 s%=0
 1940 cA%=A%(eqn%?A%):cB%=B%(eqn%?A%):cC%=C%(eqn%?A%):cD%=D%(eqn%?A%)
 1950 vS%=SGN(cA%*Vx+cB%*Vy+cC%*Vz+cD%)
 1960 N%=pp%(B%)
 1970 pS%=SGN(FNeq(pldefs%?N%))
 1980 IFvS%=0 vS%=D%*pS%
 1990 IFvS%*pS%=-D%:tt=tt+TIME-t:=FALSE
 2000 N%=N%+1:IFN%<>pp%(B%+1) GOTO1970
 2010 =TRUE
 2020 :
 2030 DEF PROCdraw(A%)
 2040 LOCALC%
 2050 GCOL0,1
 2060 start%=pldefs%+pp%(A%)
 2070 X0%=X%(?start%):Y0%=Y%(?start%):MOVEX%(start%?1),Y%(start%?1):MOVEX0%,Y0%
 2080 FORZ%=2TOpp%(A%+1)-pp%(A%)-1:PLOT85,X%(start%?Z%),Y%(start%?Z%):MOVEX0%,Y0%:NEXT
 2090 PROCoutline(A%,0):ENDPROC
 2100 DEFPROCoutline(A%,col%):GCOL0,col%
 2110 C%=col%(A%)
 2120 MOVEX%(pldefs%?pp%(A%)),Y%(pldefs%?pp%(A%)):FORB%=pp%(A%+1)-1TOpp%(A%)STEP-1:PLOT4+(C%AND1),X%(pldefs%?B%),Y%(pldefs%?B%):C%=C%DIV2:NEXT
 2130 ENDPROC
 2140 :
 2150 DEF PROCtransform(th,ph)
 2160 Cth=COSRADth:Sth=SINRADth
 2170 Cph=COSRADph:Sph=SINRADph
 2180 FORA%=0TOnpts%-1:PROCtrans(A%):NEXT
 2190 Vz=dist*Cph:Vy=dist*Sph:Vx=Vz*Sth:Vz=Vz*Cth
 2200 ENDPROC
 2210 :
 2220 DEF PROCtrans(A%)
 2230 Zl=Z(A%)*Cth+X(A%)*Sth
 2240 X=-Z(A%)*Sth+X(A%)*Cth
 2250 Y=-Zl*Sph+Y(A%)*Cph
 2260 Z=Zl*Cph+Y(A%)*Sph
```





```
 2270 S=scale/(dist-Z)
 2280 X%(A%)=X*S:Y%(A%)=Y*S
 2290 ENDPROC
 2300 :
 2310 DEF PROCeqn(N%)
 2320 cA%=A%(N%):cB%=B%(N%):cC%=C%(N%):c
D%=D%(N%):ENDPROC
 2330 :
 2340 DEF PROCdivideAbyB(A%,B%)
 2350 TIME=0
 2360 LOCALN%,M%
 2370 Co%=col%(A%):Cn%=0
 2380 PROCeqn(B%)
 2390 np%=pp%(A%+1)-pp%(A%)
 2400 ct%=2^(np%-1)
 2410 start%=pp%(A%)+pldefs%
 2420 nextpt%=pp%(npls%)
 2430 ol%=-1:o2%=-1
 2440 d2=FNeq(start%?0):s2=SGNd2
 2450 FORN%=0TOnp%-1
 2460 M%=(N%+1)MODnp%
 2470 d1=d2:s1=s2
 2480 d2=FNeq(start%?M%):s2=SGNd2
 2490 IFs1=0 pldefs%?nextpt%=start%?N%:C
n%=Cn%+Cn%:nextpt%=nextpt%+1
 2500 pldefs%?nextpt%=start%?N%:Cn%=Cn%+
Cn%-((Co%ANDct%)<>0):nextpt%=nextpt%+1:I
Fs1=0:ol%=o2%:o2%=nextpt%-1
 2510 IFs1*s2=-1 pldefs%?nextpt%=FNsplit
(start%?N%,start%?M%,d1,d2):Cn%=Cn%+Cn%:
nextpt%=nextpt%+1:pldefs%?nextpt%=pldefs
%?(nextpt%-1):Cn%=Cn%+Cn%-((Co%ANDct%)<>
0):nextpt%=nextpt%+1:ol%=o2%:o2%=nextpt%
-1
 2520 Co%=Co%+Co%
 2530 NEXT
 2540 FORN%=0TOol%-pp%(npls%)-1:pldefs%?
(nextpt%+N%)=pldefs%?(pp%(npls%)+N%):NEX
T
 2550 FORN%=pp%(npls%)TOnextpt%-1:pldefs
%?N%=pldefs%?(N%+ol%-pp%(npls%)):NEXT
 2560 Cn%=(Cn%DIV(2^(nextpt%-o2%)))+2^(o
2%-pp%(npls%))*(Cn%MOD(2^(nextpt%-o2%)))
 2570 npls%=npls%+1:pp%(npls%)=pp%(npls%
-1)+o2%-ol%
 2580 col%(npls%-1)=Cn%AND(2^(o2%-ol%)-1
)
 2590 eqn%?(npls%-1)=eqn%?A%
 2600 npls%=npls%+1:pp%(npls%)=nextpt%
 2610 Cn%=Cn%DIV(2^(o2%-ol%))
 2620 col%(npls%-1)=Cn%AND(2^(nextpt%-o2
%+ol%-pp%(npls%-3))-1)
 2630 eqn%?(npls%-1)=eqn%?A%
 2640 LOCALX%
 2650 FORX%=0TOneq%-1
 2660 R%=X%+neq%*A%
 2670 Q%=(USRrd AND3):IFQ%<>3:R%=X%+neq%
*(npls%-2):CALLwr:R%=R%+neq%:CALLwr:GOTO
2710
 2680 PROCeqn(X%)
 2690 PROCrel(X%,npls%-2)
 2700 PROCrel(X%,npls%-1)
 2710 NEXT
 2720 ENDPROC
 2730 :
 2740 DEFFNsplit(p1%,p2%,m,n)
 2750 X(npts%)=(m*X(p2%)-n*X(p1%))/(m-n)
 2760 Y(npts%)=(m*Y(p2%)-n*Y(p1%))/(m-n)
 2770 Z(npts%)=(m*Z(p2%)-n*Z(p1%))/(m-n)
 2780 PROCtrans(npts%)
 2790 npts%=npts%+1:=npts%-1
 2800 :
 2810 DEFFNeq(P%)
 2820 LOCALn:n=cA%*X(P%)+cB%*Y(P%)+cC%*Z
(P%)+cD%:IFABSn<.1:=0ELSE=n
 2830 :
 2840 DEFFNerror
 2850 IFERR=17 PROCscreendump:=FALSE
 2860 REPORT:PRINT;" at line ";ERL:=TRUE
 2870 :
 2880 DEFPROCscreendump
 2890 REM Insert call to screendump here
 2900 ENDPROC
 2910 :
 2920 DATA Point data
 2930 DATA39
 2940 DATA100,50,40,100,50,-40,50,50,-40
,50,50,-25,-50,50,-25,-50,50,-85,-100,50
,-85,-100,50,40
 2950 DATA100,0,40,100,0,-40,50,0,-40,50
,0,-25,-10,0,-25,-50,0,-85,-100,0,-85,-1
00,0,40,-50,0,40,-10,0,40
 2960 DATA-10,30,-25,-50,30,-25,-50,30,4
0,-10,30,40,20,30,40,20,30,80,65,30,80,6
5,30,40
 2970 DATA-75,65,0,75,65,0,75,65,-40,-75
,65,-85,42,40,40
 2980 DATA20,0,80,65,0,80,65,0,40,20,0,4
0
 2990 DATA-65,20,40,-65,30,40,-85,30,40,
-85,20,40
 3000 DATA Surface data
 3010 DATA8-9-10-11.,11-12-17-34.33-8.,3
4-31-32-33.,13-14-15-16-
 3020 DATA6-7-15-14-,5-29-6-14-13-,0-1-9
-8,1-28-2-10-9,2-3-11-10
 3030 DATA13-16-20-19.,13-5-4-19.,3-4-19
-18.,18-12-11-3.
 3040 DATA12-17-21-18-,18-19-20-21-
 3050 DATA30-22-23-,30-23-24-,30-24-25-,
31-34-22-23-,32-33-25-24-
 3060 DATA7-0.30.,30.21.22-,22-34-17-21.
,0.25-30.,0.25-33-8-
 3070 DATA7.37-36.20-21.30.,16.35-36.20-
,15.38-35.16-,15.38-37.7-
 3080 DATA6-7-26-29-,4-5-29-26-,3-4-26-2
7-,7-26-27-0-,0-1-28-27-,2-3-27-28-
 3090 DATA END
```

COMPUTER CONCEPTS SPEAKS OUT



Despite its early promise, speech synthesis still seems often to be in its infancy on the Beeb. Now, with the advent of the latest Speech ROM from Computer Concepts, the infant is beginning to talk intelligibly. Alan Webster answers back.

Product : Speech ROM
Supplier : Computer Concepts, Gaddesden Place, Hemel Hempstead, Herts HP2 6EX. 0442-63933
Price : £33.35

To be quite honest, I had never been very impressed with 'computer speech', until I heard the Acorn speech PHROM, with good old Kenneth Kendall muttering his cultured tones at me. That was around two years ago now, and at that point I thought that Acorn were on to a winner. Unfortunately, nothing significant has really happened since then, other than a few 'Speech ROMs' which have claimed much, but often turned out to do little more than help Doctor Who understand the Daleks!

To use this new speech ROM on the Beeb, you need the Texas Instruments TMS5220 speech synthesis chip as well, to plug into socket IC99. The TMS5220 is NOT included in the package, so if you haven't got Acorn's original speech system (which contained the TMS5220) then you will need to buy one of these at an extra cost of about £10.00.

The speech ROM is activated by first plugging it into one of the sideways ROM sockets, and then typing the command *SPON to initialise it. Once this has been done, you must press the Break key to allow the machine to reserve two pages of memory for the speech ROM's workspace.

There are four commands to control the speech from your Beeb. The *UTTER command allows the user to specify one of 24 tones in which the speech is to be heard, and the phonemes which are to be pronounced. The *SING command must be followed by a pitch (from A-G and from one of three octaves), and the note length.

*VOICE is used to set the overall pitch of the speech, with five settings from High to Low. *SYNC allows synchronization between speech and Basic program.

The vocabulary contains 20 vowel sounds and 24 consonant sounds, which when combined can make up a whole word or even sentences. As an example, consider how to make the Beeb pronounce the phrase 'one two three'. You would have to type:

```
*UTTER<1> W +u N @ T +OO @ TH R +E
```

I found it quite easy to build up phrases, the most difficult part being in getting the intonation and stresses right. One small omission was that the package contains no phoneme editor, to allow you to create your own sounds.

Editing long sentences can prove tedious and difficult, so Computer Concepts provide, in the manual, a short utility to run in Wordwise-Plus that makes life a lot easier.

The manual supplied with the ROM is certainly comprehensive and is laid out simply and clearly. It starts off by showing how to initialise the ROM and gives a few well chosen examples to begin with. The quality of speech is quite good, but often only if you know what the processor is going to say! We tested out some of the phrases given in the manual on various members of staff at BEEBUG. Obvious phrases like 'BBC Microcomputer' were recognised, but more difficult ones like 'hot summer', which sounded as though summer had been spelt 'sunger', were often incomprehensible.

Overall, the price seems high for what can be achieved. The results are not helped either by the relatively poor quality of the Beeb's internal speaker. However, speech synthesis seems to be catching on, and this is certainly the most comprehensive and best 'toolkit' for speech available at the moment.

Virtual Arrays

Running out of space for all those arrays in your programs? Surac takes a leaf out of the mainframes' book and explains how arrays can be expanded beyond the normal memory limitations.

This month, we return to the theme of the Beeb's shortage of memory. In particular, there is not enough room for very large arrays, such as you might want in a program handling marks or scores, or in a data-base. However, if you have a disc drive, you can create and use random access files which, in some ways, you can use as slow arrays. These are called "virtual arrays".

Normally, we think of disc files as being for program storage, or as sequential files. The latter, which you can also use with a tape system, contain data written in order by a program and which another program will read back in exactly the same order. However, a disc system also has the PTR# command, specifying exactly where in a file an item of data is to be written or read. If each item of data is of a known size, then we can go straight to any particular one. For instance, if each data item occupies 20 bytes, then item 100 is 2000 bytes from the start.

VIRTUAL ARRAYS

Suppose that we want a virtual array equivalent to Basic's:

```
DIM array(200,50)
```

The first thing is to find how big a file we need to hold that much data. The array contains 201*51 (numbering starts at zero, remember) floating point (FP) numbers. Since a disc file uses 6 bytes to store an FP number, that means a total of 201*51*6, or 61506 bytes.

To create the file, we need the hex equivalent of that number - use PRINT ~61506 to calculate it. It's &F042, so:

```
*SAVE <filename> 0 +F042
```

will create a file of the correct size. At this stage, the file actually holds the contents of most of the Beeb's memory, but that will be soon overwritten.

Before you can use the virtual array, you must open the file for reading and writing using F%=OPENIN("filename") in Basic I, or F%=OPENUP("filename") in Basic II. Once the file is open, you can write to and read from it with the following routines:

Virtual Array Routines

```
10000 DEF PROCwritearray
          (fileno%,x%,y%,value)
10010 LOCAL posn%
10020 posn%=(x%*Ydim%+y%)*6
10030 PTR#fileno%=posn%
10040 PRINT#fileno%,value
10050 ENDPROC

11000 DEF FNreadarray
          (fileno%,x%,y%)
11010 LOCAL posn%,value
11020 posn%=(x%*Ydim%+y%)*6
11030 PTR#fileno%=posn%
11040 INPUT#fileno%,value
11050 =value
```

Using these routines, the equivalent of array(n1,n2)=value is:

```
PROCwritearray(F%,n1,n2,value)
```

and value=array(n1,n2) is duplicated by:

```
value=FNreadarray(F%,n1,n2)
```

Both cases assume that F% holds the channel number set by OPENIN/UP. If you've used another variable, pass that to the procedure and function.

The keys to the routines are lines 10020 and 11020, which calculate the positions of the appropriate items in the file. The PTR# statement then sets up to read or write that value. The calculations use the global variable "Ydim%", which holds the number of items along the Y-axis (second dimension) of the virtual array. Set this variable to the correct value - in the example, it is 51 (0-50).

DATABASE FILES

As an alternative, you may want to hold database information. Suppose you run a club and need, say, data about the members' names, addresses, membership numbers and scores in a club competition. It's too much to hold in RAM but you need to get to any item. This is a perfect application for a random-access file.

First, work out how big the file must be. These files only work if every record is the same size - without that, you cannot calculate where any particular record is. Let's allow 20 characters for the name, an address of three 20-character fields and an 8-character postcode, plus an integer membership number and FP score.

The DFS stores strings in their length plus 2 bytes, integers in 5 bytes and FP numbers in 6 (see page 328 of the User Guide). Each record thus has 7 fields and occupies (4*22+10+5+6), i.e. 111, bytes. If we have 200 members, we need a file of 22200 (&56B8) bytes, so create a file for our use with:

```
*SAVE D.MEMFILE 0 +56B8
```

Once that's done, OPENIN/UP the file and try the database routines. The procedures are similar to the ones for virtual arrays, but transfer data from and to global variables such as "name$" and "membno%". In practice, of course, you would use your own variables. In use, set "reclen%" to the size of each record (111 in this case). Thus, to get the 32nd record use:

```
PROCreadfile(F%,32,111)
```

Note FNpad and FNdepad. The first forces a string to a fixed size by either truncating it or adding spaces to it, while the second strips trailing spaces from a string. They ensure that all the records are the same size.

Database Routines

```
12000 DEF PROCsavefile
          (fileno%,index%,reclen%)
12010 LOCAL posn%
12020 posn%=index%*reclen%
12030 PTR#fileno%=posn%
12040 PRINT#fileno%,FNpad(name$,20),
       FNpad(add1$,20),FNpad(add2$,20),
       FNpad(add3$,20),FNpad(pcode$,8),
       membno%,score
12050 ENDPROC

13000 DEF PROCreadfile
          (fileno%,index%,reclen%)
13010 LOCAL posn%
13020 posn%=index%*reclen%
13030 PTR#fileno%=posn%
13040 INPUT#fileno%,name$,add1$,
       add2$,add3$,pcode$,
       membno%,score
13050 name$=FNdepad$(name$)
13060 add1$=FNdepad(add1$)
13070 add2$=FNdepad(add2$)
13080 add3$=FNdepad(add3$)
13090 pcode$=FNdepad(pcode$)
13100 ENDPROC

20000 DEF FNpad(str$,len%)
20010 LOCAL strlen%
20020 strlen%=LEN(str$)
20030 IF strlen%<len% THEN str$=str$
       +STRING$((len%-strlen%)," ")
       ELSE str$=LEFT$(str$,len%)
20040 =str$

21000 DEF FNdepad(str$)
21010 str$=str$+" "
21020 REPEAT
21030   str$=LEFT$(str$,LEN(str$)-1)
21040   UNTIL RIGHT$(str$,1)<>" "
21050 =str$
```

Inevitably, the routines above, and modifications of them, are very much slower than in-memory arrays, hardly surprising when you consider how hard the disc drive is working. For instance, in the first case, random virtual array elements are written at about 1 per sec and read at 2 per sec. Overall throughput, though, is much faster when the records are addressed in order.

Note, because the DFS uses a part of memory for disc transfers, some virtual array or database accesses may be satisfied from this buffer area and require no physical disc access at that time.

VIEWSTORE

Acornsoft claims that the latest release in the View family sets new standards for database packages on the Beeb. Peter Rochford has been investigating and reports with some enthusiasm.

Product : ViewStore Database Manager
Supplier : Acornsoft, Cambridge Technopark, 645 Newmarket Road, Cambridge CB35 8PD. Tel: 0223-214411
Price : £58.90

After a long wait, Acornsoft has finally released the database manager we all knew must eventually arrive to complete the View 'family'.

Like the word processor (View) and the spreadsheet (ViewSheet), ViewStore is supplied on a ROM to be plugged into one of the paged ROM sockets on the BBC micro.

It comes in the usual smart Acornsoft packaging complete with a 115 page manual, reference card, keystrip, utilities disc and fitting instructions.

ViewStore is a random access database and as such will not work with the cassette filing system. It will, however, operate with the DFS, NFS and the new ADFS. The ability to operate with the ADFS means that it can also be used with Acorn Winchester hard disc drives and should also work on an Electron with Plus 3. I should point out that ViewStore does not work very well with the old Acorn 0.9 DFS and Acornsoft advise the fitting of the latest Acorn DNFS 1.2.

Apart from working with the standard model B BBC computer and the model B+, ViewStore will work with the 6502 second processor and with shadow RAM boards such as the Aries B-20 and new B-32.

Maximum file size in ViewStore is a staggering 4096 megabytes! In reality, the maximum file size is dictated by the drive connected to the computer. With the standard Acorn DFS, ViewStore has the drawback of only being able to utilize one disc surface for a datafile. This is unlike, say, Starbase or Datagem where multiple disc surfaces may be used as one continuous file. Using the ADFS with a double sided drive, however, will allow up to 720K. A Winchester gives between 10 and 30 megabytes.

Leaving aside this drawback of file size with the DFS, the rest of the specification of ViewStore reads like no other database yet released for the BBC micro. Maximum record size is 60K in theory, but limited by the screen mode that you choose to operate in. Yes, ViewStore will operate in any of the BBC's screen modes.

The maximum number of fields allowed is 254, and the maximum field size is 239 characters. Unlike other databases I have used, ViewStore operates both with the usual card layout and a spreadsheet layout too.

In spreadsheet layout, the screen scrolls sideways field-by-field, and up-and-down from record to record using the cursor keys. The displayed width of each field can be set by the user so although the field may be up to 239 characters, you can arrange to display only the part you generally need to see. Thus more fields can be accommodated on the screen at one time. Should you need to see the rest of the characters in the field, ViewStore has the unusual facility of letting you scroll the field window back and forth.

Each field can be given a title or referred to by number. Should you name the field you are restricted to 15 characters but this is no problem as ViewStore allows you to enter a description of the contents of the field at the time you set up the database. This can be up to 79 characters long and is displayed at the top of the screen as you move from field to field.

The data in each field can be alphanumeric, textual, numeric, date or American date. Data input validation is defined by the user when setting up the database, and as well as distinguishing between text, numerical data and date, the user can specify high and low limits to be entered in a field. Furthermore, a list of

the permitted entries in the field can be specified too.

In card layout mode, the screen displays the maximum number of records that will fit. The design of the card layout is determined by the user and is effected by a system of marking and placing, using the cursor keys.

ViewStore is entered by typing *STORE which takes you to the command screen as in View and ViewSheet. The various files used in ViewStore are kept in certain directories and via the PREFIX command you can tell ViewStore which drives they are stored on.

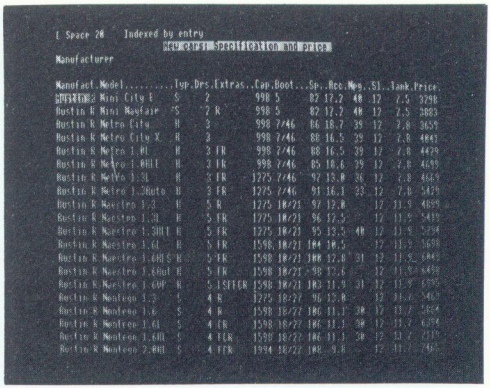
To create a new database, a utility called SETUP is used. This puts a blank data file and format file onto the disc. When loaded into ViewStore you go to the edit mode and then enter the details for the format of the database and the header. This determines the layout of the database as regards fields, indexes, key width for sorting, and data validation. After this has been done you can enter your data.

One of the benefits of having separate format and datafiles, as in ViewStore, is that you may create as many format files as you require to operate on one set of data.

Your datafiles can also be changed in ViewStore, with the CONVERT utility. This allows the size of a datafile to be increased or decreased and permits the size of each record within that file to be changed also.

The IMPORT utility is designed to convert datafiles from other databases so that they can be read into ViewStore. This is an excellent idea and in practice worked with all the files I attempted to convert from other databases provided they were in ASCII format.

Searching for records in ViewStore is done using index files. Each field can only be searched if it has its own index file created. ViewStore allows up to 9 indexes which are continually updated, and as many read-only as you require. The read-only type are updated by the INDEX utility whilst the updateable ones are done automatically as records in the



database are changed or added.

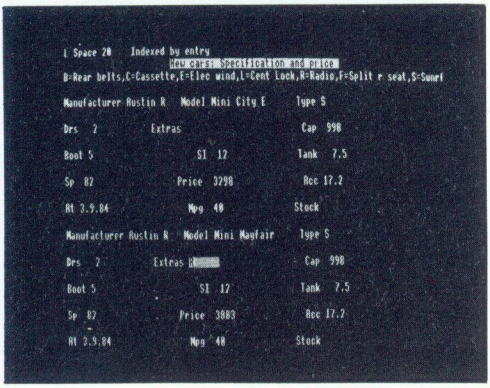
Apart from indexed searching, ViewStore has a utility called SELECT to create subsets of data. SELECT can be used on any number of fields with AND, OR and arithmetic operators also available. As with many of the other operations in ViewStore, leading and trailing wildcards may be used too.

After the records have been selected, they can be sorted, again on any number of fields and with a key length of up to 250 characters. In use, I found the sorting in ViewStore very fast indeed.

Subsets created by SELECT can be CONVERTed into datafiles if required and can also be used with the other utilities in ViewStore.

Outputting the data from ViewStore can be done to a SPOOL file or sent to a printer. The REPORT utility can give you a simple listing or one whose format is user-defined. User-defined reports can be made very complex with totalling and subtotalling of numeric fields, and calculations done using number registers. The results can be sent to a linking file and read into ViewSheet if desired. The report can be in spreadsheet or card type layout with text added, printer codes, headers and page numbering.

Other forms of output from ViewStore are via the other utilities supplied. LABEL will print labels for mailing lists etc., while MACRO produces macros for use with View and LINK will extract numeric data for reading into ViewSheet.



The manual supplied with ViewStore is very comprehensive but requires careful study. There is an example database called CARS provided on the utilities disc, and a large proportion of the manual is given over to working with this to illustrate all the features of ViewStore.

VERDICT

In the space of this review I have really only outlined the main features of what is an extremely sophisticated and powerful database, and unlike any other I have used on the BBC micro.

ViewStore's one major drawback is its ability to utilize only one disc surface for a datafile, but then I blame the Acorn DFS in this respect for not allowing configuring of more than one disc surface, and for being only single-density [An ADFS upgrade is now available for the model B - see News page].

Any other criticisms of ViewStore must be regarded as nit-picking. Within the constraints of the BBC's memory and filing system, Mark Colton has produced a remarkable piece of software and at £58.90 inc. VAT, it is excellent value for money.

ViewStore is not the simplest of database packages to understand and some time and effort will be needed to appreciate fully what it is capable of. Despite that, there is no doubt in my mind that this is now the definitive database package for the BBC micro and the one that all others will be judged against in the future.

← 9

BEEBUG FILER

```
 6220 :
20000 DEF FNask(msg$)
20020 LOCAL A:PRINT msg$;
20040 REPEAT:A=INSTR("YyNn",GET$):UNTIL
A:IF A>2 THEN PRINT"N" ELSE PRINT"Y"
20060 =A
20080 :
20200 DEF PROCheader
20220 PROCwindow1:CLS
20240 IF open% THEN PROCinv(1):PRINT STR
ING$(80," "):PRINTTAB(5,0)"File: "F$;TAB
(28,0)"Number of records: "SPC(4-LEN(STR
$(rec-1)));rec-1:PROCinv(0)
20260 PROCwindow2:ENDPROC
20280 :
20400 DEF PROCwindow1:IFw=2 X=POS:Y=VPOS
20420 VDU28,0,19,79,3:w=1:ENDPROC
20440 DEF PROCwindow2:VDU28,0,24,79,21
20460 VDU31,X,Y:w=2:ENDPROC
20480 :
20600 DEF PROCinv(on)
20620 IF on THEN COLOUR0:COLOUR129 ELSE
COLOUR1:COLOUR128
20640 ENDPROC
20660 :
30000 DEF PROCstar(p$)
30010 PROCwindow1:CLS:PROCoscli(p$):PROC
window2
30020 PRINT"Press any key to continue";
30030 G=GET:PROCheader:PRINT
30040 ENDPROC
30050 :
30060 DEF PROCoscli($os)
30070 LOCAL X%,Y%
30080 X%=os:Y%=os DIV 256:CALL &FFF7
30090 ENDPROC
30100 :
31000 DEF PROCerror
31010 IF ERR=17 THEN PROCwindow2:PRINT"C
ommand aborted":GOTO 140
31020 ON ERROR OFF
31030 PROCwindow2
31040 REPORT:PRINT" at ";ERL
31050 PROCclose:VDU26:*FX4,0
31060 ENDPROC
```

Programming sideways



Following our project on constructing an EPROM programmer, J.P. Jakubovics describes a utility for converting your machine code programs to sideways ROM or RAM.

With more and more users having sideways ROM expansion boards, many of them fitted with RAM, it is useful, as well as interesting, to write your own 'sideways' software. To help such users, and as a follow up to our recent articles on building your own EPROM programmer (BEEBUG Vol.4 Nos.4 and 5), we present two programs to help aspiring ROM programmers get started. The first program, in machine code, is a skeleton ROM program which only needs your own routines to be added. The second program can be used to test the code before 'blowing' it into an EPROM - only needed by users who have no sideways RAM.

HOW ROM SOFTWARE WORKS

There are two kinds of ROMs - language ROMs and service ROMs. Language ROMs are self-contained programs such as programming languages, wordprocessors, databases, etc. Service ROMs are collections of machine code routines that can be called, usually with a '* command', from within a language. We are only concerned here with the latter type of ROM. More information on ROMs can be found in the Advanced User Guide, page 317.

When the operating system encounters a * command, either directly typed in from the keyboard, or from a program, it passes it to each ROM in turn. Each ROM then decides whether to respond to the command or to ignore it. There are two important forms of * commands a ROM may encounter. It may be a command name after the '*', which may be followed by one or more arguments (such as, for example, the filename that follows *LOAD or *SAVE), or it may be *HELP optionally followed by a name. The ROM can distinguish between the two cases as, on entry to the ROM, the accumulator contains 4 or 9 according to whether a general or HELP command has been encountered. The memory location containing the beginning of the command is pointed to by locations &F2 and &F3 together with the Y register, so that LDA (&F2),Y can access the first character of the command, and the rest of the command may similarly be read by incrementing the Y register. It will already be apparent that the ROM has to contain quite a lot of machine code just to enable it to decode commands, before it can even begin to respond to them.

A DESCRIPTION OF PROGRAM 1

Program 1 is a universal ROM header: it contains all the code needed for the ROM to respond to commands, including *HELP, and it needs nothing more than the actual text of the help messages and the machine code routines to be appended to it - any number of these up to 26. The header enables the ROM to respond to commands consisting of a single letter after the '*', such as *A, *B, etc., with no full stop. No distinction is made between upper and lower case, and this is why the maximum number of routines this ROM program can accommodate is 26. It is unlikely that these commands will conflict with any commercial ROM software. If you have any other ROM that does respond to a single letter command, you should avoid using that letter in your own ROM. (For example the BEEBUGSOFT HELP ROM responds to *H).

```
  1 REM PROGRAM 1
  2 REM VERSION B0.1
  3 REM AUTHOR  J.P.JAKUBOVICS
  4 REM BEEBUG  NOVEMBER
  5 REM PROGRAM SUBJECT TO COPYRIGHT
  6:
 10 MODE7
 20 DIM help%(25),com%(25)
 30 assemb%=&5C00
 40 dest%=&8000
 50 os%=dest%-assemb%
 60 FOR I%=0 TO 25
 70 help%(I%)=-os%
 80 com%(I%)=-os%
 90 NEXT
100 title$="MYROM"
110 version$="1.00"
120 year$="1985"
130 author$="J.P.Jakubovics"
```

```
  140 FOR I%=0 TO 2 STEP 2
  150 P%=assemb%
  160 [OPT I%
  170 .start% BRK
  180 BRK
  190 BRK
  200 JMP service%+os%
  210 OPT FNequs(CHR$(&82)+CHR$(offpoint
%-start%)+CHR$1+title$+CHR$0+" "+version
$+" ")
  220 .offpoint% BRK
  230 OPT FNequs("(C)"+year$+" "+author$
)
  240 BRK
  250 .genhelp% OPT FNequw(&D0D)
  260 OPT FNequs("HELP available on:"+CH
R$13)
  270 .helptable%
  280 ]
  290 FOR J%=0 TO 25
  300 [OPT I%
  310 OPT FNequw((help%(J%)+os%) AND &FF
FF)
  320 ]
  330 NEXT
  340 [OPT I%
  350 .comtable%
  360 ]
  370 FOR J%=0 TO 25
  380 [OPT I%
  390 OPT FNequw((com%(J%)+os%) AND &FFF
F)
  400 ]
  410 NEXT
  420 [OPT I%
  430 .service% PHP
  440 PHA
  450 TXA
  460 PHA
  470 TYA
  480 PHA
  490 LDA &6F
  500 PHA
  510 LDA &6E
  520 PHA
  530 TSX
  540 LDY &103,X
  550 STY &6F
  560 LDA &105,X
  570 STA &6E
  580 CLD
  590 LDX #4
  600 .save6% LDA &69,X
  610 PHA
  620 DEX
  630 BNE save6%
  640 LDA &6E
  650 CMP #9
  660 BNE nothelp%
  670 JMP help%+os%
  680 .nothelp% CMP #4
  690 BEQ command%
  700 .norespexit% LDX #&FA
  710 .rest6% PLA
  720 STA &70,X
  730 INX
  740 BNE rest6%
  750 PLA
  760 TAY
  770 PLA
  780 TAX
  790 PLA
  800 PLP
  810 RTS
  820 .command% JSR stripspaces%+os%
  830 AND #&DF
  840 CMP #65
  850 BCC norespexit%
  860 CMP #91
  870 BCS norespexit%
  880 SEC
  890 SBC #65
  900 ASL A
  910 TAX
  920 LDA comtable%+os%+1,X
  930 BEQ norespexit%
  940 STA &6B
  950 LDA (&F2),Y
  960 CMP #13
  970 BNE notcr%
  980 STY &6D
  990 .comjump% STA &6C
 1000 LDA comtable%+os%,X
 1010 STA &6A
 1020 LDA &6E
 1030 PHA
 1040 LDA &6F
 1050 PHA
 1060 LDA #(pull6%+os%-1) DIV 256
 1070 PHA
 1080 LDA #(pull6%+os%-1) MOD 256
 1090 PHA
 1100 LDY &6D
 1110 LDA &6C
 1120 JMP (&6A)
 1130 .notcr% CMP #32
 1140 BEQ space%
 1150 JMP norespexit%+os%
 1160 .space% JSR stripspaces%+os%
 1170 JMP comjump%+os%
 1180 .help% JSR stripspaces%+os%
 1190 CMP #13
 1200 BNE helpname%
 1210 LDX #0
 1220 JSR &FFE7
 1230 .wrtitle% LDA start%+os%+9,X
 1240 CMP #40
 1250 BEQ endtitle%
 1260 JSR &FFE3
 1270 INX
```

```
1280 BNE wrtitle%
1290 .endtitle% JSR &FFE7
1300 JMP norespexit%+os%
1310 .helpname% AND #&DF
1320 CMP #65
1330 BCS compz%
1340 JMP norespexit%
1350 .compz% CMP #91
1360 BCC alpha%
1370 JMP norespexit%+os%
1380 .alpha% STA &6C
1390 LDA (&F2),Y
1400 CMP #32
1410 BEQ checkchar%
1420 CMP #13
1430 BNE longer%
1440 .checkchar% LDA &6C
1450 SEC
1460 SBC #65
1470 ASL A
1480 TAX
1490 LDA helptable%+os%+1,X
1500 BNE comexists%
1510 JMP norespexit%+os%
1520 .comexists% STA &6B
1530 LDA helptable%+os%,X
1540 STA &6A
1550 LDY #0
1560 JSR mode7%+os%
1570 LDA #42
1580 JSR &FFEE
1590 LDA &6C
1600 JSR &FFEE
1610 .nextchar% LDA (&6A),Y
1620 BEQ ecomhelp%
1630 JSR &FFE3
1640 INY
1650 BNE nextchar%
1660 INC &6B
1670 BNE nextchar%
1680 .ecomhelp% JSR &FFE7
1690 .respexit% LDX #&FA
1700 .restore6% PLA
1710 STA &70,X
1720 INX
1730 BNE restore6%
1740 PLA
1750 TAY
1760 PLA
1770 TAX
1780 PLA
1790 LDA #0
1800 PLP
1810 RTS
1820 .longer% LDA &6C
1830 LDX #0
1840 CMP start%+os%+9,X
1850 BEQ clonger%
1860 JMP norespexit%+os%
1870 .clonger% INX
1880 LDA (&F2),Y
1890 CMP #46
1900 BEQ fstop%
1910 CMP #32
1920 BEQ elonger%
1930 CMP #13
1940 BEQ elonger%
1950 AND #&DF
1960 CMP start%+os%+9,X
1970 BEQ nlonger%
1980 JMP norespexit%+os%
1990 .fstop% LDA start%+os%+9,X
2000 BNE wrhelp%
2010 JMP norespexit%+os%
2020 .nlonger% INY
2030 BNE clonger%
2040 .elonger% LDA start%+os%+9,X
2050 BEQ wrhelp%
2060 JMP norespexit%+os%
2070 .wrhelp% LDX #0
2080 JSR mode7%+os%
2090 .nextwr% LDA start%+os%+9,X
2100 CMP #40
2110 BEQ endname%
2120 JSR &FFEE
2130 INX
2140 BNE nextwr%
2150 .endname% JSR copychars%+os%
2160 JSR copychars%+os%
2170 LDX #0
2180 .nextcom% LDA helptable%+os%+1,X
2190 BEQ skiphelp%
2200 TXA
2210 LSR A
2220 CLC
2230 ADC #65
2240 JSR &FFEE
2250 JSR &FFE7
2260 .skiphelp% INX
2270 INX
2280 CPX #54
2290 BNE nextcom%
2300 JMP respexit%+os%
2310 .stripspaces% LDA (&F2),Y
2320 INY
2330 CMP #32
2340 BEQ stripspaces%
2350 STY &6D
2360 RTS
2370 .pull6% PLA
2380 STA &6F
2390 PLA
2400 STA &6E
2410 JMP respexit%+os%
2420 .mode7% LDA #22
2430 JSR &FFEE
2440 LDA #7
2450 JSR &FFEE
2460 RTS
2470 .copychars% LDA start%+os%+9,X
```

```
2480 BEQ endcopy%
2490 JSR &FFE3
2500 INX
2510 BNE copychars%
2520 .endcopy% INX
2530 RTS
2540 .help%(ASC("L")-65) OPT FNequs(CHR
$13+CHR$13+"Programs"+CHR$129+"f0"+CHR$1
35+"for paged listing.")
2550 BRK
2560 .jltext% OPT FNequs("K.0*FX230,1|M
|!k7|M|N|!I|M|O*FX230,0|M")
2570 OPT FNequb(13)
2580 .com%(ASC("L")-65) LDX #((jltext%+
os%) MOD 256)
2590 LDY #((jltext%+os%) DIV 256)
2600 JSR &FFF7
2610 RTS
9740 \
9750 \...Insert new routines here
9760 \
9770 .endcode%
9780 ]
9790 NEXT
9800 PRINT"Help address table starts at
&";~helptable%;"."
9810 PRINT"Routine address table starts
at &";~comtable%;"."
9820 PRINT"First free byte after m/c: &
";~endcode%;"."
9830 END
9840 :
9850 DEFFNequb(byte%)
9860 ?P%=byte%
9870 P%=P%+1
9880 =I%
9890 :
9900 DEFFNequw(word%)
9910 ?P%=word% MOD 256
9920 P%?1=word% DIV 256
9930 P%=P%+2
9940 =I%
9950 :
9960 DEFFNequs(string$)
9970 $P%=string$
9980 P%=P%+LEN(string$)
9990 =I%
```

The program as it stands has only one routine appended to it, to illustrate how this is done. The routine runs from line 2540 to 2610 with the *HELP text first at line 2540. It performs the simple task of setting function key 0 to list Basic programs in paged mode. The command used for this is *L. It will also respond correctly to the various forms of *HELP commands. Typing just *HELP (or *H.) displays the names of all ROMs in the machine, including this ROM. Its name is determined by the string assigned to the variable 'title$' in line 100, i.e. 'MYROM'. The title can be changed, but it should be made up of letters only. If you now type *HELP MYROM (or any abbreviation down to *H.M.), you will get the complete copyright message as made up from lines 100 to 130 and a list of the commands that the ROM understands, at present only 'L'. You can then type *H.L, and you will see an explanation of the command *L as given in line 2540.

USING THE PROGRAM IN SIDEWAYS RAM

Type in and load Program 1, alter line 30 to read: 30 assemb%=&8000 (enables direct assembly into sideways RAM), and save it. Run the program and press the Break key when assembly has finished. Type 'OLD' to regain your Basic program, and then you are ready to test the machine code by typing the various forms of *HELP, or *L followed by key f0. Once the program works correctly in RAM, you can change line 30 back to its original state (set assemb% to &5C00) and run the program again. Then *SAVE the machine code from address &5C00 up to address &7BFF (using '*SAVE code 5C00 7BFF'). Subsequently, this code can be *LOADed back into sideways RAM at &8000 using *LOAD code 8000.

```
   1 REM PROGRAM 2
   2 REM VERSION B0.1
   3 REM AUTHOR  J.P.JAKUBOVICS
   4 REM BEEBUG  NOVEMBER 1985
   5 REM PROGRAM SUBJECT TO COPYRIGHT
   6:
  10 DIM str% 255
  20 REPEAT
  30 INPUT LINE $str%
  40 IF MID$($str%,2,2)="H." THEN A%=9:
D%=str%+3 ELSE A%=4:D%=str%+1
  50 Y%=0
  60 ?&F2=D% MOD 256
  70 ?&F3=D% DIV 256
  80 CALL &5C03
  90 UNTIL FALSE
```

TESTING THE SIDEWAYS ROM CODE

Type in both Program 1 and Program 2 and save them to tape or disc.

Load Program 1 and change the value of dest% in line 40 from &8000 to &5C00 and then run the program. Now type PAGE=&5000, followed by NEW, and load and run Program 2. In response to the question mark, enter any command to which the ROM program

should respond, although as it's really in RAM *H. will only display the name of this ROM program. Note however that, for simplicity, Program 2 will only accept the abbreviation *H., not the full command name *HELP. If you are satisfied that the program works, press Break and then type OLD. Then alter dest% in line 40 back to &8000, run the program and *SAVE the machine code from &5C00 to &7BFF.

Whether you have paged RAM or not, you could now blow the machine code into an EPROM, though of course it will only be worth doing that when a few more routines have been added.

ADDING ROUTINES TO THE PROGRAM

New routines, including the corresponding help messages, can now be added. New instructions should be typed with line numbers 2620 onwards. To insert a help message, use the function FNequs, as on line 2540, or for Basic II users just EQUS. The end of the message should be marked with a BRK instruction. The message should not include the command name, as this is automatically printed out. The routine must terminate with an RTS instruction. In order for the help message and the routine to be recognised, they must be correctly labelled. For a routine that responds to *A, the beginning of the help message must be labelled '.help%(0)' and the entry point of the routine (i.e. the first instruction to be executed) must be labelled '.com%(0)'. If the command is to be *B, the labels should be '.help%(1)' and '.com%(1)', and so on. In fact, the help messages and routines can be in any order from line 2620 onwards, provided they are correctly labelled and terminated (with BRK and RTS respectively).

The code in Program 1 can even decode commands that have one or more arguments. When execution reaches the entry point of any routine appended to the header, the accumulator will contain the ASCII code of the first non-space character that follows the command, and the Y register together with &F2 will point to the next character after that. If the accumulator contains any character other than a carriage return (&0D), then there is at least one argument after the command. From then on, it is up to your routine to identify the arguments and respond to them as appropriate.

A NOTE ON MEMORY USAGE

The routines appended to the header may use any memory location without interfering with the working of the header. However, if you are not careful in the way your routines use memory locations, you may find that your ROM software interferes with other ROMs in the machine. You can safely use &6A-&6F in all circumstances, because these locations are protected by the header. If you need more workspace, then save its contents on the stack at the beginning of your routine, and restore it again just before exiting. Don't, for example, assume that locations &70-&8F are always safe to use. This is so only if you call your routines from Basic. If you want to call them from other languages, such as Wordwise, then &70-&8F may not be freely available.

HINTS HINTS HINTS HINTS HINTS HINTS

*FX VALUES - Eric Pope

To find the current values assigned to FX calls 166 to 255 use PRINT?(n+400) where n is the FX number concerned. E.g. at switch on, PRINT?(610) will produce the value zero as sound is enabled (*FX210,0) at switch on.

QUICK LINES - Roger Burg

Horizontal lines can be produced three times faster than the usual DRAW by clearing a graphics window of zero height. Note, however, that you must not try this with lines that protrude beyond the screen boundaries.

DEFAULT VECTOR VALUES - Geoff Smith

To find the default values of the various indirection vectors used by the Operating System, the following function can be used.

```
oldval=V%!(!&FFB7-&200)AND &FFFF
```

where V% is the address of the vector required in the range &200 to &235.

1st course

Print Formatting (part 2)

Following last month's discussion on the printing of numbers, J.Pike introduces a 'PRINT USING' command, much easier to use than Basic's own @% variable.

When printing numbers across the page or in a table, we often want to print the numbers compactly and also ensure that they cannot overflow their allotted space. Last month we looked at the formats available for printing numbers using @%. This is a special variable in BBC Basic which can be used to produce a variety of print formats. The disadvantage of all these formats is that the number can overflow its field for some values of that number. This has serious implications in a table of numbers because all the subsequent numbers along the row are pushed out of position.

We can overcome this difficulty by defining the following set of properties for a new format to supplement those available using @%.

(1) Numbers are represented using a specified number of characters.

(2) Numbers requiring less characters in their representation than the number specified have spaces added to the left of the number until (1) is satisfied.

(3) The decimal point position can be fixed and numbers are rounded if they need to be truncated.

(4) Numbers which cannot be represented with the fixed decimal point position are represented if possible with the decimal point to the right of the specified position.

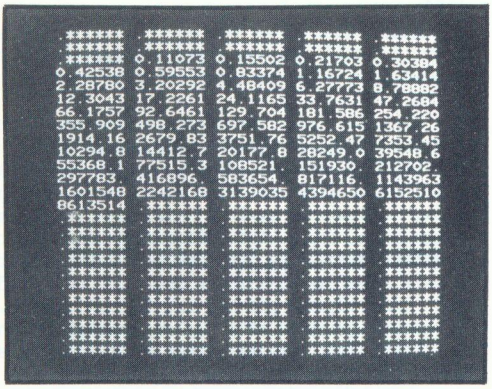
(5) Numbers which still cannot be represented cause asterisks (or some other overflow symbol) to be printed.

A further property that the format needs is that it must be easy to remember and to program. That is, the format needs to be defined either as an extension to @% or to use a simple self-evident system of specification. The system most suited to fixed length numbers is one similar to the PRINT USING instruction used in some other versions of Basic. In such a system, for example, a five character number with two decimal places is defined by using the string "**.**" or similar.

The new format is introduced into BBC Basic as the function FNUSING listed below (extra spaces have been added here to improve readability):

```
9000 DEF FNUSING(format$,number)
9010 LOCAL numb$,space$,D%,L%,@%
9020 numb$=STR$(number)
9030 IF number>0 AND ASC(format$)=45
       THEN space$=CHR$32:
            format$=MID$(format$,2)
9040 L%=LEN(format$)
9050 D%=INSTR(format$+CHR$32,".")
9060 IF D% THEN @%=(66048+L%-D%)*256
          ELSE number=INT(number+.5)
9070 IF INSTR(numb$,"E")
       THEN =space$+format$
9080 IF INSTR(numb$+".",".")>L%+1
       THEN =space$+format$
9090 format$=numb$
9100 IF LEN(format$)>L%
       THEN =space$+LEFT$(format$,L%)
9110 =space$+RIGHT$(STRING$(L%," ")
           +format$,L%)
```

This function implements the format properties given above by returning a string of length LEN(format$), which contains the number formatted as defined by format$. The number may thus be made to occupy an exact position on the page (or screen) by preceding it with a TAB



command. For example the command:

```
PRINT TAB(18) FNUSING("**.**",number)
```

will print number with characters in positions 18 to 22 and no other.

Note that by making @% a local variable (in line 9010 of the function definition), any value already assigned to @% remains unaffected when using the function. In line 9030 a useful extra feature is introduced into the format. By preceding the format string in format$ with a negative sign, the number when printed is preceded by a space for positive numbers and a negative sign for negative numbers. Without this option the maximum positive number which can be represented numerically has 10 times the magnitude of the largest negative number because the negative sign replaces the most significant digit. Line 9040 sets L% to the string length of format$. After this, the string length of FNUSING is now given by LEN(space$+format$).

The variable D% is set to one more than the number of digits in front of the decimal point. However, if format$ does not contain a decimal point (e.g. format$="***" for a 3 digit integer) then D% is set to zero. The extra space added to format$ here is to avoid any possibility of incurring the fatal INSTR bug described on page 281 of the User Guide. Line 9060 ensures the appropriate rounding of the number, either by setting @% to a fixed format or rounding integer formats directly. Lines 9070 and 9080 set FNUSING=format$ if the number cannot be represented, returning the format definition string instead of a number. The character representing a digit in format$ does not have to be *. Other characters can be used, which permits some control over the overflow display. Note, however, that the sign of the number will still be correctly indicated if the first character of format$ is given as a negative sign. Lines 9090 and 9100 set up numbers when the decimal point has to be put to the right of the specified position, leaving line 9110 to format the numbers which are in range.

A typical integer format string for format$ might be "***", which would print numbers in the range -99.5 to 999.4999 as integers, or alternatively format$="-***" which prints numbers in the range -999.5 to 999.4999 as integers. For decimal numbers format$="**.**" prints numbers in the range -9.995 to 99.9949 using 2 decimal places, -99.95 to 999.9499 using 1 decimal place, -999.5 to 9999.4999 using the decimal point as the fifth character and -9999.5 to 99999.4999 as a 5 digit integer. Numbers outside this range are printed as the string format$, that is as '**.**'.

If we set format$=" . " then all the numbers print as before except that numbers which try to overflow the allotted space are now indicated by a single point. A more novel use of this facility is given by format$="-01.E3" which gives a value to FNUSING of 1000 or -1000 to numbers outside this range. In addition, format$="-.***" where the decimal point is to the left of all the asterisks, gives numbers with the maximum accuracy within the space available (but preceded here by a single space or negative sign). Using this format definition, 8 columns of numbers can be printed across the screen (in 40 character modes). For example:

```
100 number=1/2^10
110 REPEAT
120 number=1.2*number
130 PRINT FNUSING("-.***",number);
140 UNTIL number>1.0E10
```

Using a function like FNUSING will make all your formatting much easier to follow. It also shows how defining your own function actually makes programming much easier. Why not try out FNUSING in some of your own programs?

Generating Diagrams and Drawings

Scorning the drawing board, Geoff Bains tries out two software packages to produce tidy and accurate diagrams on his Beeb.

No-one can have failed to notice the potential of computers for CAD and other drawing operations. Whereas full technical drawings are really beyond the BBC micro, this computer can excel in the production of diagrams.

For these purposes, a 'diagram' is a drawing made up of specialized symbols arranged on, and connected to, horizontal and vertical lines. Unlike a 'picture' there are no circles or arcs, excepting those contained within the symbols. The classic example is an electrical circuit diagram. However, flowcharts and architects' plans, for example, also fit into this category. Because of the constrictions, diagrams are not easily produced using 'normal' Beeb drawing packages. However, there are a few packages especially written for this purpose - BEEBUGSOFT's Design, for example. Here we look at two that prove to be a boon to anyone wanting neat and accurate diagrams.

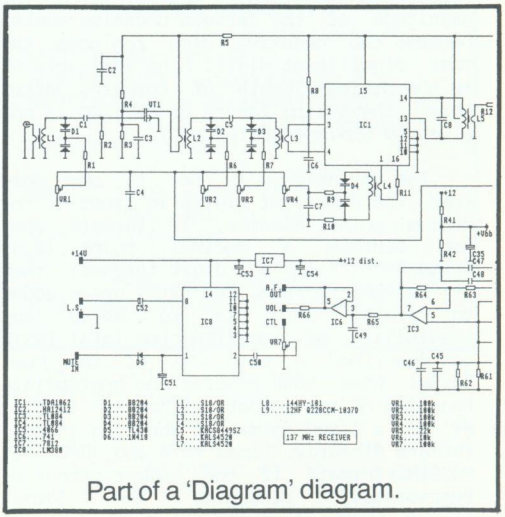
Product : Cirkwik
Supplier : Datapen Microtechnology Ltd.
Kingsclere Road, Overton,
Hampshire RG25 3JB.
0256-770488
Price : £24.95 (disc only)

This diagram package is primarily intended for use with Datapen's light pen (costing £25). However, it will also work with the Marconi trackerball (about £60). There are three main functions to the package. Firstly, there is a symbol designer to create your specialized symbols. Cirkwik limits these to 16 by 16 pixels in size but, as you can use 640 different symbols in one diagram you are not restricted from making up big symbols from several parts. However, any letters needed must also be specially defined.

Cirkwik is clearly intended for circuit diagram production and several electronics symbols are supplied on the disc. You can create your own easily enough using the symbol editor. This is very much like the numerous character definers available for the Beeb. The light pen is moved across a large 16 x 16 grid and individual pixels toggled on or off. The whole package is only concerned with two colours, so, although this editor is very simple it is reasonably effective.

The second part of the package is the diagram editor. A single diagram comprises up to eight mode 4 screens, each created and accessed separately and joined together at the printing stage. Across the top of the screen is a menu. This allows you to toggle (using the light pen) between drawing lines and placing symbols on the diagram. Lines are drawn much in the same way as they are on any Beeb drawing program, by 'rubber-banding'. However, the difference here is that the final, fixed, line is only printed horizontally or vertically. Stray lines are juggled into place by the program.

The symbols, too, are positioned onto the diagram using the menu at the top of the screen to select the desired symbol



Part of a 'Diagram' diagram.

from those you have designed earlier. Alternatively you can design as you go, as the symbol designer is available, without the loss of your diagram, at any time.

The positioning of the symbols and lines is achieved with the light pen in a rather peculiar manner. The light pen circuitry in the Beeb can only resolve the pen's position to one character width accuracy. To accurately align the symbols and lines on the diagram using the light pen, first the symbol/line is roughly aligned and the pen button pressed. Then the pen is moved in the direction that the symbol/line needs to go and the symbol is slowly dragged into position. A final press of the pen button fixes it there.

The final part of the package is the printer driver. The package is designed for Epson compatible printers but there is the facility to insert the relevant codes for a different printer.

Overall the Cirkwik package does provide a reasonably comprehensive diagram producing facility. However, the mixture of the use of lightpen and keyboard and the inherent inaccuracy of the lightpen (even with 'dragging') mean that this package is never too easy to use and positively frustrating at times.

Product : Diagram
Supplier : Pineapple Software
39 Brownlea Gardens,
Seven Kings, Ilford, Essex.
Price : £25 (disc only. Includes version for sideways RAM.)

Pineapple's Diagram is a similar package to Cirkwik but vastly more sophisticated and friendly. The diagrams are displayed and edited in mode Ø. A single diagram can contain up to 39 screens. As if this wasn't enough you can scroll around the whole diagram and edit any screenful you want. This makes lining up the various sections of the diagram much easier.

Like Cirkwik, Diagram has a symbol editor for symbols up to 32 x 24 pixels in size. There is no limit to the number of symbols allowed with each diagram, excepting that these must comprise no more than 128 character blocks (8 x 8) in total. However, the ability to transfer symbols, either individually or in groups, from one diagram to another means that this restriction does not normally worry too much. The symbol editor is again fairly standard. The cursor keys are used to move around the grid of pixels. The space bar fills a pixel and Shift empties it. A nice feature is the ability to wipe the whole symbol if wanted.

Creating and editing diagrams is also much easier with Diagram. A full mode Ø screen is used. A symbol menu covers a small section at the bottom but this can be removed if you want to draw something in this area. Symbols are placed on the screen by selecting the desired symbol from the menu, moving the cursor to the required position and pressing Return. Simple. Alphabetic characters are positioned in a similar way but the alphabetic keys on the keyboard are used to actuate them with no loss of special symbols.

Lines are drawn in a very strange but highly versatile way. When put into line mode the cursor will leave a trail as it is moved across the screen. This restricts the lines to character positions but then this is what diagrams are all about - the drawing following a grid pattern. If the space bar is pressed, as a line crosses another, then a blob is left at the junction to show, for example, wires in a circuit diagram joined and not just crossing. The lines are made up of symbols too. The first 16 symbols are ready-defined as horizontal, vertical, crossed, joined, and bent lines, with and without joining blobs. These are automatically printed as needed as the cursor is moved around the screen in line drawing mode. The advantage of having the line made up of characters is that these can be altered so that line drawing produces, say, double lines.

Diagram also has an Epson printing facility and the ability to produce further codes. Any of the 39 screens can be printed in various sizes. This is done not as a screen dump but directly from the character information stored on disc.

Diagram does take a little getting used to, as do all full packages. However, after a few practices it is a joy to use. Everything is menu driven and it almost seems to take care of itself. Full marks for an excellent product.

Quasimodo

Jonathon Temple rings the changes with this colourful eleven screen game for the BBC micro.

Quasimodo's sweetheart, Princess Esmeralda, has been imprisoned in the wicked Baron's fortress. Can you, Quasimodo the hunchback, save her from the Baron's evil clutches?

Before being re-united with your love you have to tackle eleven screens, but what with those cruel guards throwing rocks and arrows at you and those tricky swinging ropes it's going to be difficult.

You guide Quasimodo using the Z and X keys for left and right, and Return to jump. You can 'freeze' the game by pressing Ctrl. Pressing Shift will restart it again.

Quasimodo has the usual computer character's quota of three lives, one of which he will lose each time he misjudges a jump or is hit by a rock or arrow - all common occurrences when you first begin to play.

To complete a screen, Quasimodo must jump up to the bell rope and ring the bell. He is then awarded a bonus, the size of which depends on the current screen and how long it took him to complete it. After the 11th screen Quasimodo gets to meet his princess for one brief moment (Ahhh...) and then he's whisked away back to the start to try again.

Entering the program is straightforward enough - just type it in and away you go. Disc users, however, will need to miss out any unnecessary spaces and the instructions, or set PAGE to &1200 (type PAGE=&1200) before loading the program - if you choose the latter course remember not to press Break as this will corrupt the program.

PROGRAM NOTES

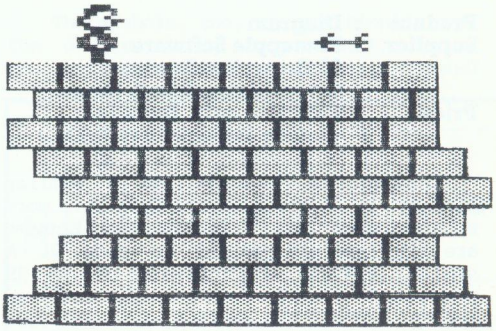
The data for the eleven screens is held in lines 2840-2890, with four numbers for each screen. The first number represents the type of screen where:

0 - is a flat wall;
1 - represents turrets;
2 - is a pit with a rope and
3 - is a pit with platforms.

The next three numbers represent the chance of an arrow, rope and boulder appearing respectively. If 0, it will not appear on that screen.

The chance is decided using RND(<number in data>). If this is equal to 1 a new arrow or boulder is made to appear. So that the arrows and boulders come at the same time and position at each go, making it easier to plan a route through each screen, the RND function is seeded using RND(-<number>). This means that the numbers produced will be the same every game.

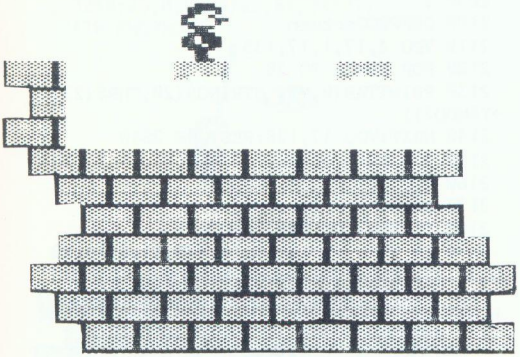
However, if Quasimodo should reach Esmeralda, when he starts again the seed (the variable RS%) for RND is changed, which means that in some of the screens the player will have to learn a new route.



Jumping is achieved by having two arrays, A%(6) and B%(6), the data for which is in lines 2820-2830. A variable, J%, is decreased as Quasimodo leaps through the air, and this is used to obtain two values from these arrays which are then added to Quasimodo's X and Y co-ordinates. By changing the values in these two lines it would be possible to get Quasimodo to jump further - useful for cheats!

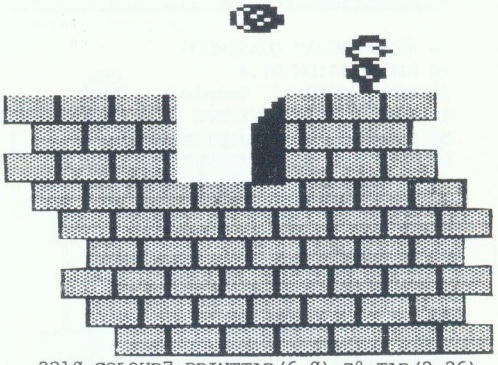
A useful procedure included in the program is PROCtriple, which when called with PROCtriple(X,Y,C,A$) will print the string A$ at tab position X,Y in colour C in triple-height characters. local variables are used for this procedure, so it can be lifted straight out and used in your own programs.

The chiming sound used when Quasimodo rings a bell is taken from Ian Waugh's excellent series of articles, 'Making Music on the Beeb' (BEEBUG Vol.3 No.8 to Vol.4 No.2).



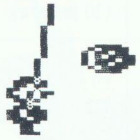
```
  10 REM PROGRAM QUASIMODO
  20 REM VERSION B1.4
  30 REM AUTHOR  J. Temple
  40 REM BEEBUG  NOVEMBER 1985
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 *TV 255
 110 ON ERROR GOTO 2930
 120 MODE 1
 130 PROCinst
 140 PROCchars
 150 PROCinit
 160 REPEAT
 170 L%=3:S%=0:K%=1:RS%=3
 180 REPEAT
 190 MODE 2
 200 PROCscreen
 210 REPEAT
 220 PROCman
 230 IF F% PROCrope
 240 IF P% PROCarrow
 250 IF M% PROCboulder
 260 UNTIL E%
 270 IF E%=1 L%=L%-1:SOUND 0,1,50,1 ELS
E PROCbonus
 280 UNTIL L%=0
 290 IF L%=0 PROCkilled
 300 UNTIL FALSE
 310 :
1000 DEFPROCman
1010 Z%=Z%-5:VDU 4,31,6,0
1020 IF Z%>-1 PRINT ;Z%;"   ";
1030 VDU 5:A%=X%:B%=Y%:C%=V%:D%=W%
1040 IF INKEY-2 REPEAT UNTIL INKEY-1
1050 IF INKEY-74 IF J%+JF%=0 J%=6:N%=(I
NKEY-98)-(INKEY-67):SOUND 1,1,10,5
1060 JF%=0:IF J% PROCjump:ENDPROC
1070 IF F% IF X%=G%-32 GOTO 1110
1080 IF INKEY-98 IFX%>0 X%=X%-32:W%=W%
EOR 3:SOUND 18,-10,50,1:IF V%<>231 V%=23
1:W%=233
1090 IF INKEY-67 IFX%<1216 X%=X%+32:W%=
W% EOR 7:SOUND 18,-10,50,1:IF V%<>232 V%
=232:W%=235
1100 IF POINT(X%,Y%-64)=0 IF POINT(X%+5
6,Y%-64)=0 E%=1:F%=0
1110 IF D%<>W% GCOL3,6:MOVE A%,B%:VDU C
%,10,8,D%:MOVE X%,Y%:VDU V%,10,8,W%
1120 IF D%=W% FOR N=1 TO 30:NEXT
1130 ENDPROC
1140 :
1150 DEFPROCjump
1160 X%=X%+A%(J%)*N%:Y%=Y%+B%(J%)
1170 IF X%=0 IF N%=-1 N%=0
1180 IF X%=1216 IF N%=1 N%=0
1190 J%=J%-1:GCOL 3,6:MOVE A%,B%
1200 VDU C%,10,8,D%:MOVE X%,Y%
1210 VDU V%,10,8,W%:IF J%=0 JF%=1
1220 IF X%=1216 IF Y%=668 E%=2
1230 ENDPROC
1240 :
1250 DEFPROCrope
1260 GCOL 3,7:MOVE 640,896:DRAW G%,604
1270 G%=G%+H%:IF G%=320 OR G%=960 H%=-H
%
1280 GCOL 3,7:MOVE 640,896:DRAW G%,604
1290 IF J%=0 IF ABS(G%-X%)<65 IF ABS(60
4-Y%)<65 GCOL3,6:MOVE X%,Y%:VDU V%,10,8,
W%:X%=G%-32:MOVE X%,Y%:VDU V%,10,8,W%
1300 ENDPROC
1310 :
1320 DEFPROCarrow
1330 IF R%=0 GOTO 1400
1340 IF ABS(Q%-X%)<33 IF ABS(604-Y%)<33
E%=1
1350 GCOL 3,3:MOVE Q%,604:VDU 226
1360 Q%=Q%-32:MOVE Q%,604:VDU 226
1370 IF Q%<-32 R%=0
1380 IF ABS(Q%-X%)<33 IF ABS(604-Y%)<33
E%=1
1390 ENDPROC
1400 IF RND(P%)=1 Q%=1216:R%=1:GCOL 3,3
:MOVE 1216,604:VDU 226
1410 ENDPROC
1420 :
1430 DEFPROCboulder
```

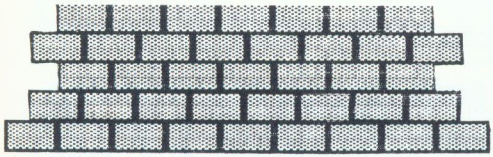
```
1440 IF U%=0 GOTO 1510
1450 IF ABS(T%-X%)<33 IF ABS(672-Y%)<33
E%=1
1460 GCOL 3,6:MOVE T%,672:VDU 227
1470 T%=T%+64:MOVE T%,672:VDU 227
1480 IF T%>1216 U%=0
1490 IF ABS(T%-X%)<33 IF ABS(672-Y%)<33
E%=1
1500 ENDPROC
1510 IF RND(M%)=1 T%=0:U%=1:GCOL 3,6:MO
VE 0,672:VDU 227
1520 ENDPROC
1530 :
1540 DEFPROCkilled
1550 VDU 4,28,2,26,17,20,12
1560 PROCtriple(3,1,2,"GAME OVER")
1570 PRINTTAB(1,5)"Press Spacebar"
1580 REPEAT UNTIL GET=32
1590 ENDPROC
1600 :
1610 DEFPROCbonus
1620 SOUND 2,2,81,16:SOUND 2,2,81,16
1630 FOR N=1 TO 300:NEXT:IF Z%<0 Z%=0
1640 VDU 4,28,2,26,17,20,12
1650 K%=K%+1:S%=S%+Z%
1660 IF K%=12 PROCcongrats:ENDPROC
1670 PROCtriple(2,1,3,"BONUS = "+STR$(Z
%))
1680 TIME=0:REPEAT UNTIL TIME>200
1690 ENDPROC
1700 :
1710 DEFPROCcongrats
1720 K%=1:VDU 26,12:IF RS%=3 L%=L%+1
1730 PROCscreen
1740 VDU 4,28,2,26,17,16,12
1750 PROCtriple(3,1,3,"WELL DONE !")
1760 VDU 26,5,18,3,5,25,4,960;636;231,1
0,8,230,18,3,6
1770 FOR X%=0 TO 864 STEP 16
1780 MOVE X%,636:VDU V%,10,8,W%
1790 W%=W% EOR 7:MOVE X%+16,636
1800 VDU V%,10,8,W%:FOR N=1 TO 40
1810 NEXT,:PLOT 69,928,616
1820 RESTORE 2900:N=81:*FX 15,0
1830 FOR T=1 TO 10:READ A,D:N=N+A
1840 SOUND 1,-15,N,D:SOUND 2,-10,N+48,D
1850 NEXT:TIME=0
1860 REPEAT UNTIL TIME>200:VDU 4
1870 PROCtriple(4,21,2,"Now try again")
1880 COLOUR 5:PRINTTAB(7,25)"<SPACE>"
1890 REPEAT UNTIL GET=32:RS%=RS%+1
1900 ENDPROC
1910 :
1920 DEFPROCtriple(X,Y,C,A$)
1930 LOCAL A%,N%,X%,Y%
1940 X%=&70:Y%=0:A%=10:COLOUR C
1950 FOR N%=1 TO LEN(A$)
1960 ?&70=ASC(MID$(A$,N%)):CALL &FFF1
1970 VDU23,253,?&71,?&71,?&71,?&72,?&72
,?&72,?&73,?&73
1980 VDU23,254,?&73,?&74,?&74,?&74,?&75
,?&75,?&75,?&76
1990 VDU23,255,?&76,?&76,?&77,?&77,?&77
,?&78,?&78,?&78
2000 VDU 31,X+N%-1,Y,253,10,8,254,10,8,
255
2010 NEXT
2020 ENDPROC
2030 :
2040 DEFPROCscore(N%)
2050 S%=S%+N%:VDU 4,17,7,31,6,1
2060 PRINT LEFT$("00000",5-LEN(STR$(S%)
))+STR$(S%)
2070 VDU 5
2080 ENDPROC
2090 :
2100 DEFPROCscreen
2110 VDU 4,17,1,17,135
2120 FOR Y%=14 TO 28
2130 PRINTTAB(0,Y%) STRING$(20,CHR$(237
+Y%MOD2))
2140 NEXT:VDU 17,128:RESTORE 2840
2150 FOR N%=1 TO K%
2160 READ V%,P%,F%,M%:NEXT
2170 IF V%=1 PROCturrets
2180 IF V%=2 OR V%=3 PROCpit
2190 IF V%=3 VDU 31,5,14,225,31,8,14,22
5,31,11,14,225,31,14,14,225
2200 Z%=K%*100+400:PRINTTAB(0,0)"BONUS:
";TAB(0,1)"SCORE:"
```



```
2210 COLOUR7:PRINTTAB(6,0);Z%;TAB(2,26)
"SCREEN:";K%
2220 COLOUR6:IFL%>1 FOR X%=15 TO L%*2+1
1 STEP 2:VDU 31,X%,0,232,10,8,235:NEXT
2230 PROCscore(0):D=RND(-RS%)
2240 X%=0:Y%=636:V%=232:W%=235
2250 E%=FALSE:J%=FALSE:JF%=FALSE
2260 R%=FALSE:U%=FALSE:G%=320:H%=32
2270 VDU 18,0,6,25,4,1248;636;25,5,1248
;888;18,3,3,25,4,1216;928;228,10,8,229,1
8,3,6,25,4,X%;Y%;V%,10,8,W%,23;10,32;0;0
;0;
2280 IF F% GCOL3,7:MOVE 640,896:DRAW G%
,604
```

```
 2290 ENDPROC
 2300 :
 2310 DEFPROCturrets
 2320 PRINTTAB(4,14)G$;TAB(9,14)G$;TAB(1
4,14)G$
 2330 FOR X%=376 TO 1016 STEP 320
 2340 VDU 25,4,X%;568;25,0,0;-92;25,81,-
32;0;25,0,0;60;25,81,32;32;
 2350 NEXT
 2360 ENDPROC
 2370 :
 2380 DEFPROCpit
 2390 FOR X%=3 TO 15 STEP 2
 2400 PRINTTAB(X%,14)G$;:NEXT
 2410 VDU 25,4,1080;568;25,0,0;-92;25,81
,-32;0;25,0,0;60;25,81,32;32;
 2420 ENDPROC
```





```
 2430 :
 2440 DEFPROCinit
 2450 DIM A%(6),B%(6)
 2460 FOR N%=1 TO 6
 2470 READ A%(N%),B%(N%):NEXT:RS%=3
 2480 G$=STRING$(3,"  "+CHR$10+CHR$8+CHR
$8)
 2490 ENDPROC
 2500 :
 2510 DEFPROCinst
 2520 VDU17,130,28,10,5,28,1,12,26
 2530 PROCtriple(11,2,1,"Q U A S I M O D
O")
 2540 VDU19,3,6;0;17,128,17,3,31,0,8
 2550 PRINT"  In this version of the wel
l-known""arcade game you must guide Qua
simodo""through the eleven screens to h
is""sweetheart, Princess Esmeralda."'
 2560 PRINT"  Our hero must avoid the ar
rows and""rocks the cruel guards are th
rowing at""him, and use the ropes to sw
ing across""the dangerous gaps."'
 2570 PRINT"  You should use the Z and X
 keys to""move left and right, and <Ret
urn> to""jump. To complete each screen
Quasimodo""must jump up to the bell rop
e and ring""the bell."'
 2580 PRINT"  <Ctrl> can be used to free
ze the game""until <Shift> is pressed."
'
 2590 COLOUR2:PRINT'TAB(5)"Press the SPA
CE BAR to play..."
 2600 REPEAT UNTIL GET=32
 2610 ENDPROC
 2620 :
 2630 DEFPROCchars
 2640 VDU23,225,-1,-1,-1,239,193,0,0,0
 2650 VDU23,226,33,66,-1,66,33,0,0,0
 2660 VDU23,227,60,94,182,175,187,183,94
,60
 2670 VDU23,228,24,36,24,44,44,44,94,94
 2680 VDU23,229,94,-1,129,126,0,0,0,0
 2690 VDU23,230,60,172,92,24,56,60,62,12
6
 2700 VDU23,231,56,124,76,38,194,70,60,2
4
 2710 VDU23,232,28,62,50,100,67,98,60,24
 2720 VDU23,233,60,118,118,110,60,24,24,
56
 2730 VDU23,234,60,110,118,118,60,24,60,
100
 2740 VDU23,235,60,110,110,118,60,24,24,
28
 2750 VDU23,236,60,110,118,118,60,24,60,
38
 2760 VDU23,237,-3,-3,-3,-3,-3,-3,-3,0
 2770 VDU23,238,223,223,223,223,223,223,
223,0
 2780 ENVELOPE 1,133,8,4,8,3,1,1,126,0,0
,-10,126,0
 2790 ENVELOPE 2,4,0,0,0,0,0,0,126,-1,-1
,-1,80,0
 2800 ENDPROC
 2810 :
 2820 DATA 0,-16,32,-16,32,-16
 2830 DATA 32,16,32,16,0,16
 2840 DATA 0,2,0,0,1,0,0,0
 2850 DATA 2,0,1,0,3,0,0,0
 2860 DATA 0,2,0,4,1,2,0,0
 2870 DATA 2,99,1,0,3,0,0,2
 2880 DATA 1,5,0,2,2,99,1,2
 2890 DATA 3,10,0,2
 2900 DATA 0,4,8,4,8,4,4,4,8,8,-12,8
 2910 DATA 4,8,-12,8,8,8,-16,8
 2920 :
 2930 MODE7:PRINT'':REPORT
 2940 PRINT " at line ";ERL
 2950 END
```

BEEBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams
Assistant Editor: Geoff Bains
Production Editor: Phyllida Vanstone
Assistant Production Editor: Yolanda Turuelo
Technical Assistant: Alan Webster
Secretary: Debbie Sinfield
Managing Editor: Lee Calcraft
Additional thanks are due to Sheridan Williams, Adrian Calcraft, John Yale and Tim Powys-Lybbe.

All rights reserved. No part of this publication may be reproduced without prior written permission of the Publisher. The Publisher cannot accept any responsibility, whatsoever for errors in articles, programs, or advertisements published. The opinions expressed on the pages of this journal are those of the authors and do not necessarily represent those of the Publisher, BEEBUG Publications Limited.
BEEBUG Publications Ltd (c) 1985

Editorial Address

BEEBUG
PO BOX 50,
Holywell Hill,
St. Albans AL1 3YS

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 £40 per page, but please give us warning of anything substantial that you intend to write. A leaflet, 'Notes of Guidance for Contributors' is available on receipt of an A5 (or larger) SAE.

In the case of material longer than a page, we would prefer this to be submitted on cassette or disc in machine readable form using "Wordwise", "View", or other means, but please ensure an adequate written description of your contribution is also included. If you use cassette, please include a backup copy at 300 baud.

HINTS

There are prizes of £5 and £10 for the best hints each month, plus one of £15 for a hint or tip deemed to be exceptionally good.

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

SUBSCRIPTIONS

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

MEMBERSHIP SUBSCRIPTION RATES

£ 6.40 6 months (5 issues) UK ONLY
£11.90 UK - 1 year (10 issues)
£18 Europe, £21 Middle East
£23 Americas & Africa, £25 Elsewhere

BACK ISSUES
(Members only)

Vol	Single issues	Volume sets (10 issues)
1	90p	£8
2	£1	£9
3	£1.20	£11
4	£1.20	–

Please add the cost of post and packing as shown:

DESTINATION	First issue	Each subsequent issue
UK	30p	10p
Europe	70p	20p
Elsewhere	£1.50	50p

All overseas items are sent airmail (please send a sterling cheque). We will accept official UK orders but please note that there will be a £1 handling charge for orders under £10 that require an invoice. Note that there is no VAT on magazines.

Back issues are for members only, so it is ESSENTIAL to quote your membership number with your order. Please note that the BEEBUG Reference Card and BEEBUG supplements are not supplied with back issues.

Subscriptions, Back Issues & Software Address

BEEBUG
PO BOX 109
St. Johns Road
High Wycombe HP10 8NP

Hotline for queries and software orders

St. Albans (0727) 40303
Manned Mon-Fri 9am-4.30pm

24hr Answerphone Service for Access and Barclaycard orders, and subscriptions
Penn (049481) 6666

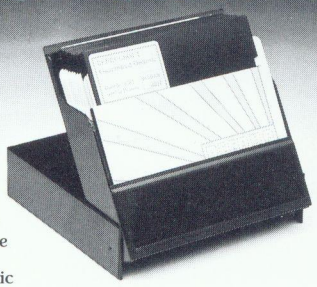
If you require members' discount on software it is essential to quote your membership number and claim the discount when ordering.

DYNAMIC DISCS

TESTED BY BEEBUG

BEEBUG, the largest independent computer user group in the UK, offer 100% tested discs supplied by one of Britain's leading disc manufacturers.

10 FREE LIBRARY CASE



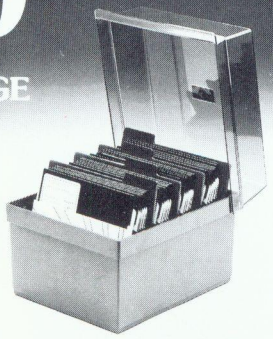
Orders for 10 discs are sent in black plastic library cases.

25 FREE STORAGE BOX



Orders for 25 are delivered in strong plastic Storage box with 4 dividers.

50 FREE STORAGE BOX



Orders for 50 are delivered in strong plastic Storage box with 4 dividers.

COMPARE PRICES

4 Types of Disc To Meet Your Exact Requirement

	48 TPI	DOUBLE DENSITY			
10	S/S D/D	£14.90	10	D/S D/D	£20.50
25	S/S D/D	£34.90	25	D/S D/D	£46.20
50	S/S D/D	£59.30	50	D/S D/D	£82.40

	96 TPI	DOUBLE DENSITY			
10	S/S D/D	£20.50	10	D/S D/D	£21.90
25	S/S D/D	£46.20	25	D/S D/D	£49.90
50	S/S D/D	£82.40	50	D/S D/D	£93.50

All prices include Storage Box, VAT and delivery to your door (UK)

Suitable for BBC Micro and all other computers using 5¼ inch discs including Atari and Commodore.

Fully Guaranteed – Not only by Beebug but by one of the UK's top disc manufacturers.

We regret that we have had to pass on a slight increase in the price of our discs, but we are now able to offer a wider range to meet your exact requirements.

These discs are the best. Please use the enclosed order form and order from our usual address.
BEEBUG PO BOX 109
St. Johns Road
High Wycombe HP10 8NP.

Official orders are welcome.

Barclaycard and Access telephone 0494 81 6666
Further information telephone 0727 60263

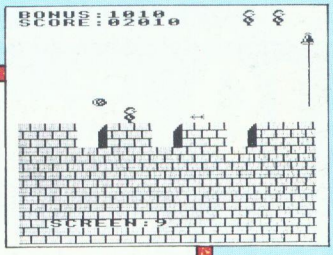
BEEBUG SOFT

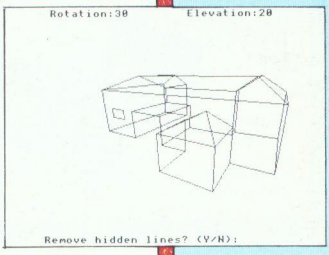
Magazine Cassette/Disc

NOVEMBER 1985 CASSETTE DISC CONTENTS

RECURSIVE TREES – displays for the armchair gardener
DYNAMIC MEMORY WINDOW – an eye into your Beeb as it runs your programs
WORDWISE PLUS EXAMPLES – segment programs from the series
LOAN REPAYMENT – calculate how much you owe
EPROM PROGRAMMER DRIVER – blow your own
WORKSHOP PROCEDURES – text compression
DISC MENU EXTENSIONS – the complete disc menu program with all the extensions
DATA STRUCTURES – linked lists and binary trees
ROULETTE – spin the wheel and break the bank
EXTRA FEATURES THIS MONTH
DISC BENCHMARKS – test out your drives with the program used for the DDFS review
3D GRAPHICS SYSTEM DEMO – a fast moving display from Glentop's graphics package

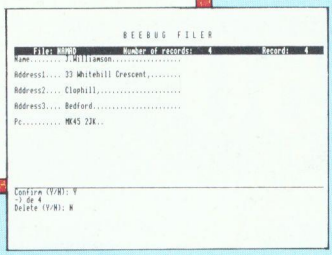
Quasimodo





Hidden Line Removal

BEEBUG Filer



All this for £3.00 (cass) £4.75 (disc) +50p p&p.
Back issues (disc since Vol. 3 No. 1, cass since Vol. 1 No. 10) available at the same prices.

Subscription rates	DISC UK	CASS UK	DISC O'seas	CASS O'seas
6 months (5 issues)	£25	£17	£30	£20
12 months (10 issues)	£50	£33	£56	£39

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

Cassette subscriptions can be commuted to disc subscription on receipt of £1.70 per issue of the subscription left to run.

All subscription and individual orders to
BEEBUG, PO BOX 109, St. Johns Road, High Wycombe HP10 8NP

Printed in England. DCL 11532