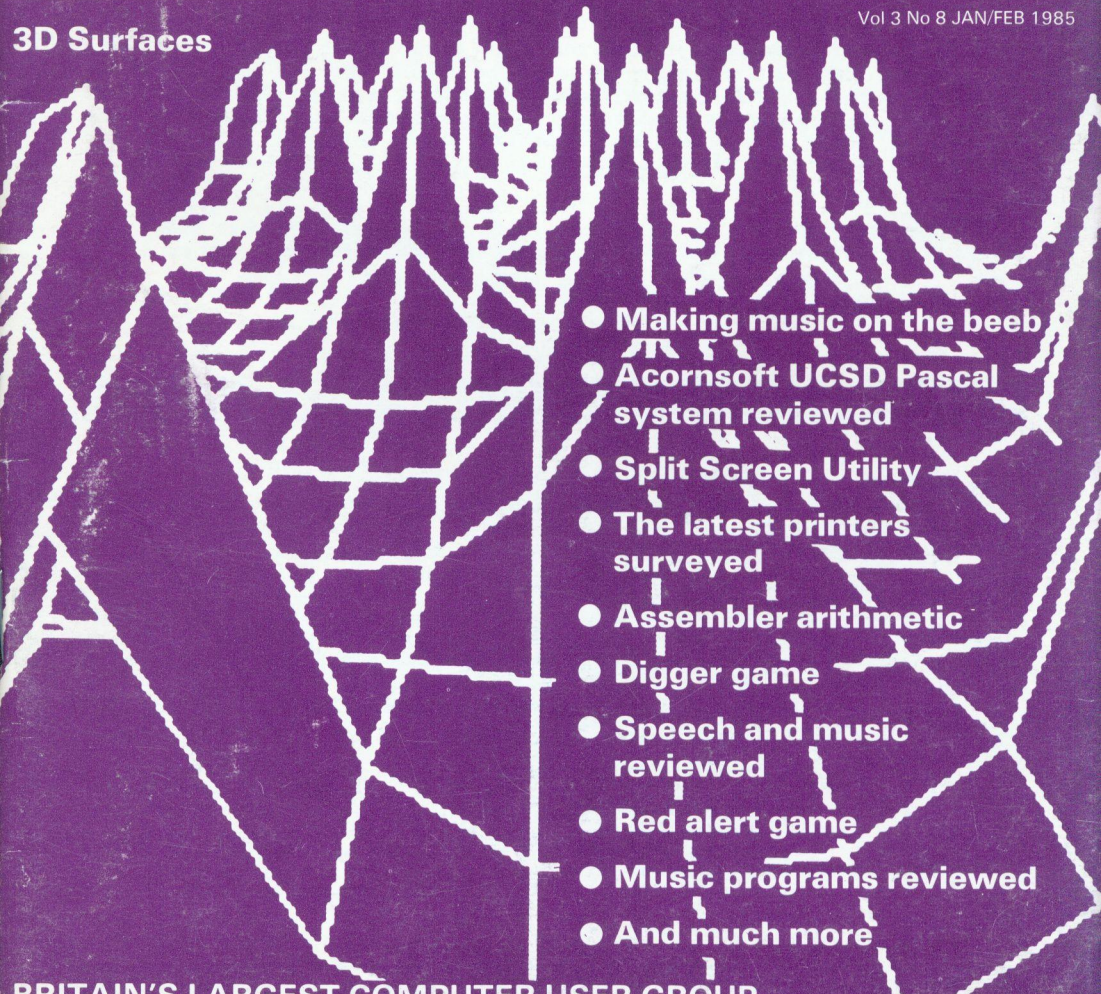
£1.20

BEEBUG

FOR THE BBC MICRO

Vol 3 No 8 JAN/FEB 1985

3D Surfaces



- Making music on the beeb
- Acornsoft UCSD Pascal system reviewed
- Split Screen Utility
- The latest printers surveyed
- Assembler arithmetic
- Digger game
- Speech and music reviewed
- Red alert game
- Music programs reviewed
- And much more

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

EDITORIAL

THIS MONTH'S MAGAZINE

We are featuring this month a program which draws the most interesting and colourful 3D surfaces - some of the results can be seen on this month's cover. This is such a fascinating and attractive program that we are running a competition for all BEEBUG members in connection with this program. We shall be awarding a prize of £50 to the most interesting surface design that anyone can achieve using this program - full details are contained within the article itself.

Talking of competitions, the advertising supplement contains this month the results of the 'Sum-Squares' Brainteaser competition set in BEEBUG Vol.3 No.5. The fortunate prize winners are Bill Wilkinson and Michael Catty. We expect to include another of these popular competitions in the next supplement. For those who are interested, we hope to include the winning program on the magazine cassette/disc for the next issue (when more space will be available).

We have also included several items this month with a speech or musical flavour, including the first part of a major new series on 'Making Music on the Beeb' by Ian Waugh, the author of one of the best known books on the subject which was reviewed last month. We have also included reviews of speech systems and of commercial music programs. We would also draw your attention to Murom, the new ROM based music program that is now available from BEEBUGSOFT.

NEXT ISSUE

Remember that this is a two month issue (January/February). The next BEEBUG will be the March edition which should be with you by the end of February. See you all in 1985.

Mike Williams

NOTICE BOARD NOTICE BOARD NOTICE BOARD NOTICE BO

HINT WINNERS

We have decided to award our first £15 prize to A.E.Wilmhurst for his hint on using the Aries B-20 board. In addition, we have awarded our monthly £5 prize to D.Morgan. New hints and tips for BEEBUG are always welcome.

MAGAZINE CASSETTE/DISC

This month'S magazine cassette/disc contains an extra item, a utility called Crunch, written by John Marriage. This most useful program will squeeze already compacted programs even further by joining together as many lines as possible so saving even more memory space. This was originally intended to be included with the Pack program in the November issue, but has been postponed until now through lack of space on the cassette/disc.

BEEBUG MAGAZINE

GENERAL CONTENTS

2 Editorial
4 3D Surfaces
6 Acorn News
7 Acornsoft's UCSD Pascal System Reviewed
10 Points Arising
11 Beginners Start Here
Introducing Machine Code (Part 1)
13 Adventure Games
15 Making Music on the Beeb (Part 1)
20 Addendum to Acornsoft P-System Review
21 The Latest Printers Surveyed
24 Disabling Break
26 Computer Games for the Blind
27 Tube Compatibility of ROM Software
28 Assembler Arithmetic the Easy Way
32 Three Speech Systems Reviewed
34 A Split Screen Utility
37 More News
38 Three Music Programs Reviewed
40 BEEBUG Workshop
Using Indirection Operators (Part 2)
42 Quicksilva's Drumkit Reviewed
43 Digger
47 Red Alert

PROGRAMS

4 3D Surfaces
11 Machine Code Examples
15 Making Music
24 Disabling Break
28 Assembler Arithmetic Routines
34 Split Screen Utility
40 Four Workshop Examples
43 Digger Game
47 Red Alert Game

HINTS, TIPS & INFO

14 Setting the Aries B-20 Board on Break
14 View Control Codes
14 Another Oddity in Basic
20 Which Day is it?
25 Double Usage
25 Quick Wait for Key
39 Tube Core Save
46 Local Parameters

3D SURFACES

by Q.A. Rice

Most users of the BBC micro will never tire of the fascinating graphics displays that this machine is capable of. This program is an excellent example of the old idea of representations of distorted three dimensional surfaces on your computer's screen.

This program will draw out representations of three dimensional surfaces on the screen of your BBC micro. The surfaces follow one of a set of equations. A choice of eight equations is included in the program and there is room for your own as well. Because of the way that the surfaces are drawn (from the back forwards) a simple form of 'hidden line removal' is achieved. This makes the surfaces very realistic.

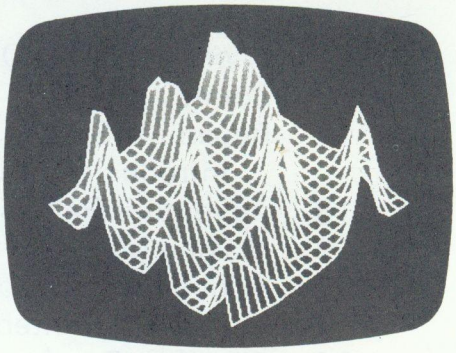
In addition the display can be drawn with perspective or not, as you choose, and in an inverted form if you so wish. The whole program is menu driven and is both simple to use and a pleasure to watch.

Type in the program carefully and then save it before you run it. Disc users will find that this program leaves them with a 'No room' message: Either use the movedown routine from the October BEEBUG, or cut out all the first few lines of REM statements, in the program and set PAGE to &1200, before you load it in.

The program gives you a choice of eight different equations for the surfaces. The ninth option is for your own equation. This should be entered in line 1860 with Z as a function of X% and Y%. The equation included in the program listing at this line just draws a flat surface. Try variations of the other equations given in lines 1780 to 1850 first, to get the hang of this.

PROGRAM NOTES

The procedure, PROCchoices, inputs from the user the equation number and the choice of whether the display is to be in perspective and whether it is to be inverted or not. PROCcalc then calls one of the two procedures, PROCpers or PROCiso, depending on which option was chosen.



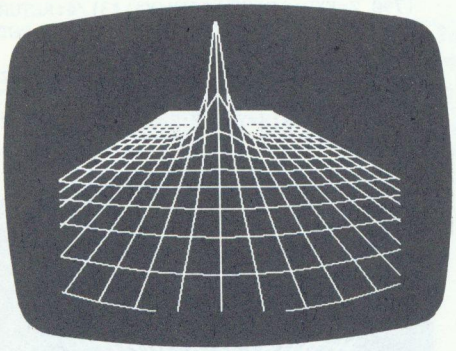
The procedure chosen, first calculates all the points to be plotted, using the equation chosen (PROCfnz) and then plots the series of squares onto the screen (PROCdraw).

There are no hard and fast rules as to making up your own equations to enter as the ninth option. Just try whatever seems best and you will be pleasantly surprised by the elegant and vivid results.

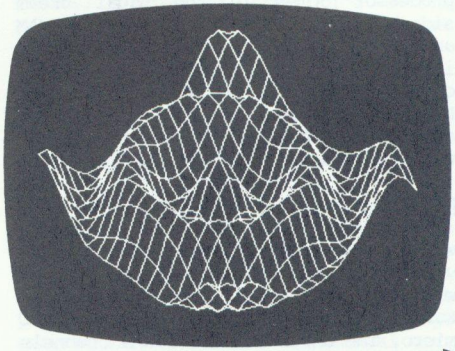
When you have experimented with this program for a while, you might like to try it with a more competitive flavour. We are offering £50 for the equation to fit into this program at line 1860, that produces the most pleasing display. Send in your ideas (only one per person please) to the editorial address. Clearly mark your envelope 'Surface Competition'. The closing date for entries is 8th February 1985 so get your ideas in soon. The judges' decision, as they say, will be final.

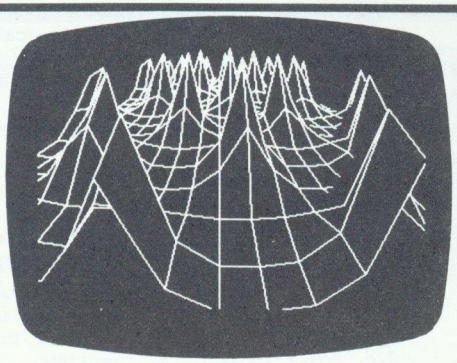
```
10 REM PROGRAM 3D SURFACE
20 REM VERSION B2.0
30 REM AUTHOR  Q.A.RICE
40 REM BEEBUG  JAN/FEB 1985
```

→

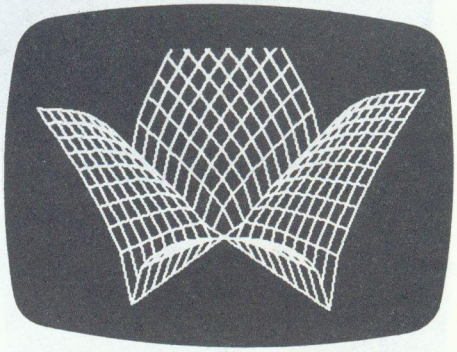


```
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO 1880
 110 :
 120 DIMP%(20,20,2)
 130 REPEAT
 140 MODE 7
 150 PROCchoices
 160 MODE 1
 170 PROCcalc
 180 UNTIL FALSE
 190 END
 200 :
1000 DEF PROCchoices
1010 PRINT TAB(9,1)"HIDDEN LINE GRAPHS
"TAB(9,2)"BLACKOUT TECHNIQUE"
1020 PRINT TAB(10,23)"ENTER CHOICE"
1030 VDU 28,8,22,39,5
1040 PRINT "1.   SQR(X)*SQR(Y)"'"2.
 COS(X)*COS(Y) #1"'
1050 PRINT "3.   COS(X)*COS(Y) #2"'"4
.   COS(X)*COS(Y)*DISTANCE"'
1060 PRINT "5.   COS(D)*COS(D)"'"6.
 EXP(DISTANCE)"'
1070 PRINT "7.   COS(CORNER DISTANCE)"
'"8.   EXP(COS(CORNER DISTANCE))"'
1080 PRINT "9.   YOUR OWN PREDEFINED"
1090 REPEAT C=(GET-48):UNTIL C>0 AND C
<10
1100 CLS
1110 PRINT "1.   ISOMETRIC"'"2.   PE
RSPECTIVE"
1120 REPEAT PM=(GET-48):UNTIL PM=1 OR
PM=2
1130 CLS
1140 PRINT "1.   NORMAL"'"2.   INVER
TED"
1150 REPEAT P=(GET-48):UNTIL P=1 OR P=2
1160 ENDPROC
1170 :
1180 DEF PROCcalc
1190 VDU23,1,0;0;0;0;
1200 COLOUR 2
1210 PRINT TAB(0,8)"Please wait - calcu
lating 400 points:"
1220 IF PM=2 THEN PROCpers ELSE PROCiso
1230 REPEAT UNTIL GET=32
1240 ENDPROC
1250 :
1260 DEF PROCpers
1270 FOR X%=1 TO 20
1280 FOR Y%=1 TO 20
1290 PROCfnz:PRINTTAB(37,8);20*X%-20+Y%
1300 S=X%-10:Z=Z-5:D=SQR(SQR(Z*Z+S*S)+Y
%*Y%)
1310 P%(X%,Y%,1)=(S/D)*400+600
1320 P%(X%,Y%,2)=(Z/D)*400+800
1330 NEXT Y%,X%
1340 CLS
1350 FOR X%=1 TO 9
1360 FOR Y%=20 TO 2 STEP-1
1370 PROCdraw
1380 NEXT Y%,X%
1390 FOR X%=19 TO 10 STEP-1
1400 FOR Y%=20 TO 2 STEP-1
1410 PROCdraw
1420 NEXT Y%,X%
1430 ENDPROC
1440 :
1450 DEF PROCiso
1460 FOR X%=1 TO 20
1470 FOR Y%=1 TO 20
1480 PROCfnz:PRINTTAB(37,8);20*X%-20+Y%
1490 P%(X%,Y%,1)=(X%/2+Y%/2)*60
1500 P%(X%,Y%,2)=(Z+X%/2-Y%/2)*40+500
1510 NEXT Y%,X%
1520 CLS
1530 FOR X%=19 TO 1 STEP-1
1540 FOR Y%=2 TO 20
1550 PROCdraw
1560 NEXT Y%,X%
1570 ENDPROC
1580 :
1590 DEFPROCdraw
1600 GCOL0,1
1610 MOVE P%(X%,Y%,1),P%(X%,Y%,2)
1620 MOVE P%(X%,Y%-1,1),P%(X%,Y%-1,2)
```





```
1630 PLOT85,P%(X%+1,Y%-1,1),P%(X%+1,Y%
-1,2)
1640 MOVE P%(X%+1,Y%,1),P%(X%+1,Y%,2)
1650 PLOT85,P%(X%,Y%,1),P%(X%,Y%,2)
1660 GCOL0,2
1670 DRAW P%(X%,Y%-1,1),P%(X%,Y%-1,2)
1680 DRAW P%(X%+1,Y%-1,1),P%(X%+1,Y%-1
,2)
1690 DRAW P%(X%+1,Y%,1),P%(X%+1,Y%,2)
1700 DRAW P%(X%,Y%,1),P%(X%,Y%,2)
1710 ENDPROC
1720 :
1730 DEF PROCfnz
1740 ON C GOSUB 1780,1790,1800,1810,18
20,1830,1840,1850,1860
1750 IF P=2 THEN Z=-Z
1760 ENDPROC
1770 :
1780 A=X%-10:B=Y%-10:Z=SQR(ABS(A))*SQR
(ABS(B)):RETURN
1790 Z=EXP(SIN(X%)*SIN(Y%)*3)/4:RETURN
1800 Z=EXP(COS(X%/2)*COS(Y%/2)*3):RETU
RN
1810 A=X%-10:B=Y%-10:T=SQR(A*A+B*B):Z=
COS(T/1.7)*EXP(-T/5)*10:RETURN
1820 A=X%-10:B=Y%-10:Z=COS(SQR(A*A+B*B
))*2:RETURN
```



```
1830 A=X%-10:B=Y%-10:Z=EXP(6-(SQR(A*A+
B*B)))/35:RETURN
1840 Z=COS(SQR(X%*X%+Y%*Y%)):RETURN
1850 Z=EXP(COS(SQR(X%*X%+Y%*Y%))*3)/4:
RETURN
1860 A=X%:B=Y%:Z=1:RETURN
1870 :
1880 ON ERROR OFF
1890 MODE 7
1900 IF ERR<>17 REPORT:PRINT" at line
";ERL
1910 END
```

NEWS NEWS NEWS

ACORN BITS

Acorn quietly unveiled a whole host of goodies at the recent Compec show at Olympia. The long awaited 32016 Second Processor is here at last. This 32 bit processor comes in the usual cream side-by-side box complete with 256K RAM and a heap of software. The software includes Acorn's own operating system called Panos (no Unix), BBC Basic, Fortran 77, Lisp, Pascal, C (not BCPL), and 32016 assembler. The entire package will set you back £899.

MUSIC

Also on the Acorn stand at Compec was the Music 500 system. This add-on unit, developed by Acorn in conjunction with Hybrid Systems, opens a whole new world of computer music for your BBC micro. The units adds 16 sound channels to your computer organised as eight voices, all controllable in amplitude, pitch, envelope, waveform, stereo position, and so on. To look after all this a music language called 'Ample' is included on ROM. The Music 500 system costs £199. A music keyboard and real time software are to be produced soon.

THE HARD STUFF

The Acorn Winchester disc system is now officially available. A choice of 10 or 30 Megabyte capacities is available with data transfer rates of up to a million bits per second. Fully compatible with Econet, the Winchester drives will be an attractive proposition to schools and other network users. The 10 Mb and 30Mb capacity systems are available at £1499 and £2229 respectively.

ACORNSOFT'S UCSD PASCAL SYSTEM

Reviewed by John Maher

Following the recent launch of ISO-Pascal, Acornsoft have now implemented the popular UCSD p-system on the BBC micro with 6502 Second Processor. Both Pascal and Fortran compilers are provided. John Maher, long time devotee of UCSD and Pascal, reports.

Product : P-system.
Supplier: Acornsoft
Price : £299.00 inc VAT.



OVERVIEW

The "UCSD Pascal System" (UCSD stands for the University of California, San Diego) was developed in the late 1970's primarily for use as a teaching package for Pascal. It has now turned into a major 16 bit operating system. Amongst micro users its main impact has been in the form of Apple Pascal, appearing in the UK in 1979/80. The p-system is too large to fit on to a standard BBC micro, but since the availability of the second processor UCSD enthusiasts have been eagerly awaiting a system such as this, which was developed by TDI for Acornsoft.

The 'p' refers to pseudo code, not to Pascal. Like ISO-Pascal from Acornsoft, the programs are compiled to an intermediate code (p-code) which has to be interpreted to run on the 6502 processor. This system produces very compact code, but is slower than fully compiled code. There have been a number of p-system versions; the Beeb runs IV.12, the Apple runs (essentially) version II. The main difference between these is in size and sophistication. Since UCSD Pascal supports program segmentation, large programs can be held on disc and segments of code brought into memory to run as needed. The operating system is written in UCSD Pascal, as are the compilers, editor, filer, etc., and these are heavily segmented. Version IV.12 has over 20 memory segments and over 180K of code. By contrast the same system configuration for the Apple uses 6 segments and needs 120K of code. As you might expect, disc use for the BBC p-system is much heavier than for the Apple version.

THE BBC VERSION OF THE P-SYSTEM

The system comes on two, nearly full, single sided 80 track discs, containing p-code compilers for Pascal and Fortran 77, a screen editor, a disc filing system, libraries for Pascal and Fortran, and various utility programs. In particular there is a program UTIL.CODE which enables users to format, copy and generate bootable discs, configure the system in various ways, and edit character fonts.

Missing are the p-system assembler and linker, the run-time symbolic debugger, and some of the utility programs described in the operating system manuals. These extra facilities are available from TDI as an Advanced Development Toolkit. For contrast Apple Pascal comes with the assembler and linker. In view of the price of this package, the assembler and linker should really have been provided.

DOCUMENTATION

Three manuals are provided: for UCSD Pascal, for Fortran 77 and for the Operating System.

The Operating System Reference Manual is well written, but should be supplemented by reading an introductory text if you are unfamiliar with the p-system. Before doing anything with your discs you should read the 70 page section in the back of this manual very

carefully. This contains details of how to get the system going from the discs provided, how to configure your system for your own machine, and descriptions of BBCUNIT and BBCGRAFIX - library units for using various BBC micro facilities.

The Pascal manual is "The UCSD Pascal Handbook" by Randy Clark and Stephen Koehler. This is an excellent book, certainly the best reference available for UCSD Pascal. It is in two parts, firstly a full description of UCSD Pascal, secondly a programmer's guide. By working through the examples in the guide you will get a good introduction to this Pascal. However, the handbook is not a book for beginners.

The Fortran 77 Reference Manual (this is the Silicon Valley Software version of Fortran 77) presupposes a knowledge of Fortran and some aspects of the operating system. It would provide an introduction for someone already proficient in Basic.

The only mention of the BBC micro in both language manuals appears on the covers. It would be very helpful if later issues of the Acornsoft p-system could at least provide some simple specific examples. This is especially true for the Fortran: the one sample program in Appendix B has text mistakes in the write statements below label 50.

Whilst the p-system documentation is excellent, I would have liked more description of the BBC micro side of the system. Acornsoft have done the same as with their Z-80 Second Processor software, namely taken another manufacturer's manuals and appended a few pages to refer to the BBC micro version. At the least, the BBC Computer P-System Manual, bound into the back of the OS Manual could have been provided as a separate booklet. The review copy was Issue 1B, June 1984; the listing of the system disc contents is out of date, especially with respect to Fortran. Whilst guesses are possible, what are the files RTLIB2.CODE, RTLIB4.CODE FORTRAN.2.CODE, and FORUNIT.CODE? (FORTRAN.2.CODE had a 'bad block' in it in any case!). In the OS Manual you will find reference to various p-system utilities not present on the system discs from Acornsoft/TDI.

The quality of the binding of the Operating System Reference Manual is poor, it was losing pages by the end of the review period.

THE REVIEW SYSTEM

The review was conducted on a BBC B loaned to me by TDI, with Tube version 0.10, DFS 1.00 and OS 1.20, together with dual double sided 80 track drives, and a 6502 second processor. This was necessary since the system crashes inexplicably on my own BBC system (see the appendix to this review).

The system can be configured to work with most Beeb systems with a 6502 Second Processor, but this is not quite as easy as it could be. As supplied, the system is configured to work with a single sided, eighty track disc drive, although it can be persuaded to work with both dual and double sided eighty track drives. Configuring is a tedious process, but makes the system much more workable once correctly done.

Booting the p-system is a complex process. Thus the master discs contain both a BBC directory with the files !BOOT, SBOOT and PSYSTEM, and a regular (6 x 512 byte code block) p-system directory. Booting the master discs is very slow, due to the absence of interleaving. Boot speed increases appreciably for copied and reconfigured discs. It should be possible to improve the speed of the system by careful 'tuning' of the disc configuration, though this may be beyond most users.

A word of warning - I would not recommend anybody to try using the p-system with a single disc drive, except where a custom application program is set up to run on its own. The Acorn p-system is just about manageable on a dual sided 80 track drive (400K), but you will find the system painful, and the reconfiguration process very tricky! The 180K is really a minimum space, not including libraries or any utility programs, or the user's own programs.

RUNNING PROGRAMS

UCSD Pascal is a complete version of the language with very many nice

extensions. It is a pleasure to work with, and very powerful, and this applies to the Acornsoft version (excluding crashes!). The p-system editor is one of the nicest screen editors I know, with none of those awkward control codes, and it can readily be used for word processing as well. On the Beeb the cursor keys are used to move around the text, the Copy key is used as the accept <etx> key. I don't like this, and prefer Ctrl-C for accept. The only problem I noticed was a tendency to give Error #400 - illegal character in text - during compilation.

BENCHMARKS

The Eratosthenes prime number sieve (BYTE 1983,8,283), for primes up to 4095 took 87 secs. The full program, for numbers up to 8190 will not fit into memory unless the boolean array used in the program is packed. With packing the program takes 5 min 31 secs! For comparison Apple Pascal takes 2 minutes 40 seconds, and Turbo Pascal running Z80 compiled code on a BBC Z80 second processor 16 seconds. I am afraid that running the PCW (Dec 1983, p242) Pascal benchmarks reveal a similar lack of pace. In Acorn UCSD Pascal the total time to run them all is 14 min 11 sec, in Turbo Pascal 2 min 23 sec. The 259 lines of code in the PCW benchmarks took 1 min 28 secs to compile, a very similar version of 244 lines took less than 2 secs in Turbo Pascal.

The 'REPEAT' benchmark in Basic takes 2 min 41 secs, in UCSD Pascal 19.08 secs. However, the maths benchmark takes four times as long in UCSD Pascal as in BBC Basic. The overall conclusion is that the Acorn UCSD Pascal system is rather slow. A nice aspect of the BBC system is that you can use the TIME function to find out just how slow it is!

The slowness of the system is unfortunate since BBC Basic, in the PCW Basic benchmarks of Nov 1982, comes fourth in a long line of 8- and 16-bit micros, behind only the SAGE II, Olivetti M20, and DAI personal computers. Don't expect to write speedy games programs with the Acorn p-system without an assembler.

However, you can run very large programs. The absolute size of programs in version IV of the p-system is governed by disc space since the system provides for extensive memory management, and uses library programs and segmentation. The efficiency of disc accesses is very critical, though hopefully this is open to improvement. This is a large system, and as usual you sacrifice speed for size.

UTILITY PROGRAMS

BBCUNIT gives the user access to the system configuration parameters from programs. There are also procedures for handling some of the BBC OS commands. A few of these were tested. Osword with A=7 (SOUND) worked well, though the timer call with A=1 seems to lose the bottom 8 bits of the 24 bit time signal.

BBCGRAFIX gives the user access to various BBC graphics, plot, draw, gcol, mode etc., though these are equally well available by simple Pascal Write(chr(vdu<number>),etc) statements. Some * commands can be used from UCSD Pascal, and the function keys can be set up in a limited fashion. Thus you can set a key to list a disc directory from the Filer, but cannot then 'Quit' the Filer! Apparently the function keys only work in the outer command level. In the UTIL program I found that a call to the character font editor CHEDIT crashed the system. The p-system utilities include a screen control unit, the p-system configuration unit (SETUP), a corrupt disc unraveller, and facilities for duplicating disc directories.

A program to transfer files between Acorn DFS and UCSD format would be a welcome addition, though users have the necessary 'hooks' to do it themselves from the OS calls.

FORTRAN

UCSD Fortran 77 is a subset of the language, as for instance it does not have complex arithmetic, and arrays have a maximum of 3 dimensions. Useful features in this version of Fortran are the ability to create overlays and to use separate compilation and libraries. You can also call Pascal modules from Fortran and vice versa. Whilst I was

able to compile the example program, I could not get it to work because I could not find the Fortran library on the discs. FORTLIB4.CODE was missing, the obvious alternative RTLIB4.CODE did not work. I suspect that the Fortran works in either 4 or 8 byte precision. We are not told in the documentation.

I can see little value in using UCSD Fortran on a micro, despite most scientists' prejudices. UCSD Pascal is very much more powerful and much easier to use, especially in its I/O format control. It is unlikely that there is any speed advantage for Fortran over Pascal on the BBC micro, both run in p-code on the same interpreter.

VALUE FOR MONEY

So far, ISO-Pascal looks as if it is the cheapest route to Pascal on the BBC micro. Running UCSD Pascal will, I regret cost £300 plus a second processor, and this puts it way beyond most home owners or schools. In addition, having bought a system you cannot run programs outside of the p-system without another p-system, and the licence only applies for one BBC micro. Softech, the American vendors of the p-system, have recently announced educational discounts. Will this apply to the Acorn version?

An alternative to ISO- or p- Pascal is to use a Z80, or slightly cheaper Torch Z80 card, and one of the other Pascals available from CP/M. My present preference would be Turbo Pascal which is available for both the above pieces of hardware. The Z80 route is also appropriate for Fortran and other languages.

SHOULD I BUY IT?

Yes, if the problems pointed out in the Appendix are sorted out, and if you have a 6502 second processor and £300 to spare. Despite the warts I like the Acorn p-system, and UCSD Pascal has a tremendous world-wide following.

APPENDIX

My own BBC runs Tube 1.10, DFS 1.20, and OS 1.20. The first p-system supplied for review refused to work properly on this hardware, though it seemed to boot properly. It crashed repeatedly; from the UTIL, from the Filer and from the Editor. In desperation I contacted TDI and told them of the problems. They were surprised to say the least, and showed me a system working satisfactorily on one of their machines, which they promptly, and very kindly agreed to loan me for the review. Whether the problems are specific to my BBC (which has no non-Acorn hardware, and only Wordwise and BCPL ROMS), or whether the problem lies with the AcornSoft/TDI p-system operating on slightly different versions of the Tube and DFS, is not completely clear. These questions need sorting out before I can recommend anyone buying the system.

USUS(UK)

This is the user group for all UCSD users in the UK. For further information, if you buy the Acorn p-system, then contact me at University of Bristol, School of Chemistry, Cantock's Close, Bristol BS8 1TS. (0272 24161 Ext 632). I have agreed with the Chairman of USUS(UK) to start off a SIG (special interest group) for the Acorn p-system. You can gain access to an extensive UCSD Pascal library via USUS(UK).



POINTS ARISING

ACK DATA BASIC COMPILER REVIEW (BEEBUG VOL. 3 NO. 6)

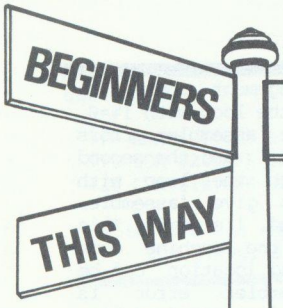
Ack Data have told us that the price of their Basic compiler on cassette is now £17.95 and not £14.95 as quoted. They are also releasing a new version to support around 90 keywords (including array, file and string handling) at £17.95 on cassette and £19.95 on disc.



WEE SHUGGY (BEEBUG VOL.3 NO.6)

Further to the reference to this game, and the absence of critical spaces, in last month's noticeboard, there is a further error in line 3180, where fairly obviously two commas have been omitted, in each case between the 231 at the end of a line and the 31 at the start of the next line.





INTRODUCING MACHINE CODE

(Part 1)

by Gordon Weston

In our 'Beginners' slot this month we present the first article in a new series which aims to explain the rudiments of assembler and machine code programming for those still teetering on the brink. If you have so far avoided the fascinations of machine code then now could be the time to take the plunge.

If you look back to the time when you were first learning programming, you may remember the sense of achievement when you entered a few lines of Basic, typed RUN, pressed Return and saw an immediate result. I hope to do the same by giving short Basic programs containing a few lines of assembly language, which you can enter or just read. All you need to start is any BBC machine and the User Guide, although I would strongly recommend the 1.2 Operating System and the Advanced User Guide as well.

Here are two simple Basic programs:

Program 1
```
10 MODE7
20 PRINT"A"
30 END
```

Program 2
```
10 MODE7
20 VDU65
30 END
```

Both programs display the letter "A" on the screen, 65 being the ASCII code for "A". (See User Guide page 486)

The 6502 microprocessor used in the BBC has its own storage spaces called registers, three of which are called the Accumulator (abbreviation 'A'), the 'X' register and the 'Y' register. Each of these three registers is capable of holding an 8 bit binary number (or byte), which in decimal terms can store integers in the range 0 to 255.

The next thing to know is that the BBC micro's Operating System has been written in such a way as to give you easy access to its routines, which you can find in a summary table on page 452 of the User Guide. If you look up the OSWRCH routine (Operating System WRite CHaracter) in the table, it gives the entry address as &FFEE and tells you that it writes the character in the Accumulator to the screen. It is not made clear, but in this routine the values in 'X' and 'Y' are ignored.

Finally, there is a Basic command "CALL" which loads the values of the resident integers A%, X% and Y% into the Accumulator ('A'), 'X' register and 'Y' register of the microprocessor. It also passes control to the routine at the address following the word "CALL". When the task of this routine is complete, control is returned to Basic. Program 3 shows this, using the same simple task as programs 1 and 2:

Program 3
```
10 MODE7
20 A%=65:CALL &FFEE
30 END
```

The ASCII code for "A", 65, is stored in A%, and then as a result of the "CALL", the value from A% is stored in the Accumulator. Control is then passed to address &FFEE where the contents of the Accumulator are printed to the screen and then control is automatically returned to Basic (a function of the CALL statement).

The CALL command is very useful for trying out Operating System routines before incorporating them in your machine code programs.

Microprocessors obey a machine code program stored in memory which is

totally composed of numbers, and most inconvenient for us to read. An assembly language exists so that machine code can be represented by easily remembered letter abbreviations (mnemonics). BBC Basic contains an assembler which translates this assembly language to machine code which the microprocessor can understand. An example instruction is "&AA" in machine code which is "TAX" in assembly language standing for "Transfer the contents of the Accumulator to the 'X' register".

An important point to remember is that when a Basic program goes wrong, error messages are often given, but when a machine code program goes wrong you get no help, except possibly for some error messages during assembly. Before running program 4, save it on cassette or disc. If when you run it, the Basic prompt symbol (">") does not return immediately, press Break, type OLD, and press Return to get back to Basic. Then check and re-check the program that you originally typed in.

Program 4 (note that square brackets appear on the screen as arrows in mode 7)

```
 10 MODE 7
 20 DIM code 100
 30 FOR I%=0 TO 3 STEP 3:P%=code
 40 [
 50 OPT I%
100 .start
110 LDA #65
120 JSR &FFEE
130 RTS
500 ]
510 NEXT
520 CLS:CALL start
530 END
```

This program should do exactly the same as the other three.

Line 20 sets aside a block of 100 memory locations (bytes) in which to store the machine code program, as it is translated from assembly language, and stores the start address of this block in the variable 'code', so that the start of this reserved block can be found by the assembler.

Lines 30,50 and 510 (see 'OPT' page 314 in the User Guide) are an assembler FOR-NEXT loop arrangement where the first pass through the loop with I%=0, sets OPT to give the 'assembler errors suppressed, no listing' and the second and final pass through the loop with I%=3, sets OPT to give 'assembler errors reported and listing'. This listing, showing the machine code program contents and location can be seen when an assembler error is reported or by removing CLS from line 520. Line 30 also contains P%=code. When the assembler is used, P% must be loaded with the start address of the block reserved in line 20 each time you pass through the FOR-NEXT loop. In this case, the start address is held in the variable 'code'.

Lines 40 and 500 contain the start and stop assembler symbols (open and close square brackets). OPT is not assembly language and is not assembled. In line 100, the full stop tells the assembler that this is a 'label' and that a Basic variable, whose name is made up from the text following the full stop, must be created containing the address of this point in the machine code program. In line 510 'CALL start' passes control to the machine code assembled at the address contained in the Basic variable 'start'. Line 110 (LDA #65) is assembly language for LoaD the Accumulator (the symbol '#' means directly) with 65.

Line 120 (JSR &FFEE) is assembly language for 'Jump to SubRoutine at address &FFEE'. We used this same routine in program 3. At line 130 RTS (ReTurn from Subroutine) in this context means go back to Basic. Up to line 520 the machine code has only been assembled and stored, but not used. The code is brought into use by CALL start where 'start' is the Basic variable containing the address to which we want to direct the microprocessor.

You can alter the value of the ASCII code in line 110 to any you like, including those not displayed on screen (such as 7 or 12) providing that they are one byte codes (See the bytes extra column on page 378 of the User Guide).

Try entering these lines, which overwrite two previous lines, in which three bytes are used to move the

cursor.

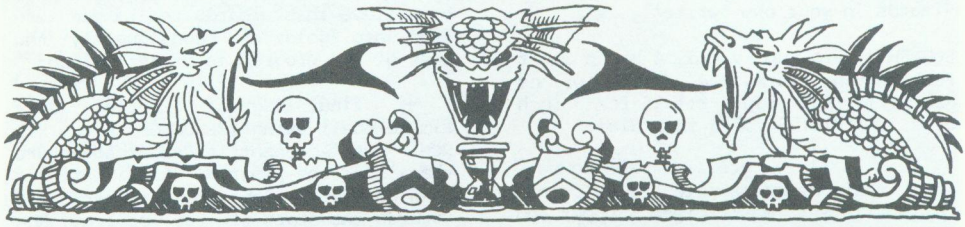
```
110 LDA #31:JSR &FFEE
115 LDA #20:JSR &FFEE
120 LDA #10:JSR &FFEE
```

This has the same effect as:

```
VDU 31,20,10
```

from within Basic (move cursor to position 20,10).

The next step will be to create a REPEAT/UNTIL arrangement in assembly language, and this will be the subject of the next article.



ADVENTURE GAMES

by 'Mitch'

Seems like Acornsoft's Space Pirates are everywhere! Not content with infiltrating everyone else's micros and zapping 'harmless' traders, they have spilled over into my dungeon.

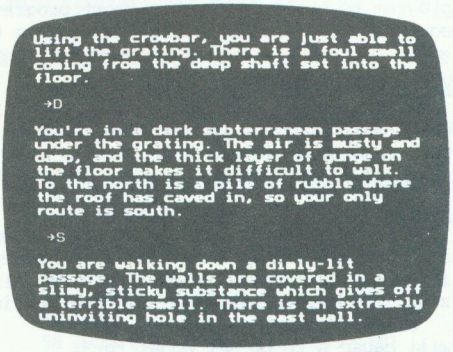
"The City of the Seventh Star", produced by Acornsoft at £9.95 inc. VAT on tape.

So here I am marooned by pirates on this distant planet outside the "City of the Seventh Star", looking for a public Teleport Box home. This new adventure breaks new ground for Acornsoft with the introduction of sound. No more lonely vigils of the night trying to solve cryptic clues by yourself; with this game you can ask your neighbours for suggestions, when they come round to complain about the noise!

The game is peopled with some very familiar faces. In the red corner is a tall, black-cloaked villain who appears to have an engine grille over his face, and in your corner is a talkative micro who makes helpful suggestions. Rat infested tunnels, electric chairs, and the bloody remnants of previous adventurers are strewn around this highly amusing science fiction world. You will have as much fun recognising the thinly disguised heroes, as solving the puzzles. Having given your name at the game's beginning you will find it used to good effect to personalise the text at later stages.

The main attraction of the game is that it refuses to take itself too seriously, and it repeatedly causes a smile with its tongue-in-cheek humour. Take for example, the 'Useless Room', this is described as containing nothing, and serving no useful purpose. My favourite villain introduces himself as "Mr. Blobov Slime" and then sticks your head into a fish tank!

On being killed the game resurrects you at a nearby location with all your hard-won goodies still in your



possession, leaving you to continue with little penalty.

This game stands head and shoulders above many others on the market and I really enjoyed it. I am sure I will finally get home before I die laughing, but just in case, I hope the Editor remembers to muck out the dragon! Now something for those of you who are wizards in your own 'write'...

SQUASH by Pro-Supply Ltd. 4 Beech Court Pocklington, York, YO4 2NE. The prices are £9.75 for tape, £11.95 for 5 inch disc, and £14.95 for 3 inch disc.

This utility program can squeeze your text messages into a tiny corner of your latest masterpiece. Upper and lower case text (plus the option of colour) can be typed into the game via a simple word editor. The data may then be compressed, recalled or stored. This is the first commercial text compressor I have seen for sale and having tried it myself, I can vouch for its simplicity of use. Having compressed the messages, the program creates a file which can be loaded into memory, at any location for your own game's use. To recall any message from within your adventure a simple CALL is made, having first set a variable to the message number required. If, like me, you prefer the task of creating the game's theme and style, rather than the actual nuts and bolts programming then this program will make a welcome addition to your book of utility spells. Unfortunately Pro-Supply appear to believe that wizards really can turn lead into gold and consequently the cost of the program is a little rich.

My final magical offering is a simple spell using BEEBUGSOFT's Toolkit ROM. *LOAD Acornsoft's Sphinx Adventure then LIST the program. You will find a mass of data lines which appear empty towards the start of the program. Now use the commands *RECOVER and LIST; Hey-Presto, all is revealed. This is caused by the ROM replacing naughty VDU disable control codes by a '#' symbol. By writing a simple loop you can use a reverse technique in conjunction with POKE ('?') commands to replace '#' characters with those VDU codes, thus making invisible data in your own games.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SETTING THE ARIES B-20 BOARD ON BREAK - A.E. Wilmhurst

When using the Aries RAM board, it is useful to be able to ensure the state of the board when a program is loaded. The following Break routine will ensure that the Aries board is turned off:

```
10*KEY10 10*FX18|M20CLS:CHAIN"file"|M
20*KEY0CALL!&FFFC|M
30*FX138,0,128
40*XOFF
```

The space in line 10 is vital. The *XOFF could be changed to *XON if the program requires this, and the name in the CHAIN command should be changed to that of the program to be loaded. This short program turns off the board, and then performs the equivalent of Break, before chaining in the program.

VIEW CONTROL CODES - Dr. F.G. Riddell

With View it is not so easy to initialise a printer directly, as you can with Wordwise, but don't forget that there is nothing stopping you doing a *BASIC, (or *B.) running a program to configure your printer, and then entering View again.

ANOTHER ODDITY IN BASIC - D. Morgan

It is quite well known that Basic does not allow you to use a variable before it has been defined, but there is an exception to this rule; if you define a variable in terms of itself, when it is not already defined, then it will equate itself to zero in order to evaluate the equation in question. For example:

```
A=A+3
```

will cause A to be set equal to 3.

MAKING MUSIC ON THE BEEB (Part 1)

by Ian Waugh

In this series of articles, the author of "Making Music on the BBC Computer", reviewed in last month's BEEBUG, explains simple music theory and suggests methods of programming and manipulating music, music data and musical effects.
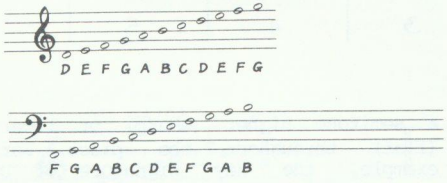
MUSIC NOTATION FOR THE BBC COMPUTER

You may have wondered if there is not a better, simpler method of representing notes than the lines and dots of traditional notation. The short answer is yes! Although other forms of notation exist and appear, on the face of it, to be easier to understand, the problem is one of communication. Traditional notation is not perfect but it is the most widely known and therein lies its strength.

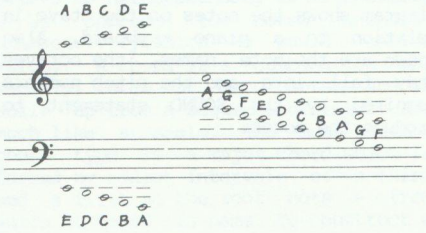
THE RUDIMENTS OF MUSIC

We'll get the rudiments out of the way as quickly as possible so we can move on to the more interesting aspect of programming. Don't try to learn them all at once - read through them and use this section for reference. It is a brief and potted summary but it should provide you with enough information to be able to take a piece of music and program it into the computer.

Music is written on a series of five lines known as a staff or stave. The pitch of a note is indicated by its position on the stave, on a line or in a space. The higher up the stave, the higher the note. The notes are given letter names, A through to G. When G is reached we start again with A.



We can extend our range of notes by placing them above and below the stave on lines called leger lines.



There are several staves. To distinguish one from another they are given clef signs which show the pitch of the notes in relation to the stave. The two most common clefs are the treble or G clef which loops around the G line and the bass or F clef whose two dots sit either side of the F line.

The interval in pitch between two notes of the same letter is known as an octave. The interval between two adjacent notes is either a tone or a semitone. The User Guide tells us that the pitch values in the SOUND statement alter the pitch by 1/4 of a semitone. This plays through the complete pitch range:

```
10 FOR pitch=0 TO 255
20 SOUND1,-15,pitch,10
30 NEXT
```

This alteration plays a semitone scale:

```
10 FOR pitch=0 TO 252 STEP 4
```

We can play octaves like this:

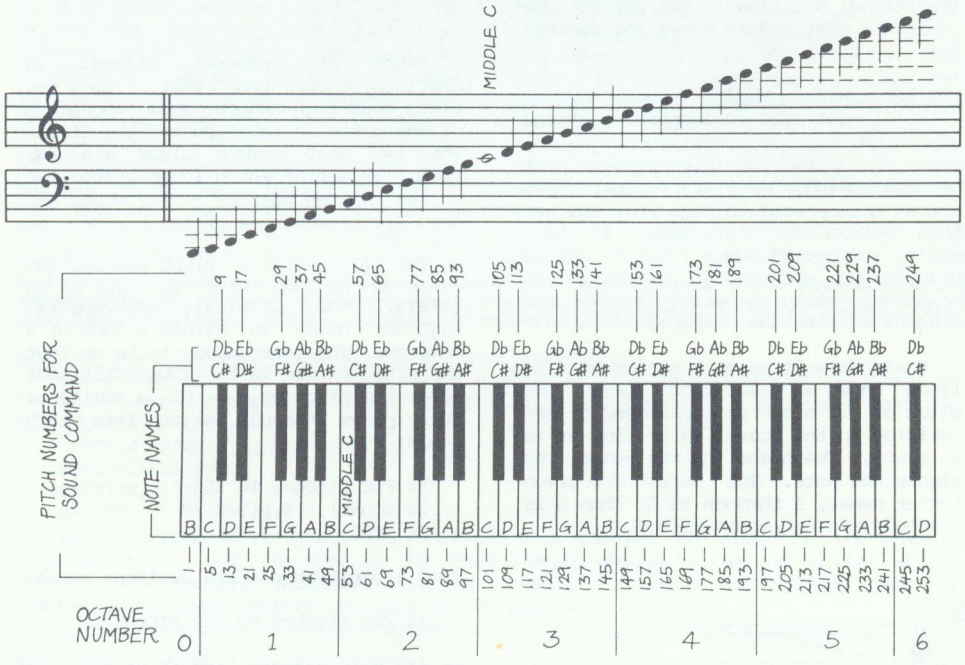
```
10 FOR pitch=1 TO 241 STEP 48
```

None of these are particularly musical. They all lack a sense of... pitch.

```
10 FOR scale=1 TO 8
20 READ note
30 SOUND1,-15,note,10
40 NEXT
50 DATA 53,61,69,73,81,89,97,101
```

→

The above plays a scale of C major and sounds more musically satisfying than a series of semitones. We say it has a tonality or an affinity towards a certain group of notes, a musical key. The intervals between the notes in this scale are: tone, tone, semitone, tone, tone, tone, semitone. We can play a scale based on any note simply by adding these intervals to it. To tie all these ideas together, the following diagram shows the notes on the stave in relation to a piano keyboard. Also shown are the note names, the octaves they fall into and the pitch numbers required by the SOUND statement to produce the notes.

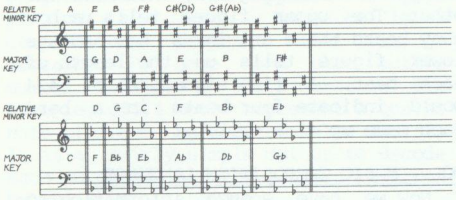


The notes have been moved or transposed an octave. The staves show most of the musically useful notes. If the Cs were lined up we would often find ourselves an octave short in the lower range.

Some notes have two names, eg C# and D flat. A sharpened C is the same as a flattened D. Both produce the same pitch and are known as enharmonics. It is convenient to refer to all such notes as sharps. Apart from anything else, the computer has a built-in sharp sign (#) and it avoids any possible confusion with the letter 'b'.

SCALES

There are 12 notes in an octave from which twelve major scales can be formed. Black notes are indicated by sharp (#) and flat (b) signs positioned on the stave to form a key signature (see overleaf). Each note with the same name as the one upon whose line or space the sharp or flat lies is played a semitone higher (sharp) or lower (flat) throughout the piece. For example, the key signature of D contains two sharps, F and C. If we start on a D note and move upwards playing the F# and C# notes as we go we will play a major scale. The figure also mentions minor scales: produced by playing a different sequence of

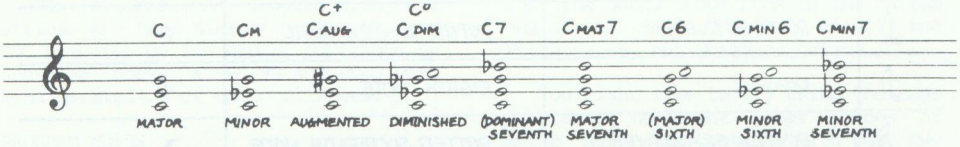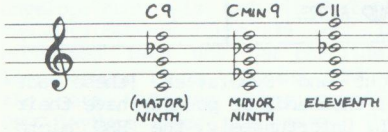


intervals. To complicate matters, there are technically two forms of minor scale - the melodic and the harmonic. They share the same key signature and differ only in the way they are played - a slightly academic point and one you need not be too concerned about. The diagram below shows the notes of the major and minor scales of C.
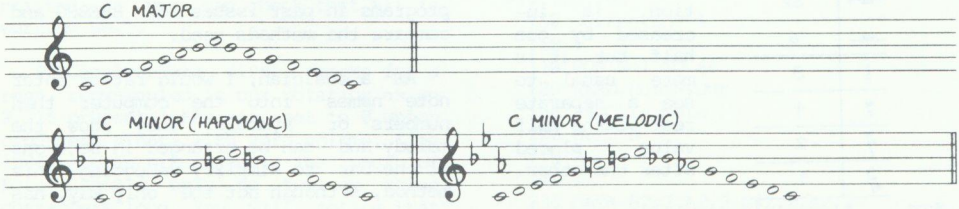
To play a note which is not a part of the designated key we place a sharp or flat immediately before it. The change only affects that one pitch, not notes an octave up or down, and lasts only for the remainder of that bar. A natural ( ) is used to return a sharp or flat to normal. Used like this, these signs are known as accidentals.

CHORDS AND HARMONY

Now is a good time to introduce chords and harmony, primarily because they relate strongly to scales. Some common chords are shown below. Harmony refers to notes sounding simultaneously - as opposed to notes sounding consecutively which we would call a melody. A chord is a combination of, usually, three or more notes and is built up from a sequence of intervals, much like a scale. The most common chord type is a major chord which is formed by adding intervals of a third and a fifth to the root note - from which it takes its name. To construct a C major chord, start on C and count that as one. Move up the scale until you reach three and that will be the third, E. The fifth is calculated in the same way and leads to a G note. Together, these three notes form the chord of C major.







A minor chord is produced by flattening the third (a D# in the case of a C chord) and this produces a 'sad' sound. Many chords are named according to their construction. Names are given in terms of flattened, augmented (sharpened) and added intervals. Chords are useful for all manner of musical things and provide a convenient method of adding harmony to a melody.

NOTE DURATIONS

The duration of a note is represented by the notation below.

NOTATION	ENGLISH NAME	AMERICAN NAME	DURATION VALUE
𝅝	SEMIBREVE	WHOLE NOTE	32
𝅗𝅥.	DOTTED MINIM	DOTTED HALF NOTE	24
𝅗𝅥	MINIM	HALF NOTE	16
♩.	DOTTED CROTCHET	DOTTED QUARTER NOTE	12
♩	CROTCHET	QUARTER NOTE	8
♪.	DOTTED QUAVER	DOTTED EIGHTH NOTE	6
♪	QUAVER	EIGHTH NOTE	4
𝅘𝅥𝅯.	DOTTED SEMIQUAVER	DOTTED SIXTEENTH NOTE	3
𝅘𝅥𝅯	SEMIQUAVER	SIXTEENTH NOTE	2
𝅘𝅥𝅰	DEMI-SEMIQUAVER	THIRTYSECOND NOTE	1

Duration is relative only to other notes in the piece. It is important to realise that it bears no relation to the speed or tempo. Rest values are shown here. Like a note, if a rest is followed by a dot its duration is increased by one half but it is more usual to see a separate rest of the half value placed after the other.

NOTATION	DURATION VALUE
𝄻	32
𝄼	16
𝄽	8
𝄾	4
𝄿	2
𝅀	1

THE TIME SIGNATURE

This is written at the beginning of the piece as two figures, one above the other. The upper figure tells us how many beats there are in a bar and the lower figure tells us the length of each beat. A time signature of 4/4 would indicate four beats in a bar, each made up of a crotchet.

MCL - MUSIC COMPOSITION LANGUAGE

Now we have enough information to tackle some programming. Large computer-based synthesizers such as the Fairlight and Synclavier (these cost several thousands of pounds) have their own MCL. Unfortunately, the BBC micro does not contain an MCL so we must devise and program our own. This can be quite an exciting and challenging task. You may like to refer to the music programs in past issues of BEEBUG and compare the methods used.

As a musician, I would rather enter note names into the computer than numbers or symbols. They show the melody and can be arranged in sections of one bar to simplify debugging. This method, although not the only way, has

many advantages, especially when we want to refer to notes and note progressions - as opposed to just playing a tune - such as in a compositional program. Notes can be described by note name plus octave number followed by their duration. For example, C#4,4 would represent a quaver note produced by a pitch value of 153. It is a fairly simple matter to decode such an entry into the numbers required by the SOUND statement.

The program is from Making Music on the BBC Computer and illustrates the principles behind the note to number conversion. PROCAnalyseNote does the work and variations on this procedure are used throughout the book. I found it better to adapt the procedure to particular requirements than to try to write a universal routine to cover every possibility. You should be able to enter data from a sheet of music and program the computer to play the melody. If you have any thoughts or ideas about note representation or music programming, I'd be interested to hear them.

The figures and program with this article are from Making Music on the BBC Computer by Ian Waugh, published by Sunshine books at £5.95 and used with kind permission of the publishers.

PROGRAM NOTES

The program plays the first eight bars of Mozart's Rondo Alla Turca. The variable, Key, in line 120 is set to 1 (more on this in a moment) and the envelope number is put in CurrentEnv. The data between lines 230 and 310 is organised into one bar per line. The loop between lines 170 and 200 reads the note and its duration and calls PROCPlayNote once per note. This in turn calls PROCAnalyseNote which returns with values for Env, Pitch and Dur which are used in the SOUND statement. Note$, Pitch and Dur are printed so you can check that all is working well.

PROCAnalyseNote first checks for a rest, represented in our notation as an R. If present, Env is set to 0 which produces a note with zero volume. Line 1080 is an error check. The length of Note$ should only be two or three characters long. You will notice that the procedure does not check for sharp signs, merely the length of Note$. The name of the note is extracted followed by the octave.

Line 1110 calculates the pitch. The INSTR function determines how far along Scale$ the note is and this is multiplied by 4 as there are four pitch increments in a semitone. Then 48 multiplied by the octave value is added as there are 48 pitch values in an octave. The variable, Key, gives us an easy method of transposition. With a value of 1 it plays as written; 5 will take the tune up a semitone. Negative values can also be used.

Lastly, the value of Pitch is checked to ensure it falls in the sound chip's range and the procedure ends. The error checking is not essential but you may find it useful. Add this:

```
205 Key=Key+4:RESTORE:GOTO 170
```

The program will eventually stop with an error message when Pitch rises above 255. If you REM out line 1120 you will hear how, as the pitch rises above 255, the sound loops back to the bottom end of the scale. Key only affects the Pitch value, not Note$ or Octave.

You might like to try using the same program to implement other pieces of music before we continue with part two of this series in the next issue.

```
  10 REM Program MUSICP1
  20 REM Version B1.0
  30 REM Author Ian Waugh
  40 REM BEEBUG Jan/Feb 1985
  50 REM Program subject to copyright
  60 :
 100 ON ERROR GOTO 1150
 110 Scale$=" C  C# D  D# E  F  F# G
G# A  A# B"
 120 Key=1:VDU15
 130 :
 140 ENVELOPE1,1,0,0,0,0,0,0,126,-2,0,
-10,126,100
 150 CurrentEnv=1
 160 :
 170 FOR N=1 TO 46
 180 READ Note$,Dur
 190 PROCPlayNote
 200 NEXT N
 210 END
 220 :
 230 DATA B2,2,A2,2,G#2,2,A2,2
```

→

```
  240 DATA C3,4,R,4,D3,2,C3,2,B2,2,C3,2
  250 DATA E3,4,R,4,F3,2,E3,2,D#3,2,E3,2
  260 DATA B3,2,A3,2,G#3,2,A3,2,B3,2,A3
,2,G#3,2,A3,2
  270 DATA C4,8,A3,4,C4,2,G3,1,A3,1
  280 DATA B3,4,A3,4,G3,4,A3,2,G3,1,A3,1
  290 DATA B3,4,A3,4,G3,4,A3,2,G3,1,A3,1
  300 DATA B3,4,A3,4,G3,4,F#3,4
  310 DATA E3,8
  320 :
 1000 DEF PROCPlayNote
 1010 PROCAnalyseNote
 1020 PRINT Note$,Pitch,Octave
 1030 SOUND1,Env,Pitch,Dur*1.5
 1040 ENDPROC
 1050 :
 1060 DEF PROCAnalyseNote
 1070 IF Note$="R" Env=0:ENDPROC ELSE E
nv=CurrentEnv
 1080 IF LEN(Note$)<2 OR LEN(Note$)>3 T
HEN PRINT"ERROR IN DATA ";Note$:PRINT"N
ote Number ";N:STOP
 1090 IF LEN(Note$)=2 THEN NoteName$=LE
FT$(Note$,1) ELSE NoteName$=LEFT$(Note$
,2)
 1100 Octave=VAL(RIGHT$(Note$,1))
 1110 Pitch=Key+INSTR(Scale$,NoteName$)
/3*4+(Octave-1)*48
 1120 IF Pitch<0 OR Pitch>255 THEN PRIN
T"ERROR IN PITCH DATA ";Note$;" Pitch =
 ";Pitch:PRINT"Note Number ";N:STOP
 1130 ENDPROC
 1140 :
 1150 ON ERROR OFF:MODE 7
 1160 IF ERR<>17 THEN REPORT:PRINT" at
line ";ERL
 1170 END
```

ADDENDUM TO ACORNSOFT P-SYSTEM REVIEW

Subsequent to the review of the Acornsoft p-system, on page 7, we have now been able to establish that the system crashes referred to in the appendix appear to have been caused by the hardware configuration used rather than the p-system itself. In further tests with other hardware the p-system has run quite successfully, though the other comments relating to software and documentation still stand.

Acorn say that on a very small percentage of machines, timing problems can arise when a second processor is in use, particularly when the tube is being driven hard by complex software such as the p-system. If you suspect that your system, with second processor, suffers from otherwise inexplicable crashes or failures, you should contact your nearest Acorn dealer.

All official dealers have been notified by Acorn of a test for this condition and a hardware fix that they can carry out, which will be free of charge to the customer. Apparently the timing problem is caused by the cumulative effect of several components operating only just within their individual tolerances in a highly critical situation.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

WHICH DAY IS IT?

The short function below allows the day of the week to be calculated from a date entered. Note that the format required by the code below is fairly exacting; the format being DD:MM:YYYY, where DD is the two digit (with a zero preceding if necessary) date, MM likewise the month and YYYY the year. The ':' may be replaced by most characters, but it is best to standardize on just one separator.

```
  100 DIMday$(7):FORA%=0TO7:READday$(A%):NEXT:CLS:PRINT
  200 REPEAT
  300 INPUT LINE"date:"A$:D%=VALA$:M%=VALRIGHT$(A$,7):Y%=VALRIGHT$(A$,4)-1600
  400 PRINTday$(FNday)"DAY"
  500 UNTIL0
 2000 DATA NO ,SUN,MON,TUES,WEDNES,THURS,FRI,SATUR
 3000 DEFFNday:L%=(Y%MOD4ORY%MOD100=0ANDY%MOD400)>0:y%=Y%-1:m%=M%-1:=((Y%+y%DIV4-y%
DIV100+y%DIV400+31*m%-(m%DIV2)-(m%=8)-(m%=10)+(m%>1)+(L%ANDm%>1)+D%+6)MOD7)+1AND(D%
>0ANDD%<(32-(m%MOD7MOD2)+(m%=1)+(L%ANDm%=1))ANDM%>0ANDM%<13ANDY%>0)
```

THE LATEST PRINTERS SURVEYED

by Tim Powys-Lybbe

Since we reviewed the popular Epson FX80 and similar printers over a year ago several new machines, including two colour printers, have appeared. We asked our expert on printers, Tim Powys-Lybbe, to survey the latest printers, and see how they compare with the Epson.

Datac 109V	£320
DRG Ensign 1650	£300
Epson FX80	£370
Integrex Colourjet 132	£400
Kaga Taxan (Canon PW1080)	£290
Mannesmann Tally MT80	£210
Mannesmann Tally MT160	£520
Seikosha GP-700A (colour)	£320

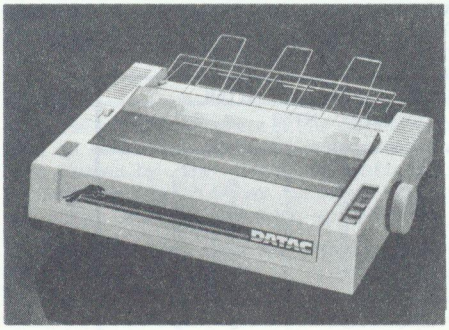
This review covers the eight printers listed including my eighteen month old FX80, principally as a standard of comparison for 'Epson compatibility'. Two of the printers were claimed to have Epson compatibility and a third was almost compatible, a fairly desirable feature in the interests of standardization. With the exception of the Mannesmann Tally printers, all the others are of Japanese origin.

	DATAC	FX80		KAGA		MT160	
		Ensign		Integrex	MT80		Seikosha
Print quality	4	5	5	4	5	4	5	3
Print speed	3	4	4	1	5	3	5	1
Graphics quality	4	5	5	5	5	4	5	3
Graphics speed	3	3	4	1	5	3	5	1
Graphics linearity	5	2	5	5	5	4	4	5
Thread paper	3	4	2	4	5	4	3	2
Near letter quality	0	3	0	0	5	0	5	0
Right justification	0	0	0	0	0	0	5	0
Underline	4	5	5	2	5	4	4	0
Noise (0=noisy)	2	3	3	5	3	2	3	0
Italics	0	3	3	3	3	0	0	0
Condensed	4	5	5	4	5	4	5	0
Elite type	4	5	5	0	5	0	5	0
Double width	4	5	5	5	5	5	5	4
Double height	0	0	0	5	0	0	0	0
Emphasised	3	5	5	0	5	5	5	0
Double Strike	3	5	5	4	5	3	3	3
Definable characters	3	5	5	0	5	0	0	0
Coloured text	0	0	0	5	0	0	0	4
Coloured background	0	0	0	5	0	0	0	0
Subscript	3	5	5	0	5	0	1	0
Superscript	3	5	5	0	5	0	1	0
Block graphics	4	0	0	5	0	2	0	0
Separated graphics	0	0	0	5	0	0	0	0
Appx. price inc VAT	320	300	370	400	290	210	520	320
	DATAC	FX80		KAGA		MT160	
		Ensign		Integrex	MT80		Seikosha

The main results of the tests are shown in the table. The tests were twofold, first to run a program consisting of most of the control codes for that printer and second to perform a graphics shaded dump. Special programs were written for each printer for both these tests.

The scores of Ø to 5 are on an arbitrary scale with 5 meaning the best and Ø meaning either the worst or that the facility does not exist.

The prices were the cheapest advertised in the October edition of Personal Computer World.



DATAC 1Ø9V

The Sales Director claimed it had 'absolute compatibility' with the Epson RX8Ø. This was not true in one respect, that of the ESC "*" graphics mode selection. The actual printing was very dark (good in itself) which however meant that emphasised and emphasised double strike were almost identical. The graphics dump suffered from the same over inking but the precision was good and the printer has one particular advantage, that it can print circles correctly.

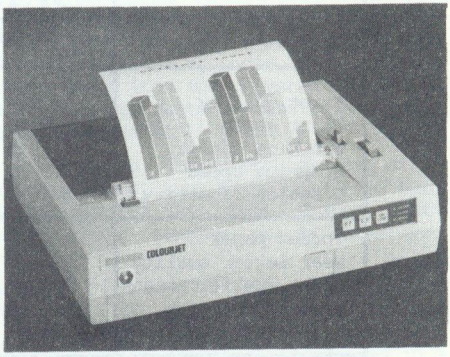
DRG ENSIGN 175Ø

This printer was claimed to be Epson compatible by Micro Peripherals, though not by Twickenham Computer Centre who provided the review machine. The only non-compatibility I could find was with the defined characters on my FX8Ø. Additionally it could not do the FX8Ø plotter graphics and so could not print circles correctly.

The print quality was identical to the FX8Ø with one exception: that it had an additional print mode called 'Fine' (Near Letter Quality, or NLQ, by Micro-P). This was the same typeface as the FX8Ø but with the gaps between dots filled in. It produced a very pleasant result, much better than anything the FX8Ø can do. But it is not a typewriter typeface and thus not really NLQ.

EPSON FX8Ø

This is still going strong after eighteen months of use, though now on its second print head. It produces good matrix print with a wide range of facilities that others are now beginning to exceed. It has one enormous advantage to me, that it will print circles circularly and cope with all graphics dumps with considerable precision in terms of pattern clarity.



INTEGREX COLOURJET 132

Although made in Japan, in view of what transpired I do not think this was a totally Japanese machine and think that the ROM has been rewritten in this country, possibly to provide a fast Teletext colour screen dump.

The excellent feature of this machine was the total clarity of the colouring of screen dumps. I could not detect any imperfection in large sheets of the same colour. It coped superbly with the very difficult picture of a desert island formed by an Acornsoft Creative Graphics program. While the graphics dumps are not fast, taking about 7 minutes, the results are well worthwhile (until someone comes along with a printer that does as well but faster). You can even print in colour

onto overhead projector transparencies, though these must be of a special type which Integrex can supply. The only problem with the graphics dumps is that it requires special paper for the best results: even top copy typewriter paper gave inferior results.

The tragedy with this printer is that not all the bugs have been removed from the ROM. When I received it the printer even missed out the last character on every, yes every, line; Integrex provided me with the latest ROM after I had pointed some faults out to them. The typing facilities, as opposed to the graphics dump facilities, were still not correct.

The machine came with a Teletext dump program; while this worked with some screens, it did not reproduce a test screen of mine correctly.

The blessing of this machine was its quiet operation: one can only just hear the jet head moving from side to side, and there is none of the strident rasp of the matrix.

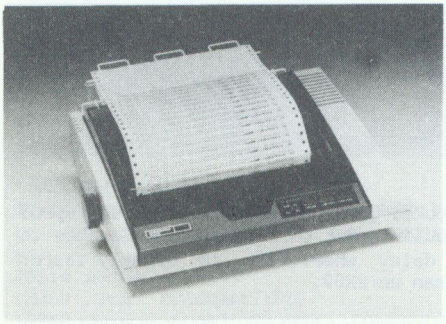
KAGA TAXAN

This printer has only recently appeared on the market, and with little attempt to disguise it, is being sold both as the branded Canon and as the less well known Kaga.

This printer has more facilities than the FX80, and it is compatible, with the sole exception of the same user defined character controls as on the Ensign. It will dump a screen linearly like the FX80, but 50% faster. Its text printing is faster than the FX80 and it has a superb Near Letter Quality mode, whose characters do resemble those of a typewriter.

MANNESMANN TALLY MT80

This is a bottom end of the market machine of probably far east origin as it has little resemblance to other Mannesmann machines of my acquaintance. As you would expect it is slow, with limited facilities and does not produce the clearest of print.

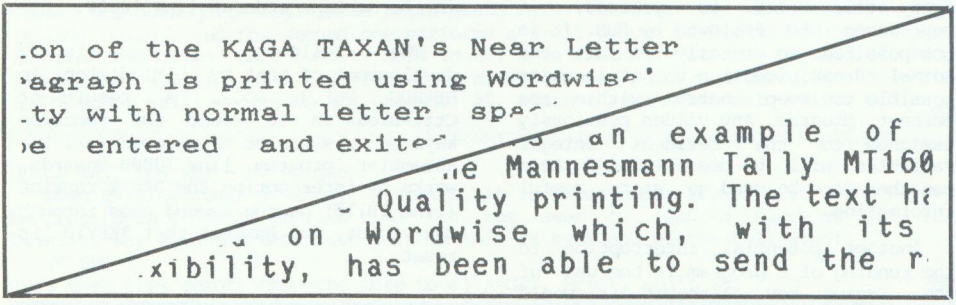


MANNESMANN TALLY MT160

This is the only machine that was not of Japanese origin. Its weight and solidity seemed to confirm its Teutonic origin. It lacked a few features but also had several possessed by no other printer in this survey.

The 'Correspondence Quality' mode enables this machine to be used in place of a daisy wheel printer. While the print quality is detectably matrix, the advantage is that there is a usable facility to proportionally right justify on the machine, as the illustration shows. This provides a high quality print with a very even spacing between letters and words of each line.

If the print quality is acceptable (and emphasised Correspondence Quality





looked good to me), its price and speed would render this machine preferable to a daisy wheel. It is clearly faster than an FX80.

Another feature of this machine is the ability to set the 'DIP switches' without having to delve inside it. This is performed by answering 'yes' and 'no' using the two appropriately marked keys on the front of the printer.

SEIKOSHA GP700-A

This is the second colour printer in this survey. I regret to say that there is nothing to commend on this printer. It is extremely noisy, the colours are poor even after obtaining replacements from DRG, the main distributors, the text print quality is poor and the facilities are extremely limited.

SUMMARY

There are three printers that have something special to commend them. The Integrex for its coloured screen dumps, the KAGA for its print quality and speed and the Mannesmann MT160 for its proportionally right justified correspondence quality, and speed. Of the others the FX80 is now looking expensive for what it offers, the Ensign is a strong contender for one's purse and the MT80 is not bad value at the bottom end of the market.

All printers were supplied for review by the main distributors except for the Ensign and the Kaga which came from Twickenham Computer Centre.

DISABLING BREAK

by Tim Powys-Lybbe

Have you ever yearned to make your program foolproof against the frantic fingers of today's computer age kids? Well here's the routine which will protect your program against forced entry, and do wonders for your self esteem.

This small routine will enable you to stop people from breaking into your program. The idea is a simple one, based on the re-programming of function key 10 (the Break key). The machine code routine sets up this key so that every time Break is pressed, it reproduces OLD followed by RUN. It is not possible to totally inhibit the normal Break function but it is quite possible to keep control within the current program. Any values previously assigned to the resident integer variables will be preserved and these can therefore be used to store useful information.

Another potential interruption to the running of a program is the use of the Escape key. Although it would appear that Escape can be easily disabled, the combination of Escape with Ctrl-Break is very powerful and succeeded in breaking into the original program (lines 120 to 140 below). So the Escape disable had to be included in the machine code at line 10060.

The following program is a development of that by G.Middleton in BEEBUG Vol.3 No.2. A result of Ctrl-Break is to clear the function keys so this has to be allowed for. The assembler program, line 10000 onwards, works by intercepting the Break routine (line 10110) on the second pass through the vector, and ensures that *KEY10 is reset.

The re-programming of the Break key is specified as a character string assigned to A$ in line 10090. As listed it performs OLD followed by RUN as already stated. The content of A$ can be changed, the only restriction being that A$ should not contain more than around 180 bytes in length. It is easiest if your program runs from the computer's PAGE value at start up; though it is possible to relocate PAGE through A$. It is best to start experimenting with the content of A$ only after you have proved your entry of the program.

Once the machine code has been assembled, another program can be CHAINed in, but anything that overwrites the machine code in memory, in page &A on disc machines or page &D on tape, will corrupt this code and cause a total machine crash.

Lines 120 to 140 listed below are there purely to demonstrate the use of the routine and should be replaced in practice by your own program, which can now be secured, against accidental (or deliberate) pressing of the Break key.

```
 10 REM PROGRAM NOBREAK
 20 REM VERSION B0.3
 30 REM AUTHOR  T.POWYS-LYBBE
 40 REM BEEBUG  JAN/FEB 1985
 50 REM PROGRAM SUBJECT TO COPYRIGHT
 60 :
 100 PROCAssemble
 110 CALLDisable
 120 MODE7
 130 PRINT"This proves that it works..
."'""Did you hear the bleep every time
the"'""little darlings pressed BREAK or"
'"CTRL BREAK or even ESCAPE, CTRL and"'
"BREAK?"
 140 END
 150 :
 160 REM Put the rest of your program
here
 170 :
10000 DEFPROCAssemble
10010 OSBYTE=&FFF4:OSWRCH=&FFEE:CLI=&FF
F7
10020 FORPass=0TO1
10030 P%=&A00 : REM Use P%=&D01 with ta
pe machines; P%=&A00 is for disc machin
es.
10040 [OPTPass*2
10050 .Start BCSStart1:RTS
10060 .Start1 LDA#7:JSROSWRCH:LDA#200:L
DX#1:JSROSBYTE:LDX#KeyMessage MOD256:LD
Y#KeyMessage DIV256:JMPCLI
10070 .KeyMessage
10080 :
10090 ]:A$="KEY10O.|MRUN|M":$P%=A$:P%=P
%+LENA$+1:[OPTPass*2
10100 :
10110 .Disable LDA#&F7:LDY#0:LDX#&4C:JS
ROSBYTE:LDA#&F8:LDX#Start MOD256:LDY#0:
JSROSBYTE:LDA#&F9:LDX#Start DIV256:LDY#
0:JMPOSBYTE
10120 :
10130 ]NEXT
10140 ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

DOUBLE USAGE - D. Morgan

If you are really concerned with reducing the number of variables used by a procedure, for example when a DIM command is used, don't forget that you can assign a result to a variable used in the calculation. By way of example, consider:

```
DIM N% N%                    A$=RIGHT$(A$,4)+LEFT$(A$,3)
```

These are both quite legitimate Basic statements. The first one will reserve N%+1 (0..N%) bytes, and the second one performs a small amount of string manipulation on A$, and then puts the result back in A$. Note that the variable on the left hand side is not altered until the right hand side has been FULLY evaluated; so A$ in the LEFT$ above still refers to the original A$, and not an intermediate value.

QUICK WAIT FOR KEY - D. Morgan

When writing 'user friendly' programs, it is often necessary to wait for a key to be pressed in order to provide time for the user to read a piece of text. The briefest way in which Basic can perform this is by means of the following:

```
IF GET
```

This uses less memory than the more usual A=GET. Brief, ain't it!

COMPUTER GAMES FOR THE BLIND

by David Calderwood

David Calderwood, himself blind, describes how he has adapted a variety of computer games so that they may be played by the blind on the BBC micro.

It was the concept of adventure games that started me off. Like radio having the edge on T.V., a text only adventure game makes use of the players' imagination - and that can build pictures far more fantastic than the most impressive main frame computer graphics. Using a BBC micro and Votrax "Type and Talk" the *FX5,2 command read me the text of Sphinx, a brilliant starter game (although there is no easy 'save game' facility). If the text was not heard properly the first time then "LOOK" could be entered for an immediate recap. Since that time I have played a number of adventures, the limiting factor being my brain rather than not being able to see.

But why stop at text games? Backgammon and Chess are played by the blind using tactile boards; surely something could be done in this field. Using "Talking Basic" I set about doctoring "Beeb Gammon" so that it read out the written moves with the addition of a review command which simply read out the players' positions on the board. The game is simply played on the tactile Backgammon board available from the Royal National Institute for The Blind. So far I have not been able to find a Chess program that can be suitably modified. Software houses are not that keen on lending out programs for such trials by the most minor of computer minority groups! The advent of "Microspeak" firmware has, however, made the possibility of Blind Monopoly come about, and in a version that can be played by either blind or sighted players, or both together.

But again why not action games? The next step was "Fruit Machine" - after all a sighted player only registers the type of fruit with his or her eyes; so they can just as easily be shouted by the speech synthesizer and the Beeb sonics can do the rest. I had to leave a little longer time for the nudge option but apart from that it's identical to a visual game. Pinball was more of a challenge. "The Who" sang of a deaf, dumb and blind boy who was a pinball wizard. How could I make this possible for the blind? I started by a table layout based on a two dimensional array BOARD%(X,Y). This was then filled with a 1, 2 or 3 etc. The "ball" then rolled down the board and reacted to the number it saw on the matrix, e.g. if 1 stood for a bumper then PROCBUMPER was called up thereby producing a bumper type sound, the speaking of the word "bump" and the direction in which the ball continues its downward movement (left, right or up for a second bump).

Other numbers stood for other goodies like rollers, flippers, spots etc. Little extras like bonus scores when buzzers sound made all the difference between a boring and an addictive game. The ball position on the X axis could be manipulated by the initial firing of the ball and by left (Z) or right (/) nudges. Too many nudges brings up the "tilt warning" message.

The success of pinball and its popularity drove me on to write a blind version of Pacman and Asteroids, Noughts and Crosses, and various card games. The limitation is that of "band width" - getting as much information over to the player as possible in the shortest possible time. Subtle use of speech and sonics are important and the use of a second speech output would make possible even faster games, i.e. one voice giving your position and the other giving the position of the monsters - listening to two voices at once is not difficult. Alas this stereo information has not been tried; Acorn were not impressed with the idea of using their speech chip as the second information source.

I do hope that this article has stimulated a few ideas in readers' minds and banished the idea that Simon is the only computer game that a blind person can cope with. The Votrax will speak any PRINT statement and screen layout is immaterial. If you think that you may have any programs that would be of particular interest to blind people, then send me anything you may write on cassette, or 40 or 80 track disc. You won't make a penny out of it but you will shed a little light in a dark world.

[David Calderwood organises a user group for blind users of the BBC micro, and distributes to members a quarterly magazine called 'Computer Talk' on an audio cassette, complete with computer programs on the reverse side. The Votrax 'Type and Talk' system with 'Talking Basic' is available from John Tilitch of Sensory Information Systems, 2B England's Lane, London NW3 4TG. talking Basic was developed by Dr Tom Vincent at the Open University, Milton Keynes, and 'Micro Speak' is the latest version of this in EPROM. Anyone wanting more information, or interested in helping in any way (by reading BEEBUG aloud for example) can contact David Calderwood at Hafan, Minfordd, Penrhyndeudraeth, Gwynydd LL48 6HP.-Ed]

TUBE COMPATIBILITY OF ROM SOFTWARE

List compiled by Benjamin Rietti

With 6502 second processors now becoming more widely available, we thought it would be helpful to members to list the ROMs that do, or don't work with a 6502 second processor active. Fortunately, Benjamin Rietti, of Viewfax Tubelink (*258216#) fame, has already compiled a very extensive list, and he has given us permission to reproduce this list here. The information refers to the current version or the stated version of the ROMs listed. If you know of any ROMs not mentioned, then please let either us or Benjamin know so that the information can be of use to others.

TABLE OF ROM SOFTWARE

ACORN MONITOR	C	DATAGEM	I	HELP	C	STARBASE	P
ADDCOMM	I	DATASTORE	C	ISO PASCAL	C	STARMON	C
ADE	P	DECCE TERMINAL	I	LISP (ACORN)	C	STARSTICK	C
AID	P	DFS UPGRADE	I	MACH1	I	TERMI	P
AIDS	P	DISC DOC. 1.0B	C	MICRONET ROM	I	TOOLKIT(BEEBUG)	I
AMCOM DFS	P	DISC SERVANT	P	M'TEST FONT ROM	C	TOOLKIT(D-WARE)	I
BCPL	C	DMON 1.03	C	MULTI-FORTH 83	I	TOOLSTAR	P
BEEBASE-1	I	DYKASAY	I	MUROM	P	ULTRACALC (NEW)	C
BEEBCALC	I	EDWORD	I	PASCAL-T HCCS	C	UROM	P
BEEBFONT	I	EPSON FX80	C	PRESTEL 4.80n	P	VIEW 1.4/2.1	C
BEEBMON	I	EXMON	P	PRINTER MON.	C	VIEWSHEET	C
BEEBPEN	P	FORTH HCCS	C	PRINTMASTER	C	WATFORD DDFS	C
BUFFER/BACKUP	P	GDUMP 2.01	C	ROM MANAGER	C	WAT. DFS 1.40	C
CARETAKER	I	GDUMP 3.00	C	SCDUMP	C	WORDWISE	I
COMMUNICATOR	P	GRAPH. EXTN.	P	SLEUTH	I	WORDWISE+	C
COMMUNICATOR 2.1	C	GREMLIN	P	SPELLCHECK II	C	WORKSTATION 1.4	C
COMMSTAR	P	HEBREW 1.00	C	SPY2	P	XCAL	P

The coding system:

C- Completely Compatible
P- Partially Compatible
I- Incompatible in all respects

List courtesy of Viewfax and Tubelink (258216).

ASSEMBLER ARITHMETIC THE EASY WAY

by Derek Chown

Some of the easiest Basic programming tasks become difficult and time consuming when using assembler. Derek Chown introduces a method of making assembler arithmetic nearly as easy as in Basic.

You are faced with a major problem the instant you attempt to rewrite a Basic program in Assembler Language, which helps to bring home to you just how much the Basic interpreter is doing. The problem is that of simple arithmetic. The 6502 instruction set cannot directly cope with floating point or high precision arithmetic.

The machine code routines required for this already exist in the Basic ROM. This program allows you to use these ROM routines from your own assembler language program, saving you the trouble of writing your own. However, the price that you have to pay is that all numbers to be used by the ROM routines have to be suitably packaged for them.

Five functions are provided by the accompanying program. These allow you to add, subtract, multiply and divide integer and floating point numbers, producing either integer or floating point answers. The fifth function enables you to find the address of a Basic variable.

The ROM routines are designed for use with Basic variables. When using them from an assembler program you have to work hand in hand with the Basic variable system. All numbers must be presented to the ROM routines in the same form as Basic variables are presented. Thus numbers must be stored in a particular format (see the later section), and in memory locations that they would occupy if they were Basic variables in a Basic program.

To achieve this, variables should first be 'declared' in Basic and then processed by your assembler program (using the routines presented here). The declaration of variables in Basic simply takes the form of assigning a value to that variable (e.g. X=4).

The value assigned is not crucial as long as Basic is informed of the variable and sets aside some memory for it. You can substitute your own values later from assembler and indeed, you will want to do this when manipulating these numbers in your assembler program. All the time, the variable's value is kept in the location assigned to it, at the start, by Basic just as it is in a Basic program. This is so that the ROM routines know where to find it. This location can be obtained from the 'address' routine.

USING THE ROUTINES FROM BASIC

The routines are called 'add', 'sub', 'mul','div' and 'address'. It is a simple matter to call these from Basic (as a preliminary to final coding in assembler, for example):

```
CALL div,NUM,DEN,QUO%
```

is equivalent to:

```
QUO%=NUM/DEN
```

```
CALL add,A,B,C
```

is equivalent to:

```
C=A+B
```

The address of C(I%) can be found by:

```
CALL address,C(I%),ADR
```

VARIABLE POINTERS

The last routine, 'address', is a useful routine, for Basic, in its own right. It can be used to simulate the function, VARPTR, found in versions of Basic on some other computers.

If you find yourself trying to convert a program from another machine that contains a line such as:

```
AD=VARPTR(V)
```

This can be replaced with a call to the address routine of the form:

```
CALL address,V,AD
```

USING THE ROUTINES FROM ASSEMBLER

To use the routines from an assembler program you must imitate the

action of the CALL statement, setting up the parameter block that points to the memory locations of all the variables used in the calculation. The parameter block always starts at address &600 with a format as given on page 214 of the User Guide. To set up the block you need to know the addresses of the variables. Although the system integers (@%, A%-Z%) are always stored, in order, in locations &404 to &468, to have easy access to any other variables, you should use the address routine as follows:

1. Assemble the 'address' and arithmetic routines (lines 1350 to 2250).
2. Use 'address' to find the addresses of the variables declared in the Basic part of the program.
3. Assign these addresses to (other) Basic variables.
4. Use these latest variables as symbolic addresses in the main assembler language part of the program and as variable address pointers when accessing the ROM routines.

For example:

A section of Basic that is to be converted to assembler uses the variables X and Y which are to be multiplied, and the result put into the integer variable Z%, i.e. Z%=X*Y.

1. At the start of the program declare the variables (eg X=4) and use 'address' to find the location of the variables needed, storing these as, say, A%, B%, and C%:

```
CALL address,X,A%
CALL address,Y,B%
CALL address,Z%,C%
```

2. In the assembler part when the calculation is needed, create a parameter block using these addresses:

```
LDA #A% MOD 256:STA &601  \ X L byte
LDA #A% DIV 256:STA &602  \ X H byte
LDA #5:STA &603  \ X variable type 5
LDA #B% MOD 256:STA &604  \ Y L byte
LDA #B% DIV 256:STA &605  \ Y H byte
LDA #5:STA &606  \ Y variable type 5
LDA #C% MOD 256:STA &607 \ Z% L byte
LDA #C% DIV 256:STA &608 \ Z% H byte
LDA #4:STA &609 \ Z% variable type 4
```

(Variable type 5 denotes a floating point variable while 4 is for an integer variable. See the User Guide page 215)

3. Call 'mul' routine (which will use the parameter block):

```
JSR mul
```

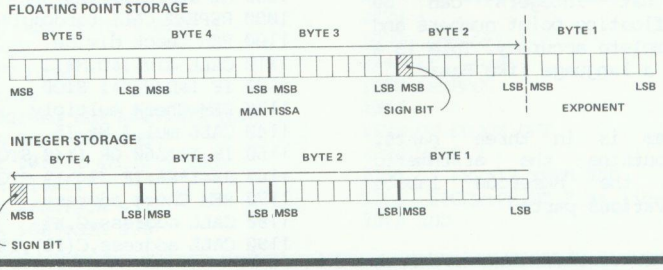
4. The last two steps are the assembler equivalent of 'CALL mul' in Basic. The result will be in the Basic variable, C%. This result is pointed to (as before) by the address at &607 and &608 ready for use.

VARIABLE STORAGE

In order to manipulate numbers in assembler language in this way you need to know how the numbers that are to be used in the calculations are stored by Basic so that you can imitate this for the benefit of the ROM routines.

Integers are easy. These are 32-bit quantities stored with the least significant byte at the lowest address. The most significant bit is the sign bit. If the sign bit is set (negative) then the number is the value of the remaining 31 bits minus (2 ^ 31)

If you are uncertain of this method of storing integers (known as two's complement) you can experiment by assigning numbers to A%, and examining the locations where this variable is stored using, say,



```
PRINT !&404' ?&407 ?&406 ?&405 ?&404
```

Some good numbers to try are 1, 2, -1, -2, 2147483647, -2147483648, 0.

Floating point numbers are a little more complicated and need five bytes of memory. The first (lowest address) byte is an exponent (in a format known as excess 128 notation), and the next four bytes form a 32-bit integer. This time the least significant byte is at the highest address.

The sign of the integer part also works differently. If the sign bit is set (1) then ALL the bits, including the sign bit, represent negative quantities (ignoring, for the moment, the exponent). If the sign bit is zero then all the bits represent positive quantities, but you must also add on

```
2 ^ 31 (=2147483648)
```

Once you have arrived at the value of the integer part, the value of the complete number is arrived at by multiplying by:

```
2 ^ (exponent - 128 - 32)
```

where the exponent is treated as a signed 8-bit number. Here is a simple example.

```
A=96
CALL address,A,AD%
PRINT ?AD%        :REM gives &87
PRINT ?(AD%+1)    :REM gives &40
```

As the remaining bytes are zero in this example the value of the integer part is:

```
2^30 + 2^31.
```

This must be multiplied by:

```
2^(&87 - 128 - 32)
```

to get the result 96. Try it for yourself.

So 96 is stored as &87, &40, &00, &00, &00, and -96 is stored as &87, &C0, &00, &00, &00.

It is worth noting that floating point numbers maintain 32-bit accuracy. This means that integers can be converted to floating point numbers and back with complete accuracy. This is a great help in a language like Basic.

PROGRAM NOTES

The program is in three parts. PROCassem contains the arithmetic routines and the numerous labels identify the various parts.

PROCbasic1 and PROCbasic2 set up the entry points to the Basic ROM for the two versions of the language.

Although the program can detect which version of Basic you have in your BBC micro, it can be shortened by leaving out the procedure not relevant to your Basic and altering line 1660 to call just the procedure that you need.

PROCtest performs a test, from Basic, on each of the routines to check that you have entered them correctly. This is performed with random variable values repeatedly for 100 times. The test number is printed out as this is being done. The program stops if any of the tests fail.

Once the routines have been checked in this way, this part of the program does not have to be included in your assembler program.

```
  10 REM Program Assembler Arithmetic
  20 REM Version B0.1
  30 REM Author  Derek Chown
  40 REM BEEBUG  Jan/Feb 1985
  50 REM Program subject to copyright
  60 :
 100 ON ERROR GOTO 2270
 110 :
 120 MODE 7
 130 DIM C(360)
 140 PROCassem
 150 PROCtest
 160 END
 170 :
1000 DEF PROCtest
1010 PRINT:PRINT"TESTING";
1020 I=RND(-TIME)
1030 C=0
1040 FOR count%=1 TO 100
1050 REM Check address of integers
1060 CALL address,Z%,A%
1070 IF A%<>&468 STOP
1080 M%=&7FFFFFFF:N%=360
1090 REPEAT CALL random:UNTIL !&D>=0
1100 REM Check divide
1110 CALL div,!&D,M%,I
1120 IF !&D/M%<>I STOP
1130 REM Check multiply
1140 CALL mul,I,N%,I%
1150 IF I%>360 OR I%<0 STOP
1160 J%=I*N%:IF J%<>I% STOP
1170 REM Check address
1180 CALL address,C,X%
1190 CALL address,C(I%),Y%
```

```
1200 C(I%)=COS(RAD(I%))
1210 !(X%+1)=!(Y%+1):?X%=?Y%
1220 IF C<>COS(RAD(I%)) STOP
1230 REM Check addition
1240 A%=RND:B=RND:CALL add,A%,B,C
1250 IF A%+B<>C STOP
1260 REM Check subtraction
1270 A=RND*RND/RND:B=A-RND
1280 CALL sub,A,B,C%
1290 D%=A-B:IF D%<>C% STOP
1300 PRINT count%;
1310 NEXT count%
1320 PRINT
1330 ENDPROC
1340 :
1350 DEF PROCbasic1
1360 REM Name some useful addresses
1370 REM within the BASIC ROM
1380 additi=&A50E
1390 divisi=&A6F2
1400 fac1=&30
1410 fac2=&3D
1420 floati=&A2AF
1430 fixedp=&A3F2
1440 multip=&A661
1450 random=&AF78
1460 setup1=&B35B
1470 transf=&A20F
1480 subtra=&A505
1490 ENDPROC
1500 :
1510 DEF PROCbasic2
1520 additi=&A500
1530 divisi=&A6E7
1540 fac1=&30
1550 fac2=&3D
1560 floati=&A2BE
1570 fixedp=&A3E4
1580 multip=&A656
1590 random=&AF49
1600 setup1=&B32C
1610 transf=&A21E
1620 subtra=&A4D0
1630 ENDPROC
1640 :
1650 DEF PROCassem
1660 IF ?&8015=&31 PROCbasic1 ELSEIF ?
&8015=&32 PROCbasic2 ELSE PRINT"Incompa
tible basic":END
1670 S%=&F3:DIM Q% S%
1680 FOR pass=0 TO 3 STEP 3
1690 P%=Q%
1700 [OPT pass
1710 .start
1720 .address LDA &601:STA &2A
1730 LDA &602:STA &2B
1740 LDY #0:STY &2C:STY &2D
1750 JSR floati
1760 LDA &604:STA &37
1770 LDA &605:STA &38
1780 LDA &606:JSR assb
1790 RTS
1800 :
1810 .set LDA &601,Y:STA &2A
1820 LDA &602,Y:STA &2B
1830 LDA &603,Y:STA &2C
1840 PHA:TYA:PHA
1850 JSR setup1
1860 PLA:TAY:PLA
1870 CMP #4:BNE tra
1880 TYA:PHA:JSR floati:PLA:TAY
1890 .tra CPY #3:BNE eset
1900 JSR transf:.eset
1910 RTS
1920 :
1930 .prep LDY #3:JSR set
1940 LDY #0:JSR set:RTS
1950 :
1960 .ass LDA &607:STA &37
1970 LDA &608:STA &38
1980 LDA &609
1990 .assb CMP #4:BEQ ias
2000 JSR sig1:LDY #4
2010 .flp LDA fac1,Y:STA (&37),Y
2020 DEY:BPL flp:JMP eas
2030 .ias JSR fixedp:LDY #3
2040 .ilp LDA &2A,Y:STA (&37),Y
2050 DEY:BPL ilp
2060 .eas RTS
2070 :
2080 .sig1 LDA #&80:EOR fac1-2:AND #&8
0:EOR fac1+1:STA fac1+1:RTS
2090 .sig2 LDA #&80:EOR fac2-2:AND #&8
0:EOR fac2+1:STA fac2+1:RTS
2100 :
2110 .div JSR prep:LDA #fac2:STA &4B:L
DA #0:STA &4C
2120 JSR sig2:JSR divisi:JSR ass:RTS
2130 :
2140 .mul JSR prep:LDA #fac2:STA &4B:L
DA #0:STA &4C
2150 JSR sig2:JSR multip:JSR ass:RTS
2160 :
2170 .sub JSR prep:LDA #fac2:STA &4B:L
DA #0:STA &4C
2180 JSR sig2:JSR subtra:JSR ass:RTS
2190
2200 .add JSR prep:LDA #fac2:STA &4B:L
DA #0:STA &4C
2210 JSR sig2:JSR additi:JSR ass:RTS
2220 :
2230 .end RTS:]
2240 NEXT pass
2250 ENDPROC
2260 :
2270 ON ERROR OFF
2280 MODE 7
2290 IF ERR=17 THEN END
2300 REPORT:PRINT " at line ";ERL
2310 END
```

THREE SPEECH SYSTEMS REVIEWED

There are two main ways by which speech can be simulated on the micro. Words spoken by a real human voice can be recorded, turned into digital form and stored on a chip. Alternatively, the small units of sound (allophones) that go together to make spoken words can be produced artificially and put together to make intelligible speech. Whilst the first method produces a more natural result, the number of words available is limited to those recorded, and various combinations of them. The other method, using allophones, is much more flexible, as almost any word can be created, but this flexibility is at the price of losing the human quality of the speech. This month we look at three of the speech systems on offer.

```
Product  : Cheetah Sweet Talker speech
           synthesizer board
Price    : £24.95 inc.VAT
Supplier : Cheetah Marketing Ltd.,
           24 Ray Street,
           London EC1R 3DJ
Reviewer : Ernest Bebbington
```

The Cheetah Sweet Talker uses the allophone method. It is a small circuit board holding a speech synthesizer chip and a few other components. It comes well protected in a box with demonstration software on cassette and an instruction leaflet.

Installation is merely a matter of inserting the circuit board into the Beeb's speech socket. The 'Beebtalk' tape, included with the unit, gives a demonstration of the Sweet Talker's capabilities. As the words are spoken through the Beeb's loudspeaker the program prints them on the screen. To test the clarity of the speech I tried not looking at the monitor, and I had no trouble in understanding what was said, in spite of the monotonic delivery. The second part of the demonstration program tells the user how to write software for the unit and possible uses for speech synthesis are also mentioned. There is a brief demonstration of its use in a game and a rather weak attempt at producing foreign words: no Frenchman would be fooled by the Sweet Talker's version of 'bonjour'.

In order for the Sweet Talker to speak, a short assembly language program, (printed in the instruction leaflet) must be incorporated into any Basic program using it. This is just a few lines and can be in the form of a procedure invoked near the beginning of a program to assemble it into memory. The resulting machine code (about 20 bytes) is then CALLed from Basic when required.

Each word spoken by Sweet Talker is split up into a series of allophones (of which a complete list is supplied), and it is very easy to make the unit say any one of them. There are even set length pauses built in to aid the user. I'm not convinced as to the completeness of the set of allophones though; I couldn't find one for the 'oo' of 'move'.

One major drawback to the Sweet Talker is the Dalek-like enunciation. This is acceptable in games, where short phrases are spoken, but in more serious software, where longer periods of speech may be needed, the monotony could be unpleasant. However, because of the flexibility of the allophone system, the relatively low price of the unit and the ease of writing its software, I recommend Sweet Talker as a good, no-frills introduction to speech synthesis on a microcomputer.

```
Product  : Easytalk Speech Utility ROM
Price    : £21.95 inc.VAT
Supplier : Galaxy Software,
           123 Links Drive,
           Solihull,
           West Midlands B91 2DJ
Reviewer : Ernest Bebbington
```

The Easytalk Speech Utility ROM is not a speech synthesizer in itself; more a powerful tool for the Acorn Speech System unit for the BBC micro. Consisting of just an EPROM, and a twenty three page booklet of instructions, the Easytalk system fits easily into a vacant ROM socket on your

Beeb. The computer must, of course, be already fitted with the Acorn Speech interface (reviewed in BEEBUG Vol.2 No.3).

Basically, what Easytalk does for the speech upgrade is to allow much easier access to the words and sounds already available, to let users create their own words from a given set of phonemes and to design their own custom built phonemes.

To access the pre-set words and sounds in the Acorn Speech interface, Easytalk provides several easy-to-use commands. *VOCAB followed by a word or phrase causes the words to be spoken without having to look up the codes. *NUMBER x, where x is any real number between -999999 and +999999, speaks the number in the correct way and not just as a string of digits. Similarly, *AMOUNT can be used to pronounce amounts of money in pounds or dollars, *TIME and *CLOCK turn the Beeb into a speaking clock (with 24 to 12 hour conversion) and *ALARM will sound a bleep at a specified time. I found that all of these commands worked correctly and were easy and straightforward to use.

The *SAY command, in conjunction with a set of phonemes, can be used to make the computer speak almost any word. For example, *SAY Y.U:N.I.V.ER.S will pronounce the word 'universe'. There are enough phonemes to provide intelligible speech with a more human quality than with a normal phoneme-type speech synthesizer. *SING is similar to *SAY except that it voices each sound at one of sixty-four specified pitches. I tried to make Easytalk sing a simple song, but the pitches available were not close together enough to produce a very musical result.

The cost of the Easytalk ROM does not seem excessive in the light of the facilities it provides. It gives the user of the Acorn Speech system a much easier way of getting at the prescribed words and, by the use of phonemes, banishes the frustration of being stuck with a limited set of words and sounds. It cuts out the complex programming needed to make the computer speak numbers, prices and times and it allows the creation of phonemes and sounds not normally available. The instruction manual is a paragon of clarity and completeness. My only frustration with Easytalk was that it would not work on my Beeb, but this may have been the fault of my Microware DFS, so I had to borrow a friend's machine to try it out. The existence of Easytalk may give people who are thinking twice about buying the Acorn Speech system the help they need in making their decision.

Product	: Beeb Speak
Price	: £25.30 inc. VAT and P&P.
Supplier	: BASYS, 48 Sundridge Drive, Walderslade, Chatham, Kent ME5 8HT Tel. 0634 660157
Reviewer	: Tim Hill

Beeb Speak, like the Cheetah Sweet Talker, uses allophones allowing nearly any word to be spoken. The system comprises a small circuit board which plugs into the speech socket. Instructions are included of course, and the board is said not to clash with a double density disc controller (if fitted), nor does the unit use any RAM etc. Although the unit makes use of the micro's own amplification circuit it does not interfere with SOUND or ENVELOPE instructions at all. The machine can make sounds and talk at the same time!

Using the device is simplicity itself. There are a couple of programs supplied with Beeb Speak which demonstrate its capabilities and the software which drives the unit is only 22 bytes of machine code which can be incorporated in a user's program and assembled to anywhere in memory. To make the device speak, an allophone is passed to the machine code using A% and then the code is CALLed. One small point is that any speech so generated must be terminated by a 'silence' - allophone 0 is useful for this - as otherwise the last allophone will continue until the machine is switched off (not even Break will help).

All in all, the Beeb Speak is simple and great fun to use. The only drawback is that a few words, particularly short ones such as "man", are a little unintelligble. Like Sweet Talker, this product can be recommended.

A SPLIT SCREEN UTILITY

by A. Nicol

One of the characteristics of many of the new machines appearing on the market today (e.g. the Apple Macintosh) is their use of multiple windows to help separate information. This utility will allow the Beeb's screen to be divided into two separate and independent windows, most useful for program development and many other applications.

The program here is a short machine code utility that provides the user with two windows in any mode, and allows for easy toggling (or switching) between the two at any time. A full record of all text and graphic details is maintained for both windows, allowing for different text colours to be set up for the two windows, and then for the windows to be switched, maintaining the correct colours for each.

Using the utility is very easy. First, type in the program, and save it away to tape or disc BEFORE running it, as it modifies itself when run. Once run, the program will report that the split screens are installed. At this point, PAGE will have been raised by &400 to accommodate the machine code and the data necessary for its correct operation. You should not alter the value of PAGE with the split screen in operation, or the machine may crash with a loss of program and data.

To use the split screen facility the two windows must first be initialised. This is accomplished by means of the *LINE command, which may be issued from either command mode, or from within a Basic program. Once ready, the toggling is performed by means of the *CODE command, which may also be issued from either command mode, or from within a program. Changing mode clears the split screens, and you will need to use *LINE again before you can swap between the two screens.

As it stands, the program locates the machine code at the default value of PAGE, and then increases PAGE by &400 to cater for this. Pressing Break will not affect the code as it intercepts this, and re-installs itself. If the code becomes corrupted however, the machine is likely to 'hang'. You can arrange for the code to avoid resetting PAGE by deleting lines

```
L.,999
  10REM PROGRAM HOBBREAK
  20REM VERSION B0.3  05/11/84
  30REM AUTHOR  T.POWYS-LYBBE
  40REM BEEBUG  JAN/FEB 1985
  50REM PROGRAM SUBJECT TO COPYRIGHT
  60:
 100PROCAssemble
 110CALLDisable
 120MODE7
 130PRINT"This proves that it works..
"''"did you hear the bleep every time t
he""little darlings pressed BREAK or"'
"CTRL BREAK or even ESCAPE, CTRL and""
BREAK?"
 140END
 150:
 160REM Put the rest of your program h
ere
 170:
>]*CODE
```

This small routine will enable you to stop people from breaking into your program. The idea is a simple one, based on the re-programming of function key 10 (the Break key). The machine code routine sets up this key so that every time Break is pressed, it reproduces OLD followed by RUN. It is not possible to totally inhibit the normal Break function but it is quite possible to keep control within the current program.

Another potential interruption to the running of a program is the use of the Escape key. Although it would appear that Escape can be easily disabled, the combination of Escape with Ctrl Break is very powerful and failed to protect the original program (lines 30 to 50 below). So the Escape disable had to be included in the machine code at line 10060.

100 to 130, and replacing this with:

```
100 A%=&900
```

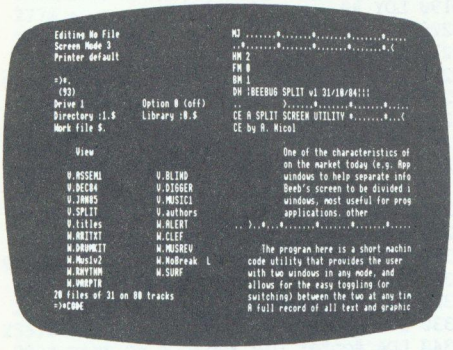
This uses memory that is normally allocated to cassette, speech, extra envelopes, RS423, user defined keys and user defined characters. To stop it intercepting Break, delete lines 2090 to 2200.

Split screens have a variety of uses, and you are likely to find yourself using them quite frequently once you are familiar with them. Split screens are particularly helpful in program development. If you get an error, you can leave the current error message on one screen, and switch over to the other and edit the program. You can't use split screens with Wordwise, but you can, if you alter the program slightly, do so with View; change all memory references from &8x to &9x (easy with BEEBUGSOFT's Toolkit), and re-run the program. Then enter View (via *WORD) and go to mode 1; enter Ctrl-V followed by 0, and a *LINE. Then use *CODE to toggle between the two screens. This is useful for looking at two separate pieces of text at the same

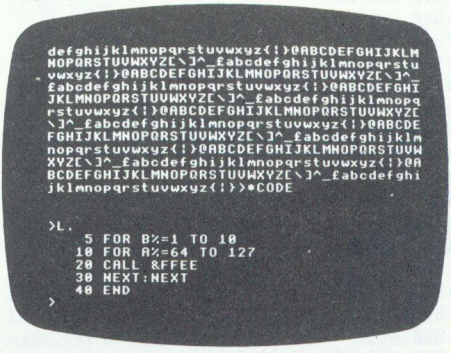
time. Note that you cannot have both Basic and View 'active' at the same time.

PROGRAM NOTES

Lines 100 to 130 of the program define a function key to move the program up in memory by &400 bytes. This is to allow room for the machine code, and these lines should be kept in if the machine code is to reside at the default value of PAGE. The use of a function key to effect this movement is because this provides the easiest way to execute a piece of code once, and then to delete it from a program.




```
  10 REM PROGRAM SPLIT
  20 REM AUTHOR A. NICOL
  30 REM VERSION B.02
  40 REM BEEBUG JANUARY/FEBRUARY 1985
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 *FX18
 110 *KEY0MODE7|MDELETE0,199|MA%=PAGE:
B%=TOP-PAGE|MFORI%=0TO B%STEP4:I%!&3000=
I%!PAGE:N.|MFORI%=0TOB%STEP4:I%!(A%+&40
0)=I%!&3000:N.|MPAGE=A%+&400|MEND|MLOME
M=TOP|MRUN|M
 120 *FX138,0,128
 130 END
 140 :
 200 DIM space 10
 210 oswrch=&FFEE
 220 osbyte=&FFF4
 230 userv=&200
 240 mstart=A%
 250 block=mstart:blocka=mstart+&A0
 260 datapnt=mstart+&C0
 270 start=mstart+&100
 280 PROCassemble
 290 CALL run
 291 PRINT '""Split screens installed.
"''
 300 END
 310 :
1000 DEF PROCassemble
1010 FOR I%=0 TO 3 STEP 3
1020 P%=start
1030 RESTORE
1040 FOR K%=0 TO 14 STEP2
1050 datapnt!K%=P%
1060 FOR J%=0 TO 37
1070 READ J%?P%
1080 NEXT
1090 P%=P%+J%
1100 NEXT
1110 [OPT I%
1120 .enter
1130 CMP #1       \test for *line entry
1140 BEQ setup
1150 CMP #0       \test for *code entry
1160 BEQ swapscr
1170 BRK
1180 ]
1190 ?P%=128:P%=P%+1
1200 $P%="Incorrect entry"
1210 P%=P%+LEN$P%
1220 [OPT I%
1230 BRK
```




```
1240 .setup
1250 LDA &355
1260 CMP #8
1270 BCC validmode
1280 BRK
1290 ]
1300 ?P%=128:P%=P%+1
1310 $P%="Invalid mode"
1320 P%=P%+LEN$P%
1330 [OPT I%
1340 BRK
1350 .validmode
1360 LDA #22
1370 JSR oswrch
1380 LDA &355
1390 JSR oswrch
1400 LDA &355
1410 ASL A
1420 TAX
```

```
1430 LDA datapnt,X
1440 STA &85
1450 LDA datapnt+1,X
1460 STA &86
1470 JSR window
1480 LDY #0
1490 .loop1
1500 LDA &300,Y
1510 STA block,Y
1520 INY
1530 BPL loop1
1540 LDY #0
1550 .loop1a
1560 LDA &D1,Y
1570 STA block1,Y
1580 INY
1590 CPY #&F
1600 BNE loop1a
1610 CLC
1620 LDA &85
1630 ADC #&13
1640 STA &85
1650 LDA &86
1660 ADC #0
1670 STA &86
1680 .window
1690 LDY #0
1700 .loop2
1710 LDA (&85),Y
1720 JSR oswrch
1730 INY
1740 CPY #&13
1750 BNE loop2
1760 RTS
1770 .swapscr
1780 LDA #8
1790 BIT &D0
1800 BNE splitset
1810 BRK
1820 ]
1830 ?P%=128:P%=P%+1
1840 $P%="Split screens not set up"
1850 P%=P%+LEN$P%
1860 [OPT I%
1870 BRK
1880 .splitset
1890 LDY #0
1900 .loop
1910 LDA &300,Y
1920 LDX block,Y
1930 STA block,Y
1940 TXA
1950 STA &300,Y
1960 INY
1970 BPL loop
1980 LDY #0
1990 .loop1
2000 LDA &D1,Y
2010 LDX block1,Y
2020 STA block1,Y
2030 STX &D1,Y
2040 INY
2050 CPY #&F
2060 BNE loop1
2070 RTS
2080 .run
2090 LDA #247
2100 LDX #76
2110 LDY #0
2120 JSR osbyte
2130 LDA #248
2140 LDX #break MOD 256
2150 LDY #0
2160 JSR osbyte
2170 LDA #249
2180 LDX #break DIV 256
2190 LDY #0
2200 JSR osbyte
2210 .brkpnt
2220 LDA #&0D            \new program
2230 STA PAGE
2240 LDA #&FF
2250 STA PAGE+1
2260 LDA #PAGE DIV 256   \change page
2270 STA 1
2280 STA 3
2290 STA &13
2300 STA &18
2310 STA &1D
2320 LDA #enter MOD 256
2330 STA userv
2340 LDA #enter DIV 256
2350 STA userv+1
2360 RTS
2370 .break
2380 BCC break1
2390 RTS
2400 .break1
2410 LDA #0
2420 STA &8F
2430 LDA &210
2440 STA &8A
2450 LDA &211
2460 STA &8B
2470 LDA #break2 MOD 256
2480 STA &210
2490 LDA #break2 DIV 256
2500 STA &211
2510 RTS
2520 .break2
2530 LDX &8F
2540 INC &8F
2550 LDA call,X
2560 CMP #32
2570 BNE end
2580 LDA &8A
2590 STA &210
2600 LDA &8B
2610 STA &211
2620 LDA #&D
```

```
2630 .end
2640 CLC
2650 RTS
2660 .call
2670 ]
2680 NEXT
2690 $call=CHR$11+CHR$152+CHR$9+CHR$12
7+"CALL"+STR$(brkpnt)+" "
2700 ENDPROC
2710 :
2720 REM data for mode 0
2730 DATA 24,0,0,0,0,128,2,255,3,29,0,
0,0,0
2740 DATA 28,0,31,39,0
2750 DATA 24,129,2,0,0,255,4,255,3,29,
129,2,0,0
2760 DATA 28,40,31,79,0
2770 :
2780 REM data for mode 1
2790 DATA 24,0,0,0,2,255,4,255,3,29,0,
0,0,2
2800 DATA 28,0,15,39,0
2810 DATA 29,0,0,0,0,24,0,0,0,0,255,4,
255,1
2820 DATA 28,0,31,39,16
2830 :
2840 REM data for mode 2
2850 DATA 24,0,0,0,2,255,4,255,3,29,0,
0,0,2
2860 DATA 28,0,15,19,0
2870 DATA 29,0,0,0,0,24,0,0,0,0,255,4,
255,1
2880 DATA 28,0,31,19,16
2890 :
2900 REM data for mode 3
2910 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2920 DATA 28,0,24,39,0
2930 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2940 DATA 28,40,24,79,0
2950 :
2960 REM data for mode 4
2970 DATA 24,0,0,0,2,255,4,255,3,29,0,
0,0,2
2980 DATA 28,0,15,39,0
2990 DATA 29,0,0,0,0,24,0,0,0,0,255,4,
255,1
3000 DATA 28,0,31,39,16
3010 :
3020 REM data for mode 5
3030 DATA 24,0,0,0,2,255,4,255,3,29,0,
0,0,2
3040 DATA 28,0,15,19,0
3050 DATA 29,0,0,0,0,24,0,0,0,0,255,4,
255,1
3060 DATA 28,0,31,19,16
3070 :
3080 REM data for mode 6
3090 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3100 DATA 28,0,12,39,0
3110 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3120 DATA 28,0,24,39,13
3130 :
3140 REM data for mode 7
3150 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3160 DATA 28,0,12,39,0
3170 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3180 DATA 28,0,24,39,13
```

NEWS NEWS NEWS

WATFORD QUALITY

Yet another ROM from Watford Electronics promises to make your Epson FX or RX 80 printer act like the very popular Kaga and Cannon 'near letter quality' printers. The latter have the useful facility to print in an excellent typeface similar to a typewriter or daisy wheel printer. The Watford ROM uses the graphics facility of the less sophisticated, but even more popular, Epsons to create this near letter quality printout. The NLQ ROM costs £18.40 (incl. VAT). Further details, and maybe, if you ask nicely, a sample printout from Watford on 0923-40588.

CHEAP COMMS

The price barrier of communications has been soundly broken by the Unicomm modem from the company of the same name. Using established technology and expecting high sales has enabled Unicomm to offer its all singing all dancing modem for only £60. When combined with a BBC micro ROM (another £24) this unit will support full auto dial, auto answer, number store, auto baud rate scan, and many other advanced features. Unicomm is on 01-482 1711.

EAGER BEAVER

The Beaver plotter from Linear Graphics offers your BBC micro unsurpassed graphics hard copy for the remarkable price of £516 (incl. VAT). The Beaver is a flat bed plotter and will take paper up to A4 in size. Two pens are held in the plotter at any one time, though these can be easily changed. Standard roller ball or felt tip pens are used. The plotter comes complete with software to intercept all screen graphics commands and reproduce them on paper. Linear Graphics can be contacted on 0286-741322.

THREE MUSIC PROGRAMS

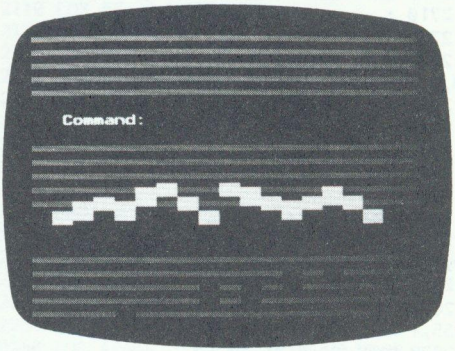
Reviewed by E. D. Bebbington

The complexities of using the BBC micro's sound commands for music deter all but the most ardent virtuoso. Ernest Bebbington looks at a handful of established packages that help the struggling composer.

```
Title   : Music
Supplier: BBC Soft (01-580 5577)
Price   : £10.00
Rating  : *

Title   : The Synth
Supplier: Musicsoft (0525-402701)
Price   : £8.00
Rating  : **

Title   : Music Processor
Supplier: Quicksilva (0703-20169)
Price   : £14.95
Rating  : ****
```



MUSIC

BBC Soft's Music package has as its best feature a very attractive display. Notes are shown in graphic form as horizontal bars overlayed onto a music stave of five lines. When recording a melody, the computer keyboard is used as an instrumental keyboard, with the keys used as though they were piano keys. The longer a key is pressed, the longer becomes a horizontal bar at the appropriate place on the screen stave, so far, rather like the 'Piano' program on the Welcome tape that came free with your BBC micro.

However, you can edit the music by moving a T-shaped cursor along the stave. Notes can be inserted, deleted or changed. As the music is played back, the cursor moves along the three staves and they scroll sideways when it hits the right edge of the screen, thus giving an interesting visual image of the music. Unfortunately only four pre-set envelopes are available and there are no commands for altering them to your own ideas. To my ears these envelopes lacked the subtlety that is possible when defining your own.

The BBC Music program is the simplest of the three packages reviewed here. I would recommend this program only to those who want a music program for fun and who do not want to be bothered by too much detail. The graphic representation of the music provides an interesting diversion.

THE SYNTH

The Synth by Musicsoft is much more sophisticated. In addition it has one very useful feature: the music can be entered in free rhythm and then put into rhythm by tapping a single key. This means that you don't have to be adept at using a keyboard, and that fast and complex melodies can be entered with ease.

After entering and putting into rhythm the notes of one voice, the next voice can be put into rhythm while the previous one is being played back, allowing the timing of the voices to be matched. It is necessary to record several dummy notes at the beginning of the first recorded voice so that the first notes of each voice can be aligned. There is a 'tidy' option which is supposed to correct small discrepancies of alignment. However, this altered the rhythm too much, resulting in music which did not have a regular pulse.

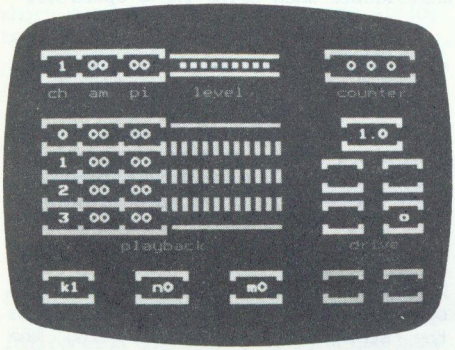
The Synth provides three music channels and one noise channel. There are four octaves and sixteen pre-set envelopes which can be changed easily. There is enough memory to enter up to

3,000 notes spread amongst the four voices. Two rows of keys of the computer keyboard are used to input the notes and four self-adhesive stickers are thoughtfully provided to mark the keys in the top row which are the gaps between the groups of 'black' notes.

The idea behind The Synth is a good one and allows much flexibility and ease of entering the music, but the results were not always satisfactory. It is very tricky to get the voices into line with each other no matter how carefully you tap out the rhythm. The documentation could also be improved. However, The Synth is a well thought-out program, even to the point of providing a means of automatically copying the cassette to disc - an all too rare benefit!

MUSIC PROCESSOR

Quicksilva's Music Processor (Muproc) adopts an interesting approach. It pretends to be a synthesizer, a tape recorder and an editing desk all in one. You are confronted by a complicated screen of



coloured panels, each of which contains some information about channels, envelopes, playback speed and so on. The computer keyboard becomes a piano-style keyboard, all four rows of keys being used. Six octaves and four channels are available.

There are so many facilities available with Muproc that I can mention only a selection. Once all the commands are familiar, Muproc proves to be easy to use and very flexible: envelopes can be specified using only five parameters and the computer can even be 'fine-tuned' to allow playing with other instruments. When the music is recorded, one voice at a time, the envelope and amplitude can be altered for each note. On playback all the information about the notes is displayed on the screen panels and a counter, like the one on a tape recorder, is displayed. The music can be wound forward and back at any speed (up to sixty notes per second) and single-stepped. Once edited, the notes recorded can be compressed, taking six bytes each instead of ten, allowing more notes to be added.

I found Muproc to be easy to control, but the use of the computer keyboard as an instrument was error-prone. The continuous display of information about the music on the screen was very useful. A very flexible program.

[There was to have been a fourth music package included in this review - 'Music Editor', from System Software. Although this was the most easy to use and gave the best results, it is unfortunately no longer available. The good news, however, is that System Software has not been idle but has just completed an improved and expanded version of Music Editor, called 'The Music System' and published by Island Logic. Although this package costs a very hefty £24.95 for the full disc version and £12.95 for the lesser cassette version, it offers a vast range of functions in an easy to use format. We hope to have a full review of The Music System for you soon - Ed.]

BEEBUGSOFT has also launched its own music system called MUROM.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

TUBE CORE SAVE

Users of the Tube who are running programs that take a long time to execute may like to use a *SAVE IMAGE 0 8000 (or B800 for Hi-Basic) to save the current variables in their program at intervals during execution in case of a power failure. If this is done, and then reloaded at a later date, then the program and variables will all be preserved, and an attempt can be made to salvage data from any disaster.

BEEBUG Workshop

Indirection Operators (Part 2)

by Surac

This month we conclude our two part series on the use of indirection operators with a look at their use in handling strings and arrays.

Although mentioned briefly at the start of last month's article, I have left the subject of the $ (dollar or string) indirection operator to now as the use of this operator is a little different to that of the other two.

As you may have guessed, the $ indirection operator (unlike ! and ?) deals with strings of characters of variable length, as opposed to a fixed number of bytes in memory. Note also that all indirection operators affect memory in the same way - it's just the interpretation put on the values that is different. In some ways the $ operator is not as flexible as normal string variables in Basic as it cannot cope in the same way with strings that have a Return (ASCII 13) in them. However, it does allow for the creation of some more flexible structures, and easier passing of strings to the operating system.

To get going, we'll try out a simple test program, and then go through it and examine it in detail. Type in the program below, run it, and observe the results.

```
 10 REM EXAM1
 20 DIM SPACE 256
 30 @%=5
 40 $SPACE="INDIRECTION OPERATORS"
 50 PRINT $SPACE
 60 FOR I%=0 TO LEN $SPACE
 70 PRINT SPACE?I%,;
 80 NEXT
 90 SPACE!4=&31323334
100 PRINT
110 PRINT $SPACE
```

When run, this program will print out a series of numbers which correspond to the ASCII values of the characters in the reserved area of memory (see page 486 onwards in the User Guide for a list of ASCII values). These will correspond to the characters assigned at line 40. Note the '13'; this corresponds to a Return, and is unavoidable when using the $ operator.

The reason for the Return is quite simple, and is due to the way in which the string is stored. If a string is to be stored flexibly, then its length is unlikely to be known in advance, and so some extra way of signifying this is needed. Normally strings stored by Basic have extra bytes associated with them that are used to determine their current and maximum length, but strings stored via indirection operators don't have any 'extra' such bytes, and so a specific character is used to terminate the string (this character being the Return character, with an ASCII value of 13). Whenever the $ indirection operator is used to assign a string to memory the last character will automatically be followed by ASCII 13 in memory.

Taking the above example program, we first allocate an area of memory in which to store our string (line 20), and alter the number formatting variable so that numbers are printed in a 'field' of five characters (line 30). Line 40 then stores a string of characters in memory ready for the program to access, and line 50

indicates the printing of a string from memory. Lines 60 to 80 then loop printing out the ASCII values of the characters present in memory, illustrating incidentally that the Return is included as part of the length. Line 90 then puts the ASCII value of four characters into the middle of the string, and the next two lines print a blank line and the string as it now stands. Note the use of the ? indirection operator that we discussed last month.

One use often made of $ is to provide a more flexible method of passing strings to the operating system (normally achieved in Basic with '*' commands, but made easier for Basic II users by the inclusion of the new command OSCLI). The short program below (although rather contrived) shows the basic format in which a key would be defined if its definition was not explicitly known before the program was run (as would be the case with the definitions in the format of *key ... Work through the program until you understand it, and then try to write a short procedure that passes any string to the operating system.

```
10 REM EXAM2
20 DIM KEY 40
30 X%=KEY AND 255
40 Y%=KEY DIV 256
50 FOR I%=0 TO 9
60 $KEY="KEY "+STR$I%+" "+STRING$(I
%,STR$I%)
70 CALL &FFF7 : REM OSCLI
80 PRINT $KEY
90 NEXT
```

! AND ? FOR ARRAYS

To complete our overview of indirection operators, we will now look at a way in which both ? and ! can be used to implement arrays which are faster and more flexible than normal Basic arrays (and in some cases more sparing on precious memory). The program below illustrates a one dimensional array in memory with each element only one byte in size (and hence only capable of holding values from 0 to 255). Note the different use of DIM in line 40 (see the User Guide page 237 for details on this).

As with the first example, @% is altered to suit the display produced by this example. N% is the number of elements minus one (the counting starts at zero). All the program does is to loop round inserting a random byte into each slot of the 'array'. Note that if all you need to store is one byte per element, then an array of the nature below will save on both memory AND execution time. Indirection operators are generally faster than most other techniques for achieving a given task; if you want a fast game, use them as much as possible!

```
10 REM EXAM3
20 @%=4
30 N%=99
40 DIM MEM% N%
50 FOR I%=0 TO N%
60 MEM%?I%=RND(255)
70 NEXT
80 FOR I%=0 TO N%
90 PRINT MEM%?I%,;
100 NEXT
```

Example 4 is a little more complex than the others listed here, and effectively simulates a 2 dimensional full integer array directly in memory. Note the necessity to add one on to the dimensions at line 40 to cater for the counting effectively starting at zero, and not one. The '*4' is to adjust the number of bytes allocated because we are dealing with integers, which use four bytes each.

```
10 REM EXAM4
20 @%=5
30 N%=15:M%=5
40 DIM MEM% (N%+1)*(M%+1)*4
50 FOR I%=0 TO N%*4 STEP 4
60 FOR J%=0 TO M%*4 STEP 4
70 MEM%!(I%*(N%+1)+J%)=I%+J%*256
80 NEXT,
90 FOR I%=0 TO N%*4 STEP 4
100 FOR J%=0 TO M%*4 STEP 4
110 PRINT MEM%!(I%*(N%+1)+J%),;
120 NEXT
130 PRINT
140 NEXT
```

That's all that we're going to do on indirection operators here, but keep reading other articles in BEEBUG to see actual applications that use indirection operators. Always bear in mind that they directly affect memory, and you can very easily corrupt your program if you use indirection operators carelessly.

QUICKSILVA'S DRUMKIT

Reviewed by Stephen Ibbs

In this review, Steve Ibbs looks at one of the more specialized music packages to be produced for the BBC micro and reports on his findings.

Title : Drum Kit
Supplier : Quicksilva
Price : £9.95
Rating : ****

Drum Kit is complementary to the earlier, acclaimed Music Processor from Quicksilva (see review elsewhere in this issue), though this is not at all essential to the use of Drum Kit. The program is supplied as a cassette with a seventeen page booklet well printed and set out. The cassette is designed for use with both tape and disc, and automatically adjusts the value of PAGE for disc systems.

After this short program the screen is filled with the QS logo, terrible music and noises, then the main program is loaded. The screen changes to show a moving dot at the top, representing where we are in the different bars, with four instruments underneath, "snare", "bass", "electro" and "sticks". The demonstration mode shows what type of sounds and rhythms are possible, and considering the limitations of the BBC sound chip, the effects are amazingly realistic.

Pressing any key stops the demonstration mode. Now new rhythm patterns can be entered extremely easily using the cursor and return keys. Entire lines of rhythm can be deleted as well as individual pulses, and the author is to be congratulated on the way he has simplified the manipulation of the patterns. Not many bars can be displayed on the screen at any one time, but there is a paging system whereby long rhythms can be entered, one screenful at a time.

Pressing the letter T enables pulses to be entered in 'real time', by tapping in the required rhythm from the keyboard, but this requires care and patience to get the pulses in exactly the right place. Pressing Escape changes the display to the 'Values' page where the variables 'tempo', 'beats to the bar', 'number of bars', and the 'start bar' can be adjusted. The last variable is a flexible means whereby sections of a long rhythm can be stored, repositioned, or edited at will.

In addition the Values page provides a metronome facility, which replaces the 'sticks' line in the rhythm display, and gives a background bleep at the start of each bar. When altering the timing of the other instruments the program prevents you from altering the metronome - a nice feature. The volume of each voice, metronome, and the accent available on each beat can be adjusted, and of course the rhythms can be stored on tape - as program data, not as audio!

From the values page we can go to an envelope editor. Even without the drum kit, this is a nice editor. Each component value of the SOUND and ENVELOPE commands can be adjusted, and pressing the space bar generates the sound. A 'scratchpad' memory is used to store the adjusted values, so that changes don't become permanent until confirmed.

So how good is it, and how useful? If your aim is to produce a synthesized drum machine effect for your band's latest 'demo' tape, then you are going to be very disapppointed. If, however, you have bought it so that you can experiment with the computers sound possibilities, and to generate and play around with rhythmical patterns, then this program is well worth buying. I have used it with groups of schoolchildren, and it has proved to be a useful teaching aid, because they can visually identify with the sounds they are hearing. Any mistakes stand out like a sore thumb on the display.

In conclusion I would recommend the program as a useful utility, that also happens to be fun!

DIGGER

by Andy Logan

Digger is an arcade style game based on the popular home micro game of 'Monsters' or 'Panic!'. It is a challenging one-player game in which you have to kill the gremlins before they kill you.

The game takes place in the familiar setting with various walkways made from brick with several different height ladders connecting each floor.

At the start you are chased around the screen by three gremlins which are lethal to the touch. To kill the gremlins you must dig a hole in the floor by pressing the space bar three times. When a gremlin falls into the hole you must hit it on the head with your shovel (by pressing the space bar again) so that it falls through to the next floor and dies. As you progress to the next skill level you must dig two holes, one directly underneath the other and drop the gremlin through both in order to kill it. On the third level the gremlins have to be dropped through three floors in a row and so on.

To make the game harder, you only have a limited amount of oxygen which slowly runs out while you play each level. Your remaining oxygen supply is displayed at the foot of the screen and if it runs out then you suffocate and the game ends. You start the game with three lives, and lose a life every time you are caught by a gremlin.

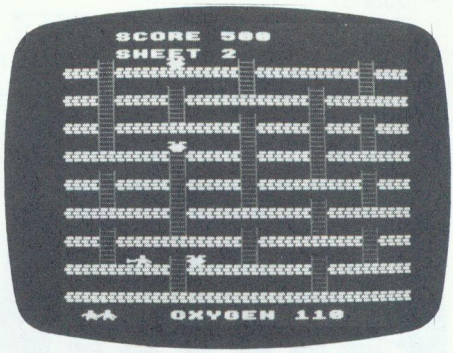
The keys to use for playing the game are 'Z' and 'X' for left and right and '*' and '?' for up and down. The spacebar is used to dig and to hit the gremlins on the head.



```
 10 REM PROGRAM DIGGER
 20 REM VERSION B0.2
 30 REM AUTHOR  ANDY LOGAN
 40 REM BEEBUG  JAN/FEB 1985
 50 REM PROGRAM SUBJECT TO COPYRIGHT
 60 :
100 ON ERROR GOTO 3190
110 MODE7:PROCtitle
120 MODE5:PROCstart
130 S%=0:U%=1:H%=3
140 PROCinit:PROCplatform
160 PROCladder:PROCsu
180 TIME=0:REPEAT
200 PROCm:PROCg
210 PROCscore
220 UNTIL DEAD% OR CO%=3
230 IF BON%<=0 GOTO260
240 IFDEAD% ANDH%<>0 CLEAR:CLS:GOTO140
250 IFCO%=3 U%=U%+1:CLEAR:CLS:GOTO140
260 COLOUR3:PRINTTAB(6,18)"GAME OVER"
:FORT=0TO3000:NEXT:*FX15
270 G=GET:CLEAR:CLS:GOTO 130
```

```
 280 END
 290 :
1000 DEFPROCtitle
1010 FORA=1 TO2:PRINTTAB(10,A)CHR$141
CHR$131 CHR$157 CHR$129;"D I G G E R";S
PC(2)CHR$156:NEXT:PRINTTAB(12)CHR$130"b
y Andy Logan"
1020 PRINT'TAB(1);CHR$134"Lure the GRE
MLINS into the holes that"'CHR$134"you
dig in the brick-work and knock"'CHR$13
4"them on the head.To kill a Gremlin yo
u"
1030 PRINTCHR$134"must knock it throug
h at least the"'CHR$134"sheet number of
 levels. Kill all three"'CHR$134"Gremli
ns before the oxygen runs out if"'CHR$1
34"you are to continue."'
1040 PRINT'CHR$133"Keys to use are:"'C
HR$130"Z - Left, X - Right, * - Up"'CHR
$130"? - Down, and SPACE to dig or knoc
k."
1050 PRINTTAB(12,22);CHR$131"PRESS SPA
CE":REPEAT:UNTIL INKEY-99:ENDPROC
1060 :
1070 DEF PROCinit
1080 IFU%=9 U%=1
1090 CO%=0:OXY%=50+(U%*30)
1100 IFU%>=4 OXY%=OXY%+150
1110 LA%=FALSE:STILL%=FALSE
1120 DIMA%(20,31),E%(5),F%(5),M$(3),PD
$(2),PRL$(2),B$(2),TRAP%(5),T%(5),DM%(5
),SG%(4)
1130 DEAD%=FALSE:R%=0
1140 FORI%=1 TO 3:M$(I%)=CHR$(I%+232):
NEXTI%
1150 L$=CHR$231:P$=CHR$230
1160 MD$=CHR$239:PRL$(1)=CHR$232:PRL$(
2)=CHR$237
1170 PD$(2)=CHR$238:PD$(1)=CHR$242
1180 G$=CHR$236:B$(1)=CHR$240:B$(2)=CH
R$241
1190 SPL$=CHR$243
1200 ENDPROC
1210 :
1220 DEFPROCstart
1230 VDU19,1,6,0,0,0
1240 VDU23,1,0;0;0;0;
1250 VDU23,230,119,119,0,238,238,0,119
,119
1260 VDU23,231,129,129,129,255,129,129
,129,255
1270 VDU23,232,0,0,0,4,7,4,0,0
1280 VDU23,233,24,24,0,124,190,25,36,34
1290 VDU23,234,24,24,0,62,125,152,36,68
1300 VDU23,235,90,90,66,126,126,36,36,
36
1310 VDU23,236,231,36,60,126,219,126,3
6,60
1320 VDU23,237,0,0,0,32,224,32,0,0
1330 VDU23,238,0,0,0,0,128,80,32,64
1340 VDU23,239,0,0,0,0,8,139,139,255
1350 VDU23,240,0,0,0,238,238,0,119,119
1360 VDU23,241,0,0,0,0,0,0,119,119
1370 VDU23,242,0,0,0,0,1,10,4,2
1380 VDU23,243,129,66,36,0,0,36,66,129
1390 A%=0:REM SET HI-SCORE
1400 ENVELOPE1,1,68,10,-127,240,113,14
,126,0,0,-126,126,126
1410 ENVELOPE2,0,80,-110,-50,159,250,1
90,126,0,0,-126,126,126
1420 ENVELOPE3,1,0,0,0,0,0,0,126,0,0,-
126,126,126
1430 ENDPROC
1440 :
1450 DEF PROCplatform
1460 COLOUR2:CLS
1470 FORJ%=5TO29 STEP3
1480 FOR I%=0TO 19
1490 PRINTTAB(I%,J%);P$;:A%(I%,J%)=-1
1500 NEXTI%
1510 NEXTJ%
1520 ENDPROC
1530 :
1540 DEF PROCladder
1550 C%=0
1560 COLOUR1
1570 FORJ%=4 TO 25 STEP3
1580 C%=C%+1
1590 IFC%=3 C%=1
1600 FOR L%=J% TO J%+3
1610 IFC%=1 PRINTTAB(2,L%);L$;:A%(2,L%
)=2:PRINTTAB(10,L%);L$;:A%(10,L%)=2:PRI
NTTAB(17,L%);L$;:A%(17,L%)=2
1620 IFC%=2 PRINTTAB(6,L%);L$;:A%(6,L%
)=2:PRINTTAB(14,L%);L$;:A%(14,L%)=2
1630 NEXTL%
1640 NEXTJ%
1650 FORI%=1TO8
1660 LX%=(RND(4)*4)-2:LY%=(RND(6)*3)+5
1670 FORJ%=LY% TOLY%+2
1680 PRINTTAB(LX%,J%);L$;:A%(LX%,J%)=2
1690 NEXTJ%:NEXTI%
1700 ENDPROC
1710 :
1720 DEF PROCm
1730 IF DEAD% ENDPROC
1740 N%=X%:M%=Y%:W%=B%:Q%=C%
1750 IFINKEY-73 Z%=3:Y%=Y%-1:GOTO1810
1760 IFINKEY-105 Z%=3:Y%=Y%+1:GOTO1810
1770 IFINKEY-98 Z%=1:X%=X%-1:GOTO1810
1780 IFINKEY-67 Z%=2:X%=X%+1:GOTO1810
1790 IFINKEY-99 PROCdig:ENDPROC
1800 STILL%=TRUE:ENDPROC
1810 STILL%=FALSE
1820 IFZ%=3 ANDA%(X%,Y%)<>2 X%=N%:Y%=M
%:ENDPROC
1830 IFA%(X%,Y%+1)=0 PROCfall:ENDPROC
1840 IFZ%=1 ANDX%<1 X%=1
1850 IFZ%=2 ANDX%>18 X%=18
1860 IFZ%<>3 ANDX%<>1 ANDX%<>18 SOUND1
,2,185,1
```

```
 1870 IFZ%=1 B%=X%-1 ELSEIFZ%=2 B%=X%+1
 1880 C%=Y%
 1890 PROCprint
 1900 IFA%(X%,Y%)=4 ORA%(X%,Y%)=5 ORA%(
X%,Y%+1)=6 ORA%(B%,C%)=4 ORA%(B%,C%)=5
PROCdead
 1910 ENDPROC
 1920 :
 1930 DEFPROCprint
 1940 PROCback(N%,M%)
 1950 PROCback(W%,Q%)
 1960 COLOUR3:PRINTTAB(X%,Y%);M$(Z%)
 1970 IFZ%=3 ORLA% ENDPROC
 1980 COLOUR2:PRINTTAB(B%,C%);PRL$(Z%)
 1990 ENDPROC
 2000 :
 2010 DEFPROCg
 2020 IFDEAD%ENDPROC
 2030 R%=R%+1:IFR%>3 R%=1
 2040 K%=E%(R%):L%=F%(R%)
 2050 IFDM%(R%)=TRUE:ENDPROC
 2060 IFNOTSTILL% SG%(R%)=FALSE
 2070 IFTRAP%(R%)ANDTIME-T%(R%)>300 TRA
P%(R%)=FALSE:A%(E%(R%),F%(R%))=-1:COLOU
R2:PRINTTAB(E%(R%),F%(R%));P$:F%(R%)=F%
(R%)-1:E%(R%)=E%(R%)+SGN(X%-K%):GOTO215
0 ELSEIFTRAP%(R%):ENDPROC
 2080 IFSG%(R%)GOTO2100
 2090 IFSGN(X%-K%)=0 ANDINT((L%-1)/3)=(
L%-1)/3 SG%(R%)=TRUE:IFRND(1)>.5 P%=1 E
LSEP%=-1
 2100 IFSG%(R%)ANDA%(K%,L%+SGN(Y%-L%))<
>2 PROCstill:GOTO2150 ELSEIFSG%(R%)SG%(
R%)=FALSE
 2110 IFL%>Y% ANDA%(K%,L%-1)=2 F%(R%)=F
%(R%)-1:GOTO2150
 2120 IFL%<Y% ANDA%(K%,L%+1)=2 F%(R%)=F
%(R%)+1:GOTO2150
 2130 IFINT((L%-1)/3)=(L%-1)/3 E%(R%)=E
%(R%)+SGN(X%-K%):GOTO2150
 2140 ENDPROC
 2150 IFA%(E%(R%),F%(R%))=4 ORA%(E%(R%)
,F%(R%))=5:E%(R%)=K%:F%(R%)=L%:PROChyp:
SOUND0,-15,200,1
 2160 IFE%(R%)<0 E%(R%)=0
 2170 IFE%(R%)>19 E%(R%)=19
 2180 IFA%(E%(R%),F%(R%)+1)=0 F%(R%)=F%
(R%)+1:A%(E%(R%),F%(R%))=6:TRAP%(R%)=TR
UE::T%(R%)=TIME
 2190 IFA%(E%(R%),F%(R%))=0 A%(E%(R%),F
%(R%))=4
 2200 IFA%(E%(R%),F%(R%))=2 A%(E%(R%),F
%(R%))=5
 2210 IFA%(K%,L%)=4 PRINTTAB(K%,L%);SPC
1:A%(K%,L%)=0
 2220 IFA%(K%,L%)=5:COLOUR1:PRINTTAB(K%
,L%);L$:A%(K%,L%)=2
 2230 COLOUR3:PRINTTAB(E%(R%),F%(R%));G$
 2240 IF(X%=E%(R%)ANDY%=F%(R%))OR(B%=E%
(R%)ANDC%=F%(R%))THENPROCdead
 2250 ENDPROC
```




```
 2260 :
 2270 DEFPROChyp:REPEAT:E%(R%)=RND(19):
F%(R%)=(RND(9)*3)+1:UNTILE%(R%)<>X% AND
F%(R%)<>Y% ANDE%(R%)<>B% ANDE%(R%)<>C%
ANDA%(E%(R%),F%(R%))<>4 ANDA%(E%(R%),F%
(R%))<>5
 2280 ENDPROC
 2290 :
 2300 DEFPROCstill
 2310 IFK%=19 P%=-1:SG%(R%)=FALSE ELSEI
FK%=0 P%=1:SG%(R%)=FALSE
 2320 IF(A%(K%+P%,L%)=4 ORA%(K%+P%,L%)=
5)SG%(R%)=FALSE:PROChyp ELSEE%(R%)=E%(R
%)+P%
 2330 ENDPROC
 2340 :
 2350 DEFPROCsu
 2360 DEAD%=0
 2370 COLOUR3
 2380  FORI%=1 TO3:E%(I%)=8:NEXTI%
 2390 F%(1)=4:F%(2)=13:F%(3)=25
 2400 FORI%=1TO3
 2410 PRINTTAB(E%(I%),F%(I%));G$:A%(E%(
I%),F%(I%))=4
 2420 NEXTI%
 2430 REPEAT:X%=RND(18):Y%=(RND(8)*3)+1
:UNTILA%(X%,Y%)=0 ANDA%(X%-1,Y%)=0:Z%=1
:B%=X%-1:C%=Y%
 2440 COLOUR3:PRINTTAB(X%,Y%);M$(1):COL
OUR2:PRINTTAB(B%,C%);PRL$(1)
 2450 COLOUR3:PRINTTAB(3,3);"SHEET ";U%
 2460 COLOUR3:FORI%=1TOH%:PRINTTAB(I%,3
1)M$(1);:NEXTI%
 2470 ENDPROC
 2480 :
 2490 DEFPROCdead:PRINTTAB(H%,31);SPC1;
:H%=H%-1
 2500 DEAD%=TRUE
 2510 *FX15,0
 2520 COLOUR3:SOUND0,3,5,1
 2530 PRINTTAB(B%,C%);SPC1
 2540 PRINTTAB(X%,Y%);MD$
 2550 ENDPROC
```

→

```
 2560 :
 2570 DEFPROCfall
 2580 LA%=TRUE
 2590 PROCprint
 2600 COLOUR3:LA%=FALSE
 2610 REPEAT
 2620 COLOUR3:PRINTTAB(X%,Y%);M$(3)
 2630 IFA%(X%,Y%)=6 PROCdead
 2640 IFA%(X%,Y%)=-1 COLOUR2:PRINTTAB(X
%,Y%);P$ ELSEPRINTTAB(X%,Y%);SPC1
 2650 Y%=Y%+1
 2660 UNTILA%(X%,Y%+1)=-1 ORA%(X%,Y%+1)
=7 ORA%(X%,Y%+1)=8 ORDEAD%
 2670 IFDEAD% ENDPROC
 2680 PRINTTAB(X%,Y%);M$(3):B%=X%-1:C%=
Y%-1:Z%=3
 2690 IFA%(X%,Y%)=4 PROCdead
 2700 ENDPROC
 2710 :
 2720 DEFPROCdig
 2730 IFA%(B%,C%+3)=2 ORA%(B%,C%+1)=0 O
RA%(B%,C%)=2 ORY%=28 ORZ%=3 ORB%=0 ORB%
=19 ENDPROC
 2740 COLOUR2:PRINTTAB(B%,C%);PD$(Z%):P
ROCg:IFDEAD% ENDPROC
 2750 COLOUR2
 2760 IFA%(B%,C%+1)=6 PROCmonstfall:GOT
O2810
 2770 SOUND0,1,206,1
 2780 IFA%(B%,C%+1)=-1 A%(B%,C%+1)=7:PR
INTTAB(B%,C%+1);B$(1):GOTO2810
 2790 IFA%(B%,C%+1)=7 A%(B%,C%+1)=8:PRI
NTTAB(B%,C%+1);B$(2):GOTO2810
 2800 IFA%(B%,C%+1)=8 A%(B%,C%+1)=0:PRI
NTTAB(B%,C%+1);SPC1
 2810 COLOUR2:PRINTTAB(B%,C%);PRL$(Z%)
 2820 ENDPROC
 2830 :
 2840 DEFPROCmonstfall
 2850 LV%=1
 2860 FORI%=1TO3
 2870 IFDM%(I%)GOTO2890
 2880 IFE%(I%)=B% ANDF%(I%)=C%+1 XX%=E%
(I%):YY%=F%(I%):RR%=I%
 2890 NEXTI%
 2900 COLOUR2:PRINTTAB(XX%,YY%);P$:A%(X
X%,YY%)=-1
 2910 YY%=YY%+1:COLOUR3:PRINTTAB(XX%,YY
%);G$
 2920 REPEAT
 2930 IFA%(XX%,YY%)=0 PRINTTAB(XX%,YY%)
;SPC1 ELSEIFA%(XX%,YY%)=6 PRINTTAB(XX%,
YY%);G$
 2940 YY%=YY%+1
 2950 PRINTTAB(XX%,YY%);G$
 2960 IFINT((YY%-2)/3)=(YY%-2)/3 LV%=LV
%+1
 2970 UNTILA%(XX%,YY%+1)=-1 ORA%(XX%,YY
%+1)=7 ORA%(XX%,YY%+1)=8
 2980 TRAP%(RR%)=FALSE
 2990 IFLV%>=U% PRINTTAB(XX%,YY%);SPL$:
SOUND0,3,5,1:PRINTTAB(XX%,YY%);SPC1:S%=
S%+(LV%*100):DM%(RR%)=TRUE:CO%=CO%+1 EL
SEE%(RR%)=XX%:F%(RR%)=YY%:A%(XX%,YY%)=4
 3000 ENDPROC
 3010 :
 3020 DEFPROCscore
 3030 COLOUR3:PRINTTAB(3,1);"SCORE ";S%
 3040 BON%=OXY%-(INT(TIME/100))
 3050 IFBON%<=99 PRINTTAB(15,31);SPC1;
 3060 IFBON%<=9 PRINTTAB(14,31);SPC1;
 3070 PRINTTAB(6,31);"OXYGEN ";BON%;
 3080 IFBON%<=0 PROCdead
 3090 ENDPROC
 3100 :
 3110 DEFPROCback(BX%,BY%)
 3120 IFA%(BX%,BY%)=0 PRINTTAB(BX%,BY%)
;SPC1:ENDPROC
 3130 IFA%(BX%,BY%)=2 COLOUR1:PRINTTAB(
BX%,BY%);L$:ENDPROC
 3140 IFA%(BX%,BY%)=-1 COLOUR2:PRINTTAB
(BX%,BY%);P$:ENDPROC
 3150 IFA%(BX%,BY%)=7 COLOUR2:PRINTTAB(
BX%,BY%);B$(1):ENDPROC
 3160 IFA%(BX%,BY%)=8 COLOUR2:PRINTTAB(
BX%,BY%);B$(2):ENDPROC
 3170 ENDPROC
 3180 :
 3190 ON ERROR OFF
 3200 MODE 7
 3210 IF ERR=17 END
 3220 REPORT:PRINT" at line ";ERL
 3230 END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

LOCAL PARAMETERS - D. Morgan

When passing parameters to a procedure, not only can you 'local' variables, but also memory locations. For example:

```
DEF PROCtest(!A%)                    DEF PROCoscli($oscli%)
LOCAL $X                             LOCAL !Z
```

are both valid. Note that the two 'types' being 'localled' are both valid Basic types, an integer (4 bytes) and a string (a variable number of bytes). To use ?A% would cause unpredictable results, as Basic does not fully cater for a 'single byte' type.

RED ALERT

by Alan Barratt and David Green

If chess is too time consuming, solitaire too anti-social and draughts too predictable, then try this simple to learn yet challenging and decidedly unpredictable board game.

RED ALERT is a game of Scandinavian origin for two players. It is played on a six by six square board. Players take turns to place one counter at a time onto the board.

One player has pale blue counters and the other yellow. You are not allowed to place a counter directly on top of an opponent's counter but you can, and indeed should, build up on your own.

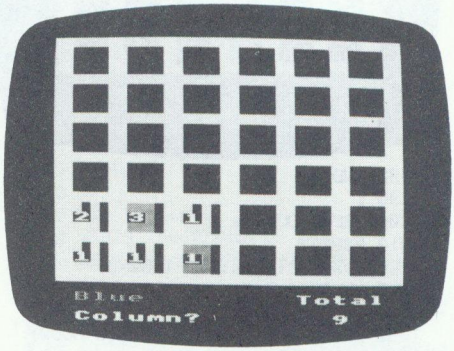
Each square has a critical mass. This value is the number of adjacent squares (horizontal and vertical) which the square has. This means that the critical masses are as follows:

Corner squares ... 2
Edge squares 3
Inner squares 4

Once the number of counters on any square reaches that square's critical mass the square 'explodes' and the counters disperse, one onto each adjacent horizontal and vertical square, leaving the original square now empty. This is the way to capture your opponent's counters: arrange an explosion next to an occupied square. Any counters on these adjacent squares become your own, and remain on these squares.

When well into a game, with several counters across the board, exploding one square can lead to a chain reaction that spreads right across the board. This gives Red Alert its unpredictability and challenge. It is quite possible to be down to two or three counters with your opponent having fifty, and then see him wiped out with a single, clever, move.

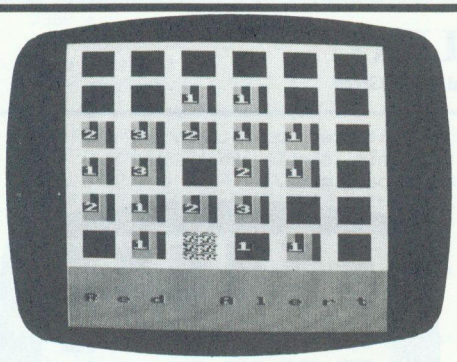
The winner is the player who eliminates all his opponent's counters. Alternatively, the game may be played for a fixed number of moves, the winner being the player with the most counters at that stage. A counter is placed by designating the column and row positions. These both count from the bottom left hand corner of the board. The total number of counters on the board is displayed at the bottom of the screen at all times.



Red Alert makes colourful use of the BBC micro's mode 2 display. If you only have a black and white TV or monitor, the counters are shaped differently so that you can tell your own from your opponent's. Players with disc systems should set PAGE to &1100 before loading and running this program.

```
 10 REM Program Red Alert
 20 REM Version B0.1
 30 REM Authors Alan Barratt
 40 REM         & David Green
 50 REM BEEBUG  Jan/Feb 1985
 60 REM Program subject to copyright
 70 :
 80 ON ERROR GOTO 2670
 90 :
100 MODE 7
110 PROCinstr
120 MODE 2
130 PROCsetup
140 PROCscreen
150 PROCgame
160 PROCfinish
170 IF Z<3 THEN RUN
```

→




```
 180 END
 190 :
1000 DEF PROCgame
1010 REPEAT
1020 Z=Z+1:Z1=3-Z:B=B+1:C2=0
1030 VDU19,4,4;0;
1040 COLOUR 128:CLS
1050 PRINT TAB(1,1) PLAYER$(Z MOD2)
1060 COLOUR 7
1070 PRINT TAB(13,1)"Total"TAB(15,3);B
-1
1080 FOR I=1 TO 2
1090 PRINTTAB(3*I-2,1+2*I);CO$(I);
1100 *FX15,0
1110 REPEAT
1120 XY$(I)=GET$
1130 UNTIL ASC(XY$(I))>48 AND ASC(XY$(
I))<55
1140 PRINT XY$(I)
1150 SOUND1,-15,220,4
1160 PROCwait(0.1)
1170 NEXT I
1180 X%=VAL(XY$(1)):Y%=VAL(XY$(2))
1190 IF new$(X%,Y%)=XO$(Z1) PROCsorry:
GOTO1050 ELSE new$(X%,Y%)=XO$(Z):new(X%
,Y%)=new(X%,Y%)+1
1200 REPEAT
1210 C1=0:C3=0:C4=0
1220 FOR X%=1 TO 6
1230 FOR Y%=1 TO 6
1240 IF NOT(new$(X%,Y%)=old$(X%,Y%) AN
D new(X%,Y%)=old(X%,Y%)) PROCplot(X%,Y%
,Z)
1250 IF C2<2 AND new(X%,Y%)>=D(X%,Y%)
PROCupdate
1260 IF new$(X%,Y%)=XO$(Z1) C3=1:C4=1
1270 NEXT Y%
1280 NEXT X%
1290 IF C4=0 C2=C2+1 ELSE IF C2=3 C1=0
1300 UNTIL C1=0:Z=2-Z
1310 UNTIL C3=0 AND B>1
1320 ENDPROC
1330 :
1340 DEF PROCsetup
1350 DIM new(6,6),old(6,6),new$(6,6),o
ld$(6,6),D(6,6)
1360 DIM XO$(2),OX$(2,3),PLAYER$(1),CO
$(2),XY$(2)
1370 PLAYER$(0)=CHR$17+CHR$6+"Blue"
1380 PLAYER$(1)=CHR$17+CHR$3+"Yellow"
1390 VDU24,0;224;1215;1023;
1400 VDU28,0,31,18,25
1410 VDU23,224,&FF,&FF,&FF,&FF,&FF,&FF
,&FF,&FF
1420 VDU23,225,&F0,&F0,&F0,&F0,&F0,&F0
,&F0,&F0
1430 VDU23,1,0;0;0;0;
1440 FOR X%=1 TO 2
1450 FOR Y%=1 TO 3
1460 READ A,B
1470 OX$(X%,Y%)=CHR$A+CHR$B
1480 NEXT Y%
1490 NEXT X%
1500 CO$(1)="Column?":CO$(2)="Row?"
1510 XO$(1)="X":XO$(2)="old"
1520 FOR X%=1 TO 6
1530 FOR Y%=1 TO 6
1540 READ D(X%,Y%)
1550 new$(X%,Y%)=" ":old$(X%,Y%)=" "
1560 NEXT Y%
1570 NEXT X%
1580 B=0:Z=0
1590 GCOL0,135:CLG
1600 PROCalert
1610 ENDPROC
1620 :
1630 DEF PROCwait(new)
1640 TIME=0
1650 REPEAT UNTIL TIME > 50*new
1660 ENDPROC
1670 :
1680 DEF PROCsorry
1690 SOUND1,-15,112,5
1700 SOUND1,-15,100,10
1710 SOUND1,-15,96,1
1720 SOUND1,-15,100,10
1730 COLOUR 12:PRINT TAB(11,3);"Not"TA
B(11,5)"Allowed"
1740 COLOUR 0:PROCwait(4):CLS
1750 ENDPROC
1760 :
1770 DEF PROCclear(X%,Y%)
1780 VDU5
1790 GCOL0,4
1800 FORE%=1TO3
1810 MOVE(X%*3-2)*64,252+(Y%*4-4+E%)*32
1820 PRINT CHR$224+CHR$224
1830 NEXT
1840 VDU4
1850 ENDPROC
1860 :
1870 DEF PROCscreen
1880 VDU19,7,11;0;19,4,11;0;
1890 VDU23,1,0;0;0;0;
```

→

```
1900 FOR X%=1 TO 6
1910 FOR Y%=1 TO 6
1920 PROCclear(X%,Y%)
1930 NEXT Y%
1940 NEXT X%
1950 VDU19,4,4;0;19,7,7;0;
1960 ENDPROC
1970 :
1980 DEF PROCplot(X%,Y%,Z)
1990 PROCclear(X%,Y%)
2000 IF new$(X%,Y%)<>" " VDU5:GCOL0,3*
Z:FOR E%=1 TO 3:MOVE(X%*3-2)*64,252+(Y%
*4-4+E%)*32:PRINT OX$(Z,4-E%):NEXT E%:M
OVE(X%*3-3)*64,252+(Y%*4-2)*32:GCOL0,7:
PRINT" ";new(X%,Y%):VDU4
2010 old(X%,Y%)=new(X%,Y%)
2020 old$(X%,Y%)=new$(X%,Y%)
2030 C1=1
2040 ENDPROC
2050 :
2060 DEF PROCupdate
2070 VDU19,4,1;0;:PROCalert
2080 IF X%<>6 new$(X%+1,Y%)=new$(X%,Y%
):new(X%+1,Y%)=new(X%+1,Y%)+1
2090 IF X%<>1 new$(X%-1,Y%)=new$(X%,Y%
):new(X%-1,Y%)=new(X%-1,Y%)+1
2100 IF Y%<>6 new$(X%,Y%+1)=new$(X%,Y%
):new(X%,Y%+1)=new(X%,Y%+1)+1
2110 IF Y%<>1 new$(X%,Y%-1)=new$(X%,Y%
):new(X%,Y%-1)=new(X%,Y%-1)+1
2120 new(X%,Y%)=new(X%,Y%)-D(X%,Y%)
2130 old(X%,Y%)=10
2140 IF new(X%,Y%)=0 new$(X%,Y%)=" "
2150 PROCsmash(X%,Y%)
2160 PROCplot(X%,Y%,Z)
2170 ENDPROC
2180 :
2190 DEF PROCsmash(X%,Y%)
2200 VDU5
2210 PROCwait(1)
2220 FOR Q%=5 TO 255 STEP 50
2230 GCOL0,RND(15)
2240 VDU23,240,RND(Q%),RND(Q%),RND(Q%)
,RND(Q%),RND(Q%),RND(Q%),RND(Q%),RND(Q%)
2250 FORE%=1TO3
2260 MOVE(X%*3-2)*64,252+(Y%*4-4+E%)*3
2:PRINTCHR$240+CHR$240
2270 NEXT E%
2280 NEXT Q%
2290 VDU4
2300 PROCclear(X%,Y%)
2310 ENDPROC
2320 :
2330 DEF PROCalert
2340 ENVELOPE1,1,4,-4,4,10,20,10,127,1
27,0,0,127,126
2350 SOUND1,1,100,30
2360 COLOUR 139:CLS:COLOUR 12
2370 PRINTTAB(1,3);"R e d   A l e r t"
2380 ENDPROC
2390 :
2400 DEF PROCfinish
2410 COLOUR 132:CLS
2420 SOUND1,-15,120,25
2430 COLOUR12:PRINT TAB(3,1)"The Winne
r!"
2440 COLOUR 0:PRINT TAB(4,3)"Score  ";
B;
2450 COLOUR 7:PRINT TAB(4,5)"Press bar
";
2460 *FX15,1
2470 I=GET:CLS
2480 PRINTTAB(2,2)"Another game ?";
2490 REPEAT Z=INSTR("YyNn",GET$):UNTIL
Z<>0
2500 ENDPROC
2510 :
2520 DEF PROCinstr
2530 VDU23,1;0;0;0;0;
2540 PRINTTAB(9,1)CHR$129;CHR$157;TAB(
18)CHR$156
2550 PRINTTAB(9,2)CHR$129;CHR$157;CHR$
135;" RED ALERT !   ";CHR$156
2560 PRINTTAB(9,3)CHR$129;CHR$157;TAB(
18)CHR$156
2570 PRINT TAB(3,7)CHR$131;"Capture yo
ur opponents squares by" ' CHR$131;"'ex
ploding' your own."
2580 PRINT TAB(3,10)CHR$131;"The first
 player to wipe out all" ' CHR$131;"tra
ce of his opponent wins the game."
2590 PRINT TAB(3,13)CHR$131;"The criti
cal mass of the squares" ' CHR$131;"var
ies around the board:"
2600 PRINT TAB(7,17)CHR$130;"Corner sq
uares......2"
2610 PRINT TAB(7,19)CHR$130;"Edge squa
res........3"
2620 PRINT TAB(7,21)CHR$130;"Inner squ
ares.......4"
2630 PRINT TAB(6,24)"PRESS SPACE BAR T
O START";
2640 REPEAT UNTIL GET=32
2650 ENDPROC
2660 :
2670 ON ERROR OFF
2680 MODE 7
2690 IF ERR<>17 REPORT:PRINT" at line
";ERL
2700 END
2710 :
2720 DATA 225,225,32,225,224,225,224,2
25,32,225,224,225
2730 DATA 2,3,3,3,3,2,3,4,4,4,4,3,3,4,
4,4,4,3,3,4,4,4,4,3,3,4,4,4,4,3,2,3,3,3
,3,2
```

IF YOU WRITE TO US

BACK ISSUES (Members only)

All back issues are kept in print (from April 1982) priced as follows:

Individual copies:
Volume 1 - £0.80
Volume 2 - £0.90
Volume 3 - £1.00

Volume 1 set (10 issues) £7
Volume 2 set (10 issues) £8

Please add cost of post and packing as shown:

No of copies	DESTINATION UK	Europe	Elsewhere
1	0.30	0.70	1.50
2 - 5	0.50	1.50	4.70
6 - 10	1.00	3.00	5.50
11 - 20	1.50	4.00	7.00

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.

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

SUBSCRIPTIONS

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

MEMBERSHIP COSTS:

U.K.
£6.40 for 6 months (5 issues)
£11.90 for 1 year (10 issues)

Eire and Europe
Membership £18 for 1 year.
Middle East £21
Americas and Africa £23
Elsewhere £25
Payment in Sterling essential.

PROGRAMS AND ARTICLES

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

HINTS

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

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

Editorial Address

BEEBUG
PO Box 50
St Albans
Herts

Subscriptions & Software Address

BEEBUG
PO BOX 109
High Wycombe
Bucks HP10 8HQ

Hotline for queries and software orders

St.Albans (0727) 60263
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.

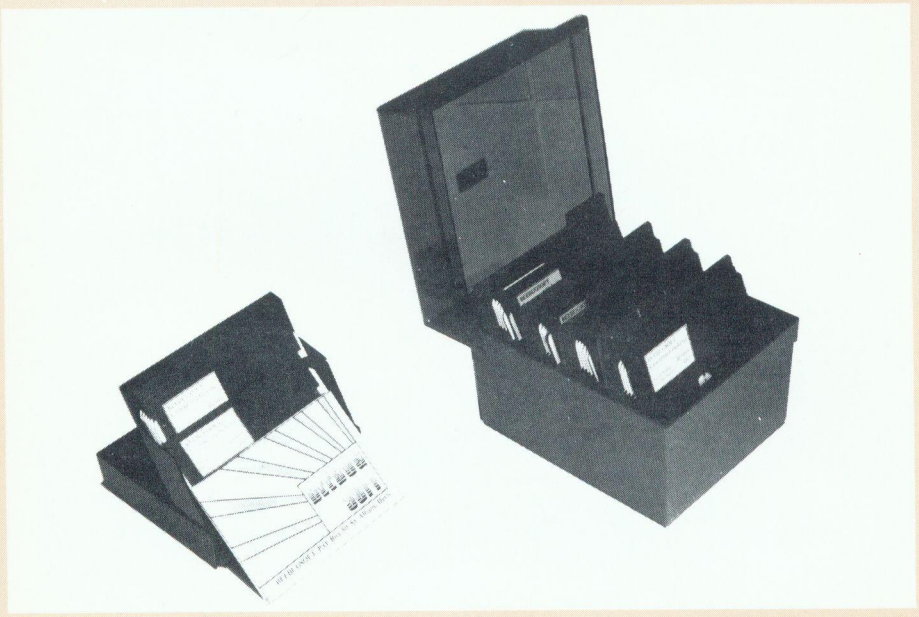
BEEBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams.
Assistant Editor: Geoff Bains. Production Editor: Phyllida Vanstone.
Technical Assistants: David Fell and Alan Webster.
Managing Editor: Lee Calcraft.
Thanks are due to Sheridan Williams, Adrian Calcraft, Matthew Rapier, John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

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

High Quality Low Priced Discs

Backed by The Reputation of BEEBUG



10 S/S D/D Discs – £13.90	10 D/S D/D Discs – £19.40
25 S/S D/D Discs – £33.45	25 D/S D/D Discs – £46.95
50 S/S D/D Discs – £59.30	50 D/S D/D Discs – £87.05

All Prices Include Storage Box, VAT and Delivery to Your Home (UK).

All discs are 100% individually tested, supplied with hub ring as standard, and guaranteed error free. They are ideal for use on the BBC Micro and have performed perfectly in extensive tests at BEEBUG over many months.

Orders for 25 or 50 are delivered in strong plastic storage boxes with four dividers. Orders for 10 are sent in smaller hinged plastic library cases.

We are also able to offer the empty storage container, which holds up to 50 discs for £10 including VAT and post.

Please use the order form enclosed
or order directly from:
BEEBUGSOFT, P.O. Box 109,
High Wycombe, Bucks HP10 8HQ.

BEEBUG SOFT

THE BEEBUG MAGAZINE ON DISC AND CASSETTE

The programs featured each month in the BEEBUG magazine are now available to members on disc and cassette.

Each month we will produce a disc and cassette containing all of the programs included in that month's issue of BEEBUG. Both the disc and the cassette will display a full menu allowing the selection of individual programs and the disc will incorporate a special program allowing it to be read by both 40 and 80 track disc drives. Details of the programs included in this month's magazine cassette and disc are given below.

Magazine cassettes are priced at £3.00 and discs at £4.75.
SEE BELOW FOR FULL ORDERING INFORMATION.

This Month's Programs Include:

A colourful and graphical 3D Surfaces display, a program for Making Music on the Beeb, a practical routine to Disable the Break Key, easy to use Assembler Arithmetic routines, four BEEBUG Workshop examples of indirection operators, a fast moving game of monsters called Digger, a Split Screen Utility providing dual windows on the screen, a very original and thought provoking game named Red Alert and an extra utility for Basic programmers, Crunch, a program that really squeezes the last ounce of memory space from your programs.

MAGAZINE DISC/CASSETTE SUBSCRIPTION

Subscription to the magazine cassette and disc is also available to members and offers the added advantage of regularly receiving the programs at the same time as the magazine, but under separate cover.

Subcription is offered either for a period of 6 months (5 issues) or 1 year (10 issues) and may be backdated if required. (The first magazine cassette available is Vol. 1 No. 10; the first disc available is Vol. 3 No. 1.)

MAGAZINE CASSETTE SUBSCRIPTION RATES

6 MONTHS	(5 issues)	UK £17.00 INC... Overseas £20.00 (No VAT payable)
1 YEAR	(10 issues)	UK £33.00 INC... Overseas £39.00 (No VAT payable)

MAGAZINE DISC SUBSCRIPTION RATES

6 MONTHS	(5 discs)	UK £25.50 INC... Overseas £30.00 (No VAT payable)
1 YEAR	(10 discs)	UK £50.00 INC... Overseas £56.00 (No VAT payable)

CASSETTE TO DISC SUBSCRIPTION TRANSFER

If you are currently subscribing to the BEEBUG magazine cassette and would prefer to receive the remainder of your subscription on disc, it is possible to transfer the subscription. Because of the difference between the cassette and disc prices, there will be an extra £1.70 to pay for each remaining issue of the subscription. Please calculate the amount due and enclose with your order.

ORDERING INFORMATION

Please send your order to the address below and include a sterling cheque. Postage is included in subscription rates but please add 50p for the first item and 30p for each subsequent item when ordering individual discs or cassettes in the UK. Overseas orders please send the same amount to include the extra post but not VAT.
SEND TO:

BEEBUGSOFT, PO BOX 109, HIGH WYCOMBE, BUCKS, HP10 8HQ

Printed in England by Staples Printers St Albans Limited at The Priory Press.

ISSN 0263-7561