A Database Publication

apple user

Vol. 5 No. 10 October 1985 £1

Two picture digitisers reviewed

Is there a 512k Apple II on the stocks?

Pascal assembly language programming

Memory-boosting Ramworks reviewed

Spreadsheet model for home budgets

Instant access to Page 3 routines

Creating your own database: Start of a major new series

apple user
Volume 5
Number 10
October 1985

News

● A 512k Apple II on the horizon, Steve Jobs is selling his Apple shares, and more. 5

MicroLink

● This month's update on news from Britain's national electronic mail service. 9

&DOSFILE

● Peter Harris starts a new series with some powerful file management routines. 11

Spreadsheet

● If you are one of the many people who pay your bills by monthly budget, this spreadsheet from Chris Burridge could save you time and money. 18

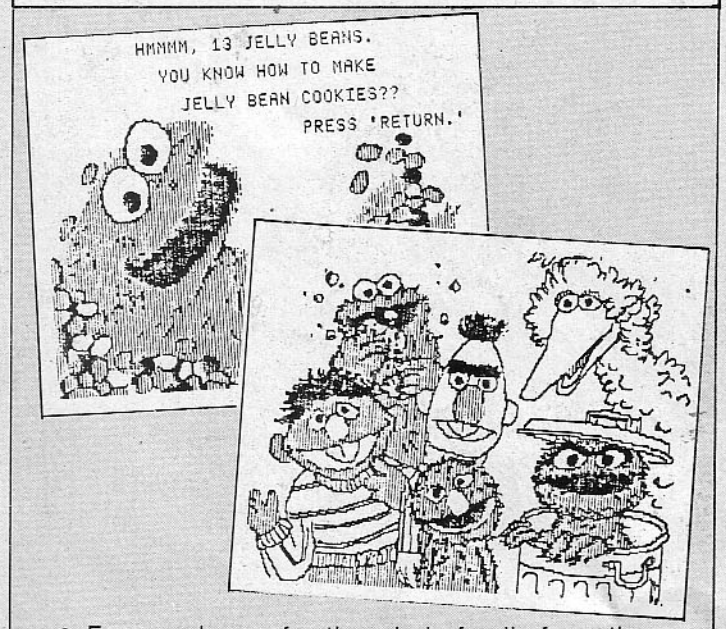
Education

● How a Hertfordshire college is putting Apples to good use. 22

Utility

● Use your language card and David Haynes' program to provide instant access to all those Page 3 routines. 23

Fun & Games



● Four packages for the whole family from the Children's Television Workshop, and at Bargain Basement prices too. 27

Review

● Geoff Wood expands his Appleworks desktop with Ramworks. 29

● Cliff McKnight looks at two digitisers for the Apple II and Macintosh. 32

Appletips

● A full page of tips here, and there's more throughout the magazine. 31

Graphics

● In Part XVI of the Apple User Graphics Library Peter Gorry gives updates for the histogram routines. 35

Programming

● S. Eveson's handy USR routine adds a factorial function to Basic. 37

Pascal

● Stuart Bell continues his tutorial series with a look at assembly language programming under Pascal. 38

● David Wells presents a modification to the Pascal output routine to use the lower case chip. 42

New Products

● All the latest ideas on how to spend your money. 45

Utility

● Find out how much space remains on your discs with Max Parrott's useful routine. 53

Classifieds

● Bargain offers from your fellow Apple users. 55

Feedback

● Spell-check under ProDOS, Belgian Mac user group, trouble with Sage Accounts, and more. 56

Order form

● Subscriptions, back issues, binders to keep them in. 58

Apple User Managing Editor **Derek Meakin** *Features Editor* **Cliff McKnight** *Technical Editor* **Max Parrott**
Production Editor **Peter Glover** *Advertisement Manager* **John Riding** *Sales* **Chris Riding**
Design **Heather Sheldrick** *News Editor* **Mike Cowley** *Editor-in-Chief Database Publications* **Peter Brameld**

Telephone: 061-456 8383 (Editorial) 061-456 8500 (Advertising) 061-480 0171 (Subscriptions)
Telex: 265871 MONREF G Quoting Ref. 79:MAG001 Telecom Gold Mailbox: 79:MAG001 Prestel Mailbox: 614568383

Published by Database Publications Ltd, Europa House, 68 Chester Road, Hazel Grove, Stockport SK7 5NY.

News trade distribution: Europress Sales and Distribution Limited, 11 Brighton Road, Crawley, West Sussex RH10 6AF. Circulation 0293 27053.

Writing for Apple User: Articles and programs relating to the Apple are welcome. Articles should preferably be typed or computer-printed, using double spacing. Unsolicited manuscripts, discs etc, should be accompanied by a self addressed stamped envelope, otherwise their return cannot be guaranteed. Unless agreed, material is accepted on an all rights basis.

ABC 12,850 – JULY -DECEMBER 1984

Subscription rates for 12 issues, post free:
£12 UK
£13 Eire (IR £16)
£20 Europe
£15 Rest of world (surface)
£30 Rest of world (airmail)

Jobs unloads $8 million

IS it the end of the affair for Apple and Steve Jobs?

Rumours are flying in the United States that co-founder Jobs is about to leave Apple and embark on a new career.

Speculation is fuelled by Jobs' move to sell 500,000 shares valued at $8 million only two months after unloading 850,000 shares for $14 million following a clash with Apple president and chief executive John Sculley.

This summer Jobs was stripped of all day-to-day duties at Cupertino and, although he remains the company's biggest shareholder with 5½ million shares, his latest trip to his brokers will mean he has disposed of 19 per cent of his stake in only a few weeks.

Is there a 512k Apple II on the horizon?

APPLE watchers in the United States are expecting a new version of the Apple II to be launched in the near future.

One report says the new machine will make an appearance "sometime between now and next January" and that a prototype would be shown to top American dealers around the end of September.

But spokesmen at Apple's Cupertino headquarters are refusing to either confirm or deny that a new computer is in the pipeline, says *Apple User's* US source.

Current rumours point to the machine taking either one of two forms.

Rumour One says it will be based around the Western Design 65802/65816 processor and include up to 512k of RAM.

Storage will be in the form of the 3.5in format pioneered by Sony.

It is suggested that, unlike the 6502-based Apple II, it will be capable of making full use of an optional mouse.

Rumour Two says it will be a 68020-based dual processor machine emulating the 6502 and 65C02 in two ways.

"The latter would make sense in view of Apple's current policy of encouraging development of software and interfaces and could do the company a lot of good after its recent problems", said *Apple User's* contact.

Apple joins Queen

POP superstars Queen have followed other famous groups like the Boomtown Rats and Mainframe into the world of Apple II operated musical sound creation.

Our picture shows the group's drummer Roger Taylor (left) trying a chord or two on the Greengate DS:3 digital sound sampling sequencer during Queen's recent tour of the Far East and Australia. Looking on is Spike, the group's keyboard player.

An Apple and DS:3 combined to provide most of the backing for the latest single from Mainframe.

The A side "Five Minutes" has a wealth of sampled sound in the bass and percussion parts plus the vocal effects and the lead lines on virtually all instruments.

More than 90 per cent of "Five Minutes" is DS:3 in both sample and sequence mode while the B side "Eric's Revenge" is all DS:3 created.

Facts – and rumours – at the big Mac show

Jim Mangles reports from Boston, USA

THE Bayside Expo Center in Boston was the location of the Summer 1985 Mac World Exhibition.

More than 300 exhibitors, including at least four from the UK, and over 20,000 visitors came together to make a very successful trade-orientated show. At least, it must be assumed that the show was trade-orientated, because it closed every evening promptly at 6pm and did not open at the weekend.

A number of exhibitors expressed surprise at this apparent attempt to keep the general public out, which seemed especially strange in a city like Boston.

There are probably more Macs in use in Boston than in the whole of the UK. As is usual in America, a very large proportion – perhaps over 50 per cent – are estimated to be owned by individuals.

The centrepiece of the whole show was the 12ft high Mac first revealed in the February issue of *Apple User*. Throughout the exhibition, top-line Macintosh software was demonstrated on it in a non-stop, almost overwhelming stream.

As might be expected, Jazz from Lotus and Excel from Microsoft were the show-stoppers. My own money is on Excel. Maybe I'm just an old-fashioned power user, but that one has me twitching to get my hands on it.

Software is no longer in short supply for the Mac. Anyone who tells you it still is was not there, or had their eyes shut, or is in the pay of Big Blue.

The Expo was awash in the stuff – every kind of product for every imaginable purpose – and many that could not have been imagined.

It is quite impossible to list all the new programs now, but here are some that got my particular attention:

MacNosy is described as "the disassembler for the rest of us". Well, maybe, if you're one of the rest of us who happens to know assembler.

But this is a very clever tool which will reconstruct 68000 source code from almost any chunk of Mac object code, "copy-protected" or not – even the 64k Rom can be disassembled.

FONTastic is a font editor. This one really is for the rest of us, and probably for those of us who have had the misfortune to struggle with Apple's Font Editor in the past as well.

Borland released **SideKick** for the Mac in Boston. It's very nice, but at $99 but with desk accessories a standard feature on the Mac and available in vast profusion in the public domain, I'm not sure that they can expect the same success they achieved with their product of the same name running in the atrocious IBM environment.

Assimilation is a company that specialises in producing low-priced products with high-priced performance. At $29 each, **Mac·Tracks** (a desk accessory that let's you "macro" almost any application, including Jazz), **Work·n·Print** (a print spooler), **Lock·It** (a passwording system for data and applications), and **Mac·Memory·Disk** (a Ram disc for 512k Macs), are too good to pass up.

On the hardware front, there can be little doubt that the most spectacular item present was a four – yes, four – mbyte upgrade on offer by one small company for only $900. I have my doubts about that one.

One and two mbyte upgrades now seem to be quite common, offered by a number of exhibitors, and at least two stalls were offering double-sided 3½in drives with 800k capacity.

General Computers unveiled their (internally installed) 20mbyte **Hyperdrive** upgrade. Now that it doesn't invalidate Apple's warranty, this is the only hard disc system worth considering for the Mac.

MacCharlie was alive and well, and living in Boston. I suppose that seeing Lotus 1.2.3 running on a Mac is impressive.

No, let me be fair. It is impressive to see the famous spreadsheet format, in the correct Mac text format of black letters on white background, in a resizable, moveable window, with mouse-activated cursor control and cut-and-paste functioning. But what a waste of a Mac!

In a similar vein, Abaton Technology Group proudly unveiled **The Abaton Transform: a2m.** Really more soft- than hard-ware, this system offers to convert any stand-alone Apple II program into a Macintosh application (!).

No, I didn't believe it either, but there it was – a Mac desktop covered with application icons with names like VisiCalc, Choplifter, Flight Simulator II, etc. I saw Flight Simulator working, just like it would on a monochrome Apple II screen, so there must be something in it.

The rumour mill was grinding away at high speed, which was to be expected after the recent changes at Cupertino.

I offer no warranty on the accuracy of what follows, and anyway events may have caught up with the rumours by the time this appears in print, but the most popular one is that we can expect to see an upward compatible "Super-Mac" sometime in the next few months, with an higher-powered member of the 68000 family as CPU, higher clock speed, either 1 or 2 mbyte Ram as standard on the motherboard, detachable monitor which has a higher resolution than the present Mac screen and can be optionally upgraded to colour, double-sided drives, and – please – SLOTS.

If such a machine really is in the wings, and at a reasonable price, Apple will have a real "AT-buster" at last.

Symbiotic, ICE link

LEADING manufacturers of hard disc systems and local area networks, Symbiotic Computer Systems and ICE have merged to form what is described as the largest European supplier within their industry.

Networking specialists Symbiotic, principal supplier of storage peripherals on the Apple range for business and education, gives its name to the new company as well as its newly designed high performance disc controller.

In its early years also an Apple system supplier, ICE has in the last two years moved throughout Europe into the MS DOS market..

The new company will benefit from Symbiotic's subsidiaries in France and the Benelux nations, coupled with ICE's distribution network in 22 other countries.

THE POLL-TOPPER

BRITAIN'S leading business computer dealers have voted Apple's recent "Test drive a Macintosh" campaign their favourite promotional event.

A survey carried out by the Inteco Corporation also revealed that UK dealers are well ahead of their counterparts on the Continent in their adoption of networking and support of multiuser systems.

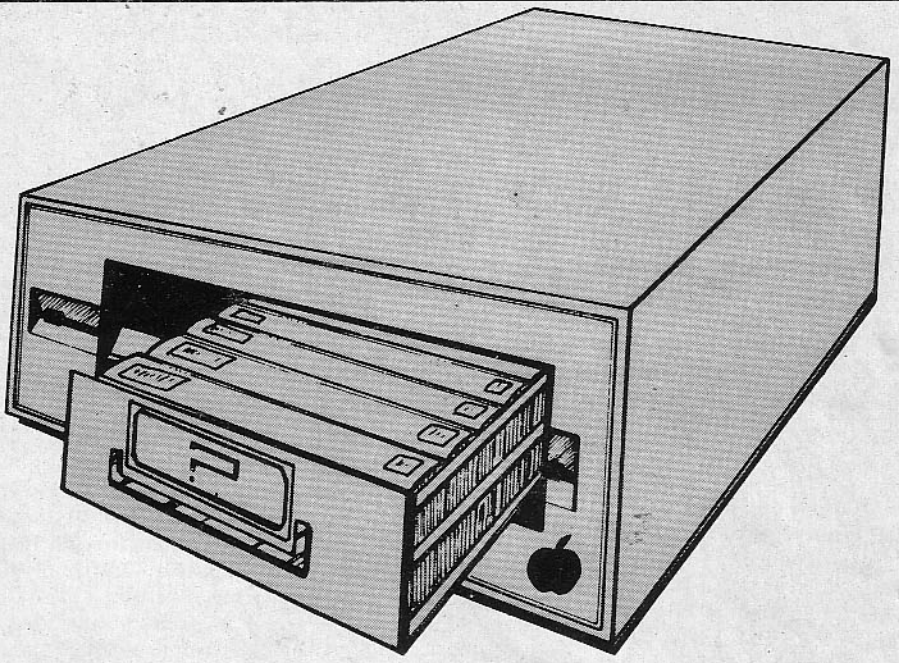
Although Apple, like IBM, had not yet made UK deliveries of networks, 66 per cent of dealers offer network solutions in response to customer requests.

Dealers voted Multiplan the most popular spreadsheet package, closely followed by Lotus 1,2,3 with Supercalc third.

In the word processor category, Wordstar was twice as popular as second placed Wordcraft.

Leading database was dBase II followed by Omnis, Delta and dBase III.

Among integrated applications, Lotus 1,2,3 was first, followed by Symphony and AppleWorks.



Fast way to find those half-forgotten files

This month PETER HARRIS starts a major series aimed at providing a database program complete with built-in form generator. He begins the series with a powerful set of file maintenance commands.

APPLE'S DOS 3.3 disc operating system is generally not difficult to use. Indeed it has some advantage over systems which need to store files on adjacent blocks on a disc in that additions can be made to any number of files without them colliding and causing "Disc full" error messages.

However there is one thing it does not allow – the possibility of opening and reading random-access files using direct commands rather than from within a program.

It would be incredibly convenient, for example, when faced with a catalogue of half-forgotten text files, if one could open and inspect each file without having to write a program for it.

Moreover some random-access files are created by somewhat odd programs, which makes it difficult to create a general-purpose file-reading program.

This problem first came to my attention when I tried to do some editing of the Animalsfile file created by the Animals game on the DOS 3.3 master disc.

My children descended on this game with more enthusiasm than zoological knowledge, with the result that some very odd animals were to be found lurking in the Animalsfile shrubbery.

Unfortunately they are not in adjacent cages – sorry, records. The result is that large numbers of empty records in the file are filled either with zeros – if that part of the disc has not been used before – or with garbage from previously deleted files.

An INPUT command, even with error handling, does not take kindly to this situation, and the file-reading program grinds to a halt.

Another need to slip under DOS's guard arose from a completely different application. This involved the transfer of data to a computer file from records previously kept on cards with holes punched round the edges.

Now each hole obviously can be represented by a single bit of memory, and there would be enormous savings of memory if eight bits could be squashed into one byte before being sent as part of a string to a random-access text file.

A preliminary experiment showed that it did not work, as the highest bit gets ignored in the text file processing, but nevertheless seven bits can be packed in.

The snag comes later, as you may have guessed. The file record has been packed with characters which may be anything between CHR$(0) and CHR$(127) and very strange things happen when one tries to access the record with an INPUT command.

On the other hand, accessing the characters one by one with a GET command takes hours, which is not what computerising records is all about . . .

At this stage, I acquired a copy of Beneath Apple DOS, by Worth and Lechner, from the Leicester Computer Centre, together with some good advice from Mike Glover, and sat down with the Pascal assembler (see my article in *Apple User,* August 1984) to write the prototype of &DOSFILE.

By bypassing the Basic interpreter and the outer layers of DOS, it proved to be very fast indeed in the original application, reading and picking out relevant records from a total of 7,000 cards in less than 12 minutes.

The principle is simple. The file manager is called directly, after the file manager parameter list has been loaded with the appropriate details taken from the interface area, which has itself been primed by POKE calls from a Basic program – or from the keyboard in immediate mode.

Similarly, the buffer areas from which the file manager takes data to write to a disc record, and to which it returns data that it has read from a disc, are both open to inspection from within a program or directly.

Furthermore, the file manager itself does not care two hoots about whether the high bit is set or not in dealing with

〉

&0(filename)	Opens a file.
&N(numeric expression or numeric variable)	Establishes which record is going to be processed.
&R(filename)	Reads a record into the read buffer (RDBUF) from a disc.
&W(filename)	Writes a record from the write buffer (WRBUF) to a disc.
&C(filename)	Closes the file.
&L(filename)	Locks the file.
&U(filename)	Unlocks the file.
&T(filename1, filename2)	Changes the name of a file from filename1 to filename2 (reTitle).
&V(filename)	Verifies the file.
&D(filename)	Deletes the file.
&P	This routine, called without any parameters, pushes the contents of the read buffer into the write buffer to facilitate the re-arrangement of records.

Table I: &DOSFILE syntax

Address		Parameter	Default value		Use by &DOSFILE
Hex	*Dec*		*Hex*	*Dec*	
390	912	NAMEBUF	—	—	Filename buffer area
to					
3AD	941		—	—	
3AE	942	RECLEN	50	80	Length of record
3AF	943	VOL	0	0	Volume no. of disc to be accessed
3B0	944	DRIVE	1	1	Drive to be accessed
3B1	945	SLOT	6	6	Slot to be accessed
3B2	946	FILETYPE	0	0	Type of file
3B3	947	RNUM	0	0	Set by &N()
3B4	948	RNUM+1	0	0	command
3B5	949	RTCODE	0	0	See text

Table II: &DOSFILE interface details

text files, but will faithfully guard whatever it is given, and produce it again on demand.

In the current version of &DOSFILE, which has been assembled using the DOS Toolkit assembler, we have total control over any bit of any byte in any file.

Calls to the routines are made via the & parser to simplify the passing of the name parameter and to avoid the use of the CALL command with various unintelligible numbers.

The coding is, however, laid out in an "open plan" fashion, so that the & parser may be bypassed if the user wishes to load parameters himself before calling the file manager command routines directly.

The filename fetcher is a very versatile routine and will accept the name of any previously-defined string or numerical variable, such as A or A$, or a number, say 123, or a string literal within quotes: "123 IS NOW A LEGAL FILENAME".

By suitably defining a title string, catalogues on your discs may now contain titles with flashing and inverse characters. Processing files with embedded control characters in their titles becomes much easier.

The syntax is as shown in Table I.

Example 1

The first 12 records in a file named ONE are to be transferred to a file named 2, starting at record 20. Note that the first record in a file is number 0.

This series of commands may also be entered in immediate mode from the keyboard and

```
100 &O("ONE"): &O(2)
110 FOR R=0 TO 11
120 &N(R): &R("ONE")
130 &P
140 &N(R+20): &W(2)
150 NEXT
160 &C("ONE"): &C(2)
```

will execute.

If the filename parameter is not given the system operates on the last given filename. This is not, however, to be tried out with the &T command, as the write buffer is used to house the second name and the results may be somewhat unexpected.

Example 2

It is required to delete the 10th record in a file of 20 records. The file is named RUBBISH, and the string variable R$ has already been assigned that value by:

```
100 R$="RUBBISH"
```

The deletion is achieved by moving records number 10-19 back one place:

```
10 WRBUF=38672: REM DEFINE
   START OF WRITE BUFFER
100 INPUT A$: REM REQUIRED
   STRING
110 A$=A$+CHR$(13): REM
   CLEARLY DEFINE END OF
   STRING
120 FOR A=1 TO LEN(A$)
130 POKE WRBUF + A -
   1,MID$(A$,1,A)
140 NEXT
```

Note that &C without any parameters closes only the one file, unlike the DOS CLOSE command.

In practice it is recommended that record 0 be used to keep data about the file itself, including the length of the record and the number of records currently in the file.

There are three interfaces to these file routines from Basic or the keyboard.

The write and read buffers, WRBUF and RDBUF, are located at $9710 and $9760 (decimal 38672 and decimal 38752) respectively, and can accommodate up to 80 bytes each in the current arrangement.

Values may be POKEd into the write buffer area and PEEKed from the read buffer as required.

In a later article I will show how to compress strings and enter them into the write buffer. In the meantime strings can be entered into the write buffer in the following way:

```
&O(R$): FOR L = 10 TO 19:
   &N(L): &R: &P: &N(L-1):
   &W: NEXT: &C
```

and extracted from the read buffer in a similar way. The reading of short strings from the read buffer is made much easier if the write routine includes a line similar to line 110 above.

The third interface is for the parameter record length (2 bytes), volume, drive, slot, and filetype into which appropriate values may be POKEd. There is no compelling reason for locating this interface in page 3, but I prefer POKEing to low-numbered addresses when possible.

Details of this interface are given in Table II.

The codes for various filetypes are:

Hex	*Dec*	
0	0	text
1	1	Integer Basic
2	2	Applesoft Basic
4	4	binary
10	16	relocatable (used by Toolkit assembler)
Three other types, recognised by DOS but not used, are:		
8	8	S type
20	32	A type
40	64	B type

The table of file types on Pages 6-10 of Worth and Lechner's book is slightly in error.

&DOSFILE commands do not stop the program when an error is encountered, so it is important to inspect the contents of RTCODE (address $3B5, decimal 949), ideally after each important disc access. If the value is 0, all is well.

The significance of other values is listed in Worth and Lechner on Pages 6-8. An almost identical list is found in the DOS Manual on Page 114 (ONERR GOTO codes).

The start of the &DOSFILE listing is shown in Listing I. Readers will have no difficulty if they copy in the machine code directly, but those using an assembler will note references in INDEXTBL to non-existent labels.

These labels are attached to routines which are provided in later articles in this series.

In the meantime, assembler users should type the series of dummy routines at the end of Listing I.

```
                2 ******************************************
0000:           3 *
0000:           4 * DOSFILE  COPYRIGHT PHP HARRIS 1985
0000:           5 *    FUNCTIONS:
0000:           6 *1. READ AND WRITE DOS FILES OF ALL TYPES
0000:           7 *2. INITIALISE AND CATALOGUE DISCS
0000:           8 *3. COMPRESSION,EXPANSION, AND COMPARISON OF DATA
0000:           9 *4. "RAMDISC" FUNCTION
0000:          10 *
0000:          11 *  PART 1 OF LISTING
               12 ******************************************

0000:          14          DSECT
0000:          15 PTR      DS   2
0002:          16 BUFP     DS   2
0004:          17 TEMP     DS   2
0006:          18 NBZ      DS   2
0008:          19          DEND

0011:          21 VALTYP   EQU  $11
0050:          22 LINNUM   EQU  $50
006D:          23 STREND   EQU  $6D
0083:          24 VARPNT   EQU  $83
009B:          25 LOWTR    EQU  $9B
00A0:          26 FACMO    EQU  $A0
00B1:          27 CHRGET   EQU  $B1
00B7:          28 CHRGOT   EQU  $B7

0000:          30          DSECT
0390:          31          ORG  $390       BASIC/&DOSFILE INTERFACE
0390:          32 NAMEBUF  DS   30
03AE:          33 RECLN    DS   1
03AF:          34 VOL      DS   1
03B0:          35 DRIVE    DS   1
03B1:          36 SLOT     DS   1
03B2:          37 FILTYPE  DS   1
03B3:          38 RNUM     DS   2
03B5:          39 RTCODE   DS   1
03B6:          40 NBUFX    DS   2          >NAMEBUF,<NAMEBUF
03B8:          41 WRX      DS   2          >WRBUF,<WRBUF
03BA:          42 RDX      DS   2          >RDBUF,<RDBUF
03BC:          43 COX      DS   2          >COMP,<COMP
03BE:          44 LIX      DS   2          >LINE,<LINE
03C0:          45 CLIX     DS   2          >CLINE,<CLINE
0000:          46          DEND

0000:          48          DSECT
9710:          49          ORG  $9710
9710:          50 WRBUF    DS   80
9760:          51 RDBUF    DS   80
97B0:          52 LINE     DS   640
9A30:          53 COMP     DS   80
9A80:          54 CLINE    DS   640
0000:          55          DEND

03D2:          57 BUFSTAR  EQU  $3D2

03D6:          58 FM       EQU  $3D6
03DC:          59 LOCFPL   EQU  $3DC
03F6:          60 AMPER    EQU  $3F6
AFF7:          61 RDVTOC   EQU  $AFF7
AFFB:          62 WRVTOC   EQU  $AFFB
B011:          63 RDDIR    EQU  $B011
B230:          64 NXTDIR   EQU  $B230
DD67:          65 FRMNUM   EQU  $DD67
DD7B:          66 FRMEVL   EQU  $DD7B
DEB8:          67 CHECKR   EQU  $DEB8
DEBB:          68 CHECKL   EQU  $DEBB
DEBE:          69 CHKCOM   EQU  $DEBE
DEC9:          70 SNERR    EQU  $DEC9
DFE3:          71 PTRGET   EQU  $DFE3
E2DE:          72 FRE      EQU  $E2DE
E3E7:          73 STRLIT   EQU  $E3E7
E600:          74 FREFAC   EQU  $E600
E752:          75 GETADR   EQU  $E752
ED34:          76 FOUT     EQU  $ED34
F7D9:          77 GETARYPT EQU  $F7D9
FBDD:          78 BELL     EQU  $FBDD
----- NEXT OBJECT FILE NAME IS DOSFILE(1).OBJ0
9000:          79          ORG  $9000
9000:A9 00     80 INITVECT LDA #>INDEX
9002:8D F6 03  81          STA  AMPER
9005:A9 90     82          LDA  #<INDEX
9007:8D F7 03  83          STA  AMPER+1
900A:A0 13     84          LDY  ##13
900C:B9 1D 90  85 IN1      LDA  PARMS,Y
900F:99 AE 03  86          STA  RECLN,Y
9012:88        87          DEY
9013:10 F7     88          BPL  IN1
9015:A9 8F     89          LDA  #<INDEX-$100 REBUILD DOS BUFFERS
9017:8D 01 9D  90          STA  $9D01          TO PROTECT
901A:4C D4 A7  91          JMP  $A7D4          NEW CODE

901D:          93 *PARAMETERS FOR 1ST 80 BYTES OF TEXT FILE
901D:          94 * ON DISC IN DRIVE 1  SLOT 6 - ANY VOLUME NUMBER
901D:50 00 01  95 PARMS    DFB  $50,0,1,6,0,0,0,0
9020:06 00 00
9023:00 00
9025:90 03     96 NBUFY    DFB  >NAMEBUF,<NAMEBUF
9027:10 97     97 WRY      DFB  >WRBUF,<WRBUF
9029:60 97     98 RDY      DFB  >RDBUF,<RDBUF
902B:30 9A     99 COY      DFB  >COMP,<COMP
902D:80 97    100 LIY      DFB  >LINE,<LINE
902F:80 9A    101 CLIY     DFB  >CLINE,<CLINE

----- NEXT OBJECT FILE NAME IS DOSFILE(1).OBJ1
9000:          103         ORG  $9000
9000:38        104 INDEX   SEC
9001:E9 C1     105         SBC  #'A
9003:0A        106         ASL  A
9004:AA        107         TAX
9005:E8        108         INX
9006:BD 12 90  109         LDA  INDEXTBL,X PUSH
9009:48        110         PHA             ROUTINE
900A:CA        111         DEX              ADDRESS
900B:BD 12 90  112         LDA  INDEXTBL,X   ON STACK
900E:48        113         PHA
900F:4C B1 00  114         JMP  CHRGET    ADVANCE TEXTPTR AND
9012:          115 *                        "RETURN" TO SELECTED ROUTINE

9012:CB DE     117 INDEXTBL DW  SNERR-1   A
9014:CB DE     118         DW   SNERR-1   B
9016:D8 91     119         DW   CLOSE-1   C
9018:CF 90     120         DW   DELETE-1  D
901A:51 92     121         DW   EXPAND-1  E
901C:54 92     122         DW   CATALOG-1 F
901E:CB DE     123         DW   SNERR-1   G
9020:CB DE     124         DW   SNERR-1   H
9022:57 92     125         DW   INIT-1    I
9024:5A 92     126         DW   JOIN-1    J
9026:5D 92     127         DW   COMPRES-1 K
9028:D2 90     128         DW   LOCK-1    L
902A:60 92     129         DW   MEMFILE-1 M
902C:15 92     130         DW   DECHEX-1  N
902E:E7 90     131         DW   OPEN-1    O
9030:30 92     132         DW   RDTOWRBF-1 P
9032:63 92     133         DW   QUICKREAD-1 Q
9034:89 91     134         DW   READ-1    R
9036:66 92     135         DW   SPLIT-1   S
9038:9B 90     136         DW   RETITLE-1 T
903A:D5 90     137         DW   UNLOCK-1  U
903C:DB 90     138         DW   VERIFY-1  V
903E:AF 91     139         DW   WRITE-1   W
9040:69 92     140         DW   XCHECK-1  X
9042:6C 92     141         DW   JCOMP-1   Y
9044:6F 92     142         DW   ZERO-1    Z

               144 ********************************
9046:          145 *
9046:          146 * FILE-HANDLING ROUTINES
9046:          147 *
               148 ********************************

9046:AD B6 03  150 FILEADR LDA  NBUFX
9049:85 06     151         STA  NBZ
904B:AD B7 03  152         LDA  NBUFX+1
904E:85 07     153         STA  NBZ+1
9050:60        154         RTS
9051:F0 0C     155 FILENAME BEQ FPL
9053:20 46 90  156         JSR  FILEADR
9056:20 BB DE  157         JSR  CHECKL
9059:20 67 90  158         JSR  GETNAME
905C:20 BB DE  159         JSR  CHECKR
905F:20 DC 03  160 FPL     JSR  LOCFPL
9062:84 00     161         STY  PTR
9064:85 01     162         STA  PTR+1
9066:60        163         RTS

9067:20 7B DD  165 GETNAME JSR  FRMEVL

906A:24 11     166         BIT  VALTYP
906C:30 06     167         BMI  GN2
906E:20 34 ED  168         JSR  FOUT
9071:20 E7 E3  169         JSR  STRLIT
9074:20 00 E6  170 GN2     JSR  FREFAC

9077:          172 * CLEAR FILENAME AREA
9077:A9 A0     173         LDA  ##A0
9079:A0 1D     174         LDY  ##1D
907B:91 06     175 CL1     STA  (NBZ),Y
907D:88        176         DEY
907E:10 FB     177         BPL  CL1

9080:          179 * TRANSFER NAME STRING TO BUFFER
9080:A0 00     180         LDY  #0
9082:B1 A0     181         LDA  (FACMO),Y STRING LENGTH
9084:48        182         PHA              -> STACK
9085:C8        183         INY
9086:B1 A0     184         LDA  (FACMO),Y INDEX TO ASCII CODE
9088:85 00     185         STA  PTR         OF STRING
908A:C8        186         INY
908B:B1 A0     187         LDA  (FACMO),Y TRANSFERRED
908D:85 01     188         STA  PTR+1       TO PTR
908F:68        189         PLA
9090:A8        190         TAY              LENGTH -> Y
9091:88        191         DEY
9092:B1 00     192 MV1     LDA  (PTR),Y   FIND ASCII CODE
9094:09 80     193         ORA  ##80      SET HIGH BIT
9096:91 06     194         STA  (NBZ),Y    STORE IN
9098:88        195         DEY             FILENAME
9099:10 F7     196         BPL  MV1
909B:60        197         RTS

909C:20 46 90  199 RETITLE JSR  FILEADR   GET
909F:20 BB DE  200         JSR  CHECKL     FIRST FILENAME
90A2:20 67 90  201         JSR  GETNAME     TO
90A5:20 BE DE  202         JSR  CHKCOM       NAMEBUF

90A8:          204 * GET SECOND FILENAME TO WRBUF
90A8:AC B8 03  205         LDY  WRX
90AB:84 06     206         STY  NBZ
90AD:AC B9 03  207         LDY  WRX+1
90B0:84 07     208         STY  NBZ+1
90B2:20 67 90  209         JSR  GETNAME    2ND FILENAME NEEDED
90B5:20 B8 DE  210         JSR  CHECKR
90B8:20 5F 90  211         JSR  FPL
90BB:A0 00     212         LDY  #0
90BD:A9 09     213         LDA  #9         RENAME CODE
90BF:91 00     214         STA  (PTR),Y
90C1:A0 02     215         LDY  #2
90C3:AD B8 03  216         LDA  WRX
90C6:91 00     217         STA  (PTR),Y
90C8:C8        218         INY
90C9:AD B9 03  219         LDA  WRX+1
90CC:91 00     220         STA  (PTR),Y
90CE:D0 2D     221         BNE  OP1
```

```
90D0:A9 05     223 DELETE  LDA  #5          DELETE CODE
90D2:2C        224         DFB  $2C
90D3:A9 07     225 LOCK    LDA  #7          LOCK CODE
90D5:2C        226         DFB  $2C
90D6:A9 0B     227 UNLOCK  LDA  #$B          UNLOCK CODE
90D8:2C        228         DFB  $2C
90D9:A9 0C     229 VERIFY  LDA  #$0C         VERIFY CODE
90DB:48        230         PHA              CODE SAVED ON STACK
90DC:20 51 90  231         JSR  FILENAME
90DF:68        232         PLA              CODE RETRIEVED
90E0:A0 00     233 V1      LDY  #0
90E2:91 00     234         STA  (PTR),Y
90E4:A0 03     235         LDY  #3
90E6:D0 15     236         BNE  OP1

90E8:20 51 90  238 OPEN    JSR  FILENAME
90EB:A9 01     239         LDA  #1          OPEN CODE
90ED:A0 00     240         LDY  #0
90EF:91 00     241         STA  (PTR),Y
90F1:A0 02     242         LDY  #2          OFFSET RECORD LENGTH
90F3:AD AE 03  243         LDA  RECLN       RECORD LENGTH
90F6:91 00     244         STA  (PTR),Y
90F8:C8        245         INY              3
90F9:A9 00     246         LDA  #0          RECORD LENGTH <256
90FB:91 00     247         STA  (PTR),Y
90FD:C8        248 OP1     INY              4
90FE:AD AF 03  249         LDA  VOL
9101:91 00     250         STA  (PTR),Y
9103:C8        251         INY              5
9104:AD B0 03  252         LDA  DRIVE
9107:91 00     253         STA  (PTR),Y
9109:C8        254         INY              6
910A:AD B1 03  255         LDA  SLOT
910D:91 00     256         STA  (PTR),Y
910F:C8        257         INY              7
9110:AD B2 03  258         LDA  FILTYPE
9113:91 00     259         STA  (PTR),Y
9115:C8        260         INY              8
9116:AD B6 03  261         LDA  NBUFX
9119:91 00     262         STA  (PTR),Y
911B:C8        263         INY              9
911C:AD B7 03  264         LDA  NBUFX+1
911F:91 00     265         STA  (PTR),Y

9121:AD D2 03  267 FBUF    LDA  BUFSTAR
9124:85 03     268         STA  BUFP+1
9126:A0 00     269         LDY  #0
9128:84 02     270         STY  BUFP

912A:B1 02     272 FB1     LDA  (BUFP),Y    LOCATE NEXT DOS BUFFER
912C:48        273         PHA
912D:C8        274         INY
912E:B1 02     275         LDA  (BUFP),Y
9130:85 03     276         STA  BUFP+1
9132:68        277         PLA


9133:85 02     278         STA  BUFP
9135:D0 0D     279         BNE  FB2
9137:A5 03     280         LDA  BUFP+1
9139:D0 09     281         BNE  FB2

913B:A9 0C     283         LDA  #$0C        ERROR CODE- NO BUFFER FREE
913D:8D B5 03  284         STA  RTCODE
9140:20 DD FB  285         JSR  BELL
9143:60        286         RTS

9144:A0 00     288 FB2     LDY  #0
9146:B1 02     289         LDA  (BUFP),Y
9148:D0 19     290         BNE  COMPARE     BRANCH IF BUFFER IN USE
914A:B1 00     291         LDA  (PTR),Y     GET FM COMMAND
914C:C9 01     292         CMP  #1           IS IT "OPEN"?
914E:F0 08     293         BEQ  MARKBUF     IF SO, FILL IN NAME
9150:C9 05     294         CMP  #5          SET UP IF DELETE,RENAME
9152:10 24     295         BPL  FB3          LOCK, OR UNLOCK
9154:A0 24     296 NXTBUF  LDY  #$24        OTHERWISE
9156:D0 D2     297         BNE  FB1          LOOK AT NEXT BUFFER
9158:A0 1D     298 MARKBUF LDY  #$1D
915A:B1 06     299 MK1     LDA  (NBZ),Y
915C:91 02     300         STA  (BUFP),Y
915E:88        301         DEY
915F:10 F9     302         BPL  MK1
9161:30 15     303         BMI  FB3
9163:A0 1D     304 COMPARE LDY  #$1D
9165:B1 06     305 CM1     LDA  (NBZ),Y
9167:D1 02     306         CMP  (BUFP),Y
9169:D0 E9     307         BNE  NXTBUF      BRANCH IF NAME DIFFERS
916B:88        308         DEY
916C:10 F7     309         BPL  CM1
916E:C8        310         INY
916F:B1 00     311         LDA  (PTR),Y     GET FM COMMAND
9171:C9 02     312         CMP  #2          : IS IT "CLOSE"?
9173:D0 03     313         BNE  FB3
9175:98        314         TYA              IF SO THEN
9176:91 02     315         STA  (BUFP),Y     MARK BUFFER AS FREE
9178:A2 00     316 FB3     LDX  #0
917A:20 E5 91  317         JSR  FLMNGR
917D:90 03     318         BCC  RETURN
917F:20 DD FB  319 ERROR   JSR  BELL
9182:A0 0A     320 RETURN  LDY  #$0A
9184:B1 00     321         LDA  (PTR),Y
9186:8D B5 03  322         STA  RTCODE
9189:60        323         RTS

918A:20 51 90  325 READ    JSR  FILENAME
918D:A9 03     326         LDA  #3          READ CODE
918F:A0 00     327         LDY  #0
9191:91 00     328         STA  (PTR),Y
9193:20 FC 91  329         JSR  PRMTRS
9196:C8        330         INY              6
9197:AD AE 03  331         LDA  RECLN
919A:91 00     332         STA  (PTR),Y
919C:C8        333         INY              7
919D:A9 00     334         LDA  #0          RECORD LENGTH<256
919F:91 00     335         STA  (PTR),Y
91A1:C8        336         INY              8
91A2:AD BA 03  337         LDA  RDX
91A5:91 00     338         STA  (PTR),Y
91A7:C8        339         INY              9
91A8:AD BB 03  340         LDA  RDX+1
91AB:91 00     341         STA  (PTR),Y
91AD:4C 21 91  342         JMP  FBUF

91B0:20 51 90  344 WRITE   JSR  FILENAME
91B3:A9 04     345         LDA  #4          WRITE CODE
91B5:A0 00     346         LDY  #0
91B7:91 00     347         STA  (PTR),Y
91B9:20 FC 91  348         JSR  PRMTRS
91BC:C8        349         INY              6
91BD:38        350         SEC
91BE:AD AE 03  351         LDA  RECLN
91C1:E9 01     352         SBC  #1
91C3:91 00     353         STA  (PTR),Y
91C5:C8        354         INY              7
91C6:A9 00     355         LDA  #0          RECORD LENGTH<256
91C8:91 00     356         STA  (PTR),Y
91CA:C8        357         INY              8
91CB:AD B8 03  358         LDA  WRX
91CE:91 00     359         STA  (PTR),Y
91D0:C8        360         INY              9
91D1:AD B9 03  361         LDA  WRX+1
91D4:91 00     362         STA  (PTR),Y
91D6:4C 21 91  363         JMP  FBUF

91D9:20 51 90  365 CLOSE   JSR  FILENAME
91DC:A9 02     366         LDA  #2          CLOSE CODE
91DE:A0 00     367         LDY  #0
91E0:91 00     368         STA  (PTR),Y
91E2:4C 21 91  369         JMP  FBUF

91E5:A0 1E     371 FLMNGR  LDY  #$1E
91E7:B1 02     372 FMGR1   LDA  (BUFP),Y    GET 3 BUFFER PTRS
91E9:48        373         PHA
91EA:C8        374         INY
91EB:C0 24     375         CPY  #$24
91ED:90 F8     376         BCC  FMGR1
91EF:A0 11     377         LDY  #$11
91F1:68        378 FMGR2   PLA
91F2:91 00     379         STA  (PTR),Y
91F4:88        380         DEY
91F5:C0 0C     381         CPY  #$0C
91F7:B0 F8     382         BCS  FMGR2
91F9:4C D4 03  383         JMP  FM

91FC:C8        385 PRMTRS  INY              1
91FD:A9 04     386         LDA  #4          PDS+RANGE CODE
91FF:91 00     387         STA  (PTR),Y
9201:C8        388         INY              2
9202:AD B3 03  389         LDA  RNUM

9205:91 00     390         STA  (PTR),Y
9207:C8        391         INY              3
9208:AD B4 03  392         LDA  RNUM+1
920B:91 00     393         STA  (PTR),Y
920D:C8        394         INY              4
920E:A9 00     395         LDA  #0          BYTE OFFSET
9210:91 00     396         STA  (PTR),Y
9212:C8        397         INY              5
9213:91 00     398         STA  (PTR),Y     BYTE OFFSET
9215:60        399         RTS

9216:D0 03     401 DECHEX  BNE  D1
9218:4C C9 DE  402         JMP  SNERR
921B:20 BB DE  403 D1      JSR  CHECKL
921E:20 67 DD  404         JSR  FRMNUM
9221:20 52 E7  405         JSR  GETADR
9224:A5 50     406         LDA  LINNUM
9226:8D B3 03  407         STA  RNUM
9229:A5 51     408         LDA  LINNUM+1
922B:8D B4 03  409         STA  RNUM+1
922E:4C BB DE  410         JMP  CHECKR

9231:AD BA 03  412 RDTOWRBF LDA RDX
9234:85 00     413         STA  PTR
9236:AD BB 03  414         LDA  RDX+1
9239:85 01     415         STA  PTR+1
923B:AD B8 03  416         LDA  WRX
923E:85 02     417         STA  BUFP
9240:AD B9 03  418         LDA  WRX+1
9243:85 03     419         STA  BUFP+1
9245:A0 00     420         LDY  #0
9247:B1 00     421 MB4     LDA  (PTR),Y
9249:91 02     422         STA  (BUFP),Y
924B:C8        423         INY
924C:CC AE 03  424         CPY  RECLN
924F:90 F6     425         BCC  MB4
9251:60        426         RTS

9252:          428 * DUMMY LABELS
9252:4C C9 DE  429 EXPAND  JMP  SNERR
9255:4C C9 DE  430 CATALOG JMP  SNERR
9258:4C C9 DE  431 INIT    JMP  SNERR
925B:4C C9 DE  432 JOIN    JMP  SNERR
925E:4C C9 DE  433 COMPRES JMP  SNERR
9261:4C C9 DE  434 MEMFILE JMP  SNERR
9264:4C C9 DE  435 QUICKREAD JMP SNERR
9267:4C C9 DE  436 SPLIT   JMP  SNERR
926A:4C C9 DE  437 XCHECK  JMP  SNERR
926D:4C C9 DE  438 JCOMP   JMP  SNERR
9270:4C C9 DE  439 ZERO    JMP  SNERR

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Banish those electricity bill blues

CHRIS BURRIDGE introduces a custom designed spreadsheet model – and gives some handy general model building hints

THE sound of electricity, gas or telephone bills popping through the letterbox usually guarantees a racing pulse and dive for the drinks cabinet. Thankfully the event is less distressing these days as many folk pay on a monthly budget.

Even so, the dreaded settlement quarter can provide a nasty shock to the system, particularly if you'd not anticipated that unthinkable – a large deficit!

One way to avoid this is by proper budgeting. It saves ulcers to ensure your monthly repayment standing order is roughly adequate – bearing in mind the effects of inflation and seemingly endless increases in tariffs. On the other hand you'd be foolish to over-budget as you'll get no interest.

A spreadsheet model designed to anticipate just the right balancing figure and provide a long term record is examined here – Figures I and II.

This was produced on Multiplan with a standard Mac but the underlying principles apply equally to other spreadsheet programs. In fact, the model is a re-hash and considerable enhancement of a Visicalc model constructed on my previous Apple IIe.

General terminology will be used rather than that specific to Multiplan as spreadsheets like Visicalc and Magicalc use a different method of referencing cells. Where this can't be avoided the convention will be to state Column X, Row Y.

For those of you scratching your heads over the Multiplan cell references, think of them as telling you the map reference *from* the current active cell. For instance, (R[–9]C[–2]) refers without fuss to the cell nine rows up and two columns back.

Although nominally covering electricity, gas and telephone the model can quickly be adapted for solid fuel or even car expenditure, when monthly figures would be more appropriate. The reader is assumed to have some basic knowledge of spreadsheets.

You'll see that the model covers a three year span on a quarterly basis. Unit and Expenditure percentage differences are shown, together with the highest and lowest value quarters and the running average of quarter, week or month.

It goes without saying that the average expenditure per month is an important statistic as this should loosely relate to your budget repayment standing order.

The object and clever part of the model is contained in the projection lines. You simply enter the budget account balance of your latest bill where indicated next to the arrow. Then hey-presto – a projection for the current year's account balance appears by magic in the bottom box.

The good or bad tidings are easily divided by the number of months of your next settlement account. The result is the estimated monthly repayment adjustment needed to achieve parity.

And so to the details of setting up the model. Visicalc

HEATING/LIGHTING & TELEPHONE TABLES

ELECTRICITY ACCOUNT

QUARTER	Units 1984	Bill £ 1984	Units 1985	Bill £ 1985	Units 1986	Bill £ 1986	DIFFERENCE between THIS Year & LAST Year: Units	Bill £	Units%	Bill £%
**										
1. JAN - MAR	1384	73.71	1826	96.75			442	23.04	31.9%	31.3%
2. APR - JUN	1460	78.82					0	0.00	0.0%	0.0%
3. JULY-SEPT	1878	99.30					0	0.00	0.0%	0.0%
4. OCT - DEC	1922	101.46					0	0.00	0.0%	0.0%
YR. TOTAL	6644	353.29	1826	96.75	0	0.00	442	23.04	31.9%	31.3%
HIGH Quarter	1922	101.46	1826	96.75	0	0.00	442	23.04	-5.0%	-4.6%
LOW Quarter	1384	73.71	1826	96.75	0	0.00	0	0.00	31.9%	31.3%
AVERAGE Qtr.	1661	88.32	1826	96.75	0	0.00	111	5.76	9.9%	9.5%
Average WEEK	128	6.79	140	7.44	0	0.00	9	0.44	9.9%	9.5%
									0.0%	0.0%
Ave. MONTH	554	29.44	609	32.25	0	0.00	37	1.92	9.9%	9.5%
PROJECTIONS:	CURRENT YEAR --->		7349	398.70	0	0.00	705	45.41	10.61%	12.85%
	Enter Budget Account Balance of LATEST ELECTRICITY Bill --->							0.00		
	Computer PROJECTION is that YEAR END ELECTRICITY Balance will be							-38.70		

GAS ACCOUNT

QUARTER	Therms 1984	Bill £ 1984	Therms 1985	Bill £ 1985	Therms 1986	Bill £ 1986	DIFFERENCE between THIS Year & LAST Year: Therms	Bill £	Therms%	Bill £%
**										
1. JAN - MAR	350	129.38	378	142.75			28	13.37	8.0%	10.3%
2. APR - JUN	134	57.22					0	0.00	0.0%	0.0%
3. JULY-SEPT	33	21.55					0	0.00	0.0%	0.0%
4. OCT - DEC	142	59.76					0	0.00	0.0%	0.0%
YR. TOTAL	659	267.91	378	142.75	0	0.00	28	13.37	8.0%	10.3%
HIGH Quarter	350	129.38	378	142.75	0	0.00	28	13.37	8.0%	10.3%
LOW Quarter	33	21.55	378	142.75	0	0.00	0	0.00	1045.5%	562.4%
AVERAGE Qtr.	164.8	66.98	378	142.75	0	0.00	7	3.34	129.4%	113.1%
Average WEEK	13	5.15	29	10.98	0	0.00	1	0.26	129.4%	113.1%
									0.0%	0.0%
Ave. MONTH	55	22.33	126	47.58	0	0.00	2	1.11	129.4%	113.1%
PROJECTIONS:	CURRENT YEAR --->		702	292.36	0	0.00	43	24.45	6.59%	9.13%
	Enter Budget Account Balance of LATEST G A S Bill --->							-22.23		
	Computer PROJECTION is that YEAR END G A S Balance will be							-14.59		

Figure I: Complete model relating to electricity and gas – in cell protected mode

TELEPHONE ACCOUNT

QUARTER	Units 1984	Bill £ 1984	Units 1985	Bill £ 1985	Units 1986	Bill £ 1986	DIFFERENCE between THIS Year & LAST Year: Units	Bill £	Units%	Bill £%
**										
1. JAN - MAR	299	31.40	972	42.61			673	11.21	225.1%	35.7%
2. APR - JUN	320	32.46					0	0.00	0.0%	0.0%
3. JULY-SEPT	243	28.97					0	0.00	0.0%	0.0%
4. OCT - DEC	1177	80.08					0	0.00	0.0%	0.0%
YR. TOTAL	2039	172.91	972	42.61	0	0.00	673	11.21	225.1%	35.7%
HIGH Quarter	1177	80.08	972	42.61	0	0.00	673	11.21	-17.4%	-46.8%
LOW Quarter	243	28.97	972	42.61	0	0.00	0	0.00	300.0%	47.1%
AVERAGE Qtr.	509.8	43.23	972	42.61	0	0.00	168	2.80	90.7%	-1.4%
Average WEEK	39	3.33	75	3.28	0	0.00	13	0.22	90.7%	-1.4%
									0.0%	0.0%
Ave. MONTH	170	14.41	324	14.20	0	0.00	56	0.93	90.7%	-1.4%
PROJECTIONS:	CURRENT YEAR --->		2799	195.44	0	0.00	760	22.53	37.27%	13.03%
	Enter Budget Account Balance of LATEST TELEPHONE Bill --->							-28.71		
	Computer PROJECTION is that YEAR END TELEPHONE Balance will be							-44.15		

NOTES:
1. Projected Balance weighted to take 'pessimistic view'. 5% & 8% added to Units /£'s 'blind' figs respectively
2. Average Quarter is calculated on actual entries, whereas Difference Columns are true averages.
3. Formula in Computer Projection Box contains normal monthly Budget for each Utility.
4. Minus figures must be preceded by a '–' eg. Bill balance –£23.19.
5. Figs for each Quarter ** relate to actual Qtr consumption took place (& Billed in subsequent Qtr).

LOGIC :
1. CURRENT YEAR PROJECTION = If This yr total Fig <1 – take last years total +% loading as above OTHERWISE [(last yr tot – this yr tot + Diff) + % loading] + (last yr + Diff). This works with negative figs.
2. When all 4 Qtrs tallied automatically gives Yr total, as 0 Diff + % loading = 0
3. For 1986 columns only, If (this yr tot + Qtr 4) > 1 then add % loading to last yr tot (as nothing tallied yet this yr) OR If this Yr tot < 1 put 0 OTHERWISE calculate formulae. (This prevents model showing a projection for 1986 ... before all figs for 1985 are in)
4. Projected Difference and % differences are straight comparisons of current & previous Yr (Actual or Projection.).
5. Gives projection even when no figs in at all yet this yr. See 1. above. & Loading is inflation buffer.
6. Projection formulae = simplified form of [(Last Yr – This Yr + Diff) + (Last Yr –This Yr + Diff + % loading) +This Yr].
7. Statistical Formulas not used – only common sense and experience!

Figure II: Remainder of full model showing telephone account (unprotected format)

users will know that setting up labels is a fairly tedious task. With Multiplan this is easier as you may type your entire line of text into one cell and it spills over automatically into adjacent cells.

Additionally Multiplan, like Magicalc, allows variable width columns offering a better presentation.

Whatever your software, please ensure that all columns are wide enough in global format to display figures generated by the program. This advice applies particularly to the percentage columns.

The usual mathematical trick can be used to produce the numbered labels. For example, 1985 is "cell 1984 + 1" and this relative reference formula can then be copied or replicated – filled right in Multiplan – to produce 1986.

The aim throughout has been to keep formulae as simple as possible. Those in the Year Total lines are fairly self explanatory, being the sum of the four quarter bill cells above. The annual differences and percentages are a simple comparison between the previous year and the present year.

Note that all formulae are copied relatively into the remaining columns and down for each quarter.

One feature of the model is that if nothing is inserted yet this year no difference will show. The model is also smart enough to automatically switch the formula when 1986 data is entered so that it compares with 1985 and not 1984.

Figure III shows the formula designed to do this in Column 8 Row 11. Note particularly the double nested IF Functions @IF in Visicalc). The percentages are similarly dealt with and take advantage of Multiplan's percentage cell format – Just include *100 in a Visicalc formula. The rather awe-inspiring but efficient formula in column 11 row 11 is:

=IF (RC[–6]=0, 0, IF (RC[–4] =0, (RC[–6]–RC[8])/RC[–8], (RC[–4]–RC[–6])/RC[–6]))

This works as follows: IF no data is entered in 1985 show zero, THEN IF no entries made in 1986 express % difference between the same period in 1984 and 1985 OTHERWISE calculate similarly between 1986 and 1985.

Incidentally, when pondering over similar long formulae yourself always build in small tested blocks – with each part self-contained within its own brackets.

Start by writing down the logic in English. It's then easy to nest one formula inside another. Don't be afraid of the very convenient IF Logic Function. All it means is: IF a condition occurs – such as <1 – then do a calculation OTHERWISE perform another alternative.

Remember to always include the condition and options within brackets separated by commas. Once the penny's dropped you'll find it a breeze.

Moving on, the High and Low quarter figures rely on standard spreadsheet functions for picking out the MAX and MIN of the range.

Although the AVErage calculations also appear obvious they only work on data actually entered.

The formula does not divide by four each time so that the true average is changing all the time – a feature harnessed to our benefit in this model.

We'll now turn to the raison d'être for the model – the Projection cells. These were the most complex and time-consuming part of the model to devise but also the most rewarding.

Before examining the formulae, you firstly need to ensure that your current monthly repayment standing order is included in the formula at Column 9 Row 27 – the actual projection cell.

This can be edited – with extreme care – on the rare occasions necessary merely by selecting the cell and editing at the top.

This arrangement keeps things straightforward but will annoy spreadsheet purists who insist that models should only be capable of accepting data and never require any tampering with formulae.

The solution would be easy with Multiplan – just create an invisible column to house the working and formula.

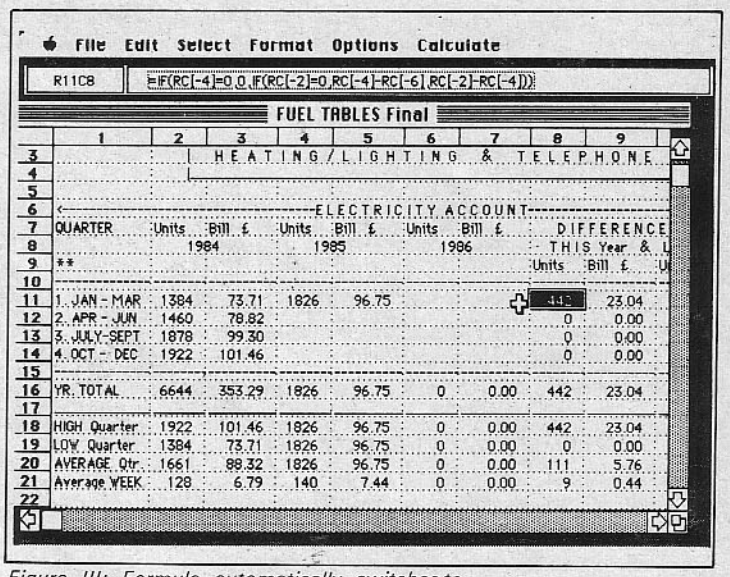


Figure III: Formula automatically switches to compare correct years – eg Col 8 Row 11

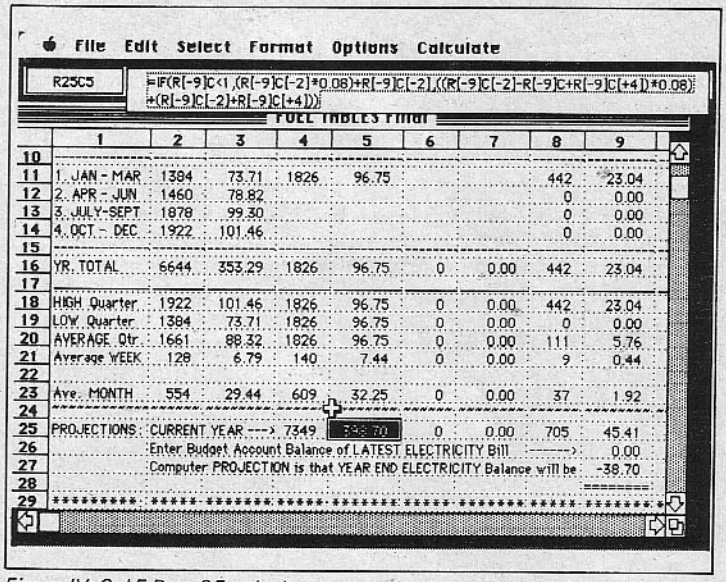


Figure IV: Col 5 Row 25 – the key current end-year projection algorithm

The key to the spreadsheet is the calculation of the current year projection – see Figure IV. What does this formula do? It attempts to combine adjusted historic precedent figures with this year's real expenditure.

This is done by incorporating alternative formulae with an IF logical function. Take a closer look at Figure IV which shows the full formula in column 5 row 25.

Boiled down this means: IF this year's total is nil (because no data has been entered), THEN calculate last year's total plus 8 per cent, OTHERWISE add this year's total so far – using the calculated differences – to figures for unknown quarters, which are assumed to be the same as last year plus 8 per cent.

Again formulae are copied relatively but observe, for reasons discussed later, that the percentage increase in the units columns is slightly lower at 5 per cent.

And so to the formula of the actual projection cell Column 9 Row 27 – see Figure V. Notice again that two different procedures are nested in an IF function.

What all this goes to show is: IF there is some 1986 £ data,

File Edit Select Format Options Calculate

R27C9 =IF(R[-2]C[-2]>0,R[-1]C+(30*12)-R[-2]C[-2],R[-1]C+(30*12)-R[-2]C[-4])

FUEL TABLES Final

	3	4	5	6	7	8	9	10	11
10									
11	73.71	1826	96.75			442	23.04	31.9%	31.3%
12	78.82					0	0.00	0.0%	0.0%
13	99.30					0	0.00	0.0%	0.0%
14	101.46					0	0.00	0.0%	0.0%
15									
16	353.29	1826	96.75	0	0.00	442	23.04	31.9%	31.3%
17									
18	101.46	1826	96.75	0	0.00	442	23.04	-5.0%	-4.6%
19	73.71	1826	96.75	0	0.00	0	0.00	31.9%	31.3%
20	88.32	1826	96.75	0	0.00	111	5.76	9.9%	9.5%
21	6.79	140	7.44	0	0.00	9	0.44	9.9%	9.5%
22								0.0%	0.0%
23	29.44	609	32.25	0	0.00	37	1.92	9.9%	9.5%
24									
25	YEAR --->	7349	398.70	0	0.00	705	45.41	10.61%	12.85%
26	dget Account Balance of LATEST ELECTRICITY Bill					------->	18.75		
27	r PROJECTION is that YEAR END ELECTRICITY Balance will be						[redacted]		
28									
29	********	********	********	********	********	********	********	********	********

Figure V: Notice the monthly repayment – £30 here – must be carefully included in the calculation. The result is the final balance projection

Fuelfables Protected

Therms 1984	Bill £ 1984	Therms 1985	Bill £ 1985	Therms 1986	Bill £ 1986	DIFFERENCE between THIS Year & LAST Year Therms	Bill £	Therms%	Bill £%
			GAS ACCOUNT						
350	129.38	378	142.75			28	13.37	8.0%	10.3%
134	57.22	[redacted]				0	0.00	0.0%	0.0%
33	21.55					0	0.00	0.0%	0.0%
142	59.76					0	0.00	0.0%	0.0%
659	267.91	378	142.75	0	0.00	28	13.37	8.0%	10.3%
350	129.38	378	142.75	0	0.00	28	13.37	8.0%	10.3%
33	21.55	378	142.75	0	0.00	0	0.00	1045.5%	562.4%
164.8	66.98	378	142.75	0	0.00	7	3.34	129.4%	113.1%
13	5.15	29	10.98	0	0.00	1	0.26	129.4%	113.1%
								0.0%	0.0%
55	22.33	126	47.58	0	0.00	2	1.11	129.4%	113.1%
CURRENT YEAR --->		702	292.36	0	0.00	43	24.45	6.59%	9.13%
Enter Budget Account Balance of LATEST G A S Bill						------->	-22.23		

Figure VI: Multiplan protected mode – only data can be entered

THEN add the budget account balance you've entered to the annual repayment (Monthly x 12) and deduct computer guestimate of 1986 year projection, OTHERWISE use 1985 figures similarly.

Once the Electricity account cells have been fully constructed to your satisfaction just copy them twice below. To complete the exercise make the necessary minor adjustments to labels and budget amounts to customise for your other utilities.

For the benefit and convenience of some readers who will not want to take the trouble to set up their own models I will gladly supply a copy in Mac/Multiplan format. Just send a 3½in Mac disc and nominal £2.50 handling charge to MacHowe, 69 The Dormers, Highworth, Swindon, Wilts SN6 7PB.

Now for those who are interested I will outline some of the theory behind the figures. In preparing the model the overriding aim has been to keep things simple. There are obviously many different ways of achieving similar results using more involved formats and algorithms.

The Projection algorithms are based entirely on common sense and actual experience over the past few years rather than on statistical conventions. Whenever data is entered or changed the whole model reflects this automatically, unless manual recalculation mode is selected – including an update of projections.

It's only necessary to output to printer for your hard copy. I recommend an ongoing quarterly record library is maintained.

Alternatively it would be easy to set up another model – which could be linked in Multiplan – to summarise all the various totals. This could run for, say, a 10 year span.

Another idea would be to save specific parts of the model in ongoing DIF Format – SYLK in Multiplan. You could then load into a separate compatible program such as Chart and produce highly visual graphs and pie charts. However that is another story which I hope to cover in a future article.

A nice touch with the formulae is that if nothing is entered this year then the whole of last year's figure is taken and adjusted. But if any data is entered this year all the unknown quarters receive adjustment.

Should all four quarters be known this year then it's clearly the sum of these you want. The hardest time for the model to project forward will be when only the first quarter's figures are known.

The algorithm then does its best but the result will become more accurate by simple averaging as more data is entered. This approach keeps things simple and gives a fairly realistic and sensitive projection in most cases.

Once the annual projection is known the model performs the final act and provides your likely year end balance. This is a simple matter of adding your present bill balance to your repayments and taking away what the model expects your expenditure to be.

"While bills inevitably rise due to inflation, units only increase if you burn more fuel!"

Negative figures reflect projected deficits and will hopefully prompt you to increase your standing order repayments.

As previously mentioned the current percentage adjustment varies between units and expenditure. This strategy recognises that while bills inevitably rise due to inflation or government dictate – units only increase if you burn more fuel!

This said, a "worst view" stance is prudent and accordingly units are increased by 5 per cent and expenditure by 8 per cent. It's easy to amend these, in the light of experience and carry out your own "What if?" trials to see how projections would alter if, say, electricity bills or units increased by X per cent.

On a general point, wherever formulae are entered these have been programmed to read zero if the cells they relate to contain no data. The format of most cells is either decimal to two places to cope with the money boxes or justified right with no decimals.

In the case of the percentage boxes Multiplan has a feature which allows the percentage character to print out each time, and again the calculation is to two decimal places.

Finally Multiplan allows a protective mode – see Figure VI – which prevents anybody altering boxes containing formulae. Areas where data should be entered are highlighted – the cursor conveniently jumps only in the underlined boxes.

Visicalc users should save their own template of the model as a reserve back-up, containing formulae and so on, but no data.

A final point to mention is that telephone bills are now issued with VAT deducted from your repayments. For the purpose of this model bills have been adjusted to adopt a common approach.

But I hear the postman approaching – it's probably my Access account. Now where's that drink?

College pioneers micro facilities

TONY LEAH reports on an investment that's paying off

LEADING the way in the provision of computer facilities for boys and girls up to the age of 18 is Bishop's Stortford College, Hertfordshire.

The independent school has nine Apple II systems, including two Apple IIcs, grouped together in a computer room for use by the upper school in a variety of subjects.

Ian Taylor, head of science and of the physics department, recommended the first Apple purchase in 1980 – a II+ with disc drive and monitor for use on the Nuffield physics course which has "lots of A level physics that is numerical and can therefore be put on to a computer".

Taylor wrote the software for the A level physics classes and when the Apple II was introduced it generated a great deal of interest.

More than 20 pupils started teaching themselves Basic programming in their activities time, and A level students started writing their own software and improving on Taylor's.

Since buying the first Apple system, Taylor has had a special computer budget each year which has enabled him to gradually expand the facility.

He decided to buy more Apple IIs partly because he found them an excellent tool for teaching his own subject.

"Instead of showing film of computer output I simply use the machine", he says.

He also considers them user and programmer friendly.

"In my experience it is easier to write software for an Apple than any other machine".

The Apple computer room at the College now provides students aged 13 upwards with a basic introduction to computers which they can pursue later on if they wish.

All boys from the age of 13 now undertake a compulsory computer appreciation course from which they learn elementary programming, computer applications, the history of computing and its implications for the future.

The course which is taught by Taylor and his colleague Colin Williams lasts one year and is done with a ratio of two pupils to one computer.

At this stage formal computer training is suspended until the sixth form where O level computer studies is now available. This year 14 sixth formers out of 70 opted to do computer studies for the "Alternative Ordinary Level" offered by the Oxford and Cambridge board.

The lack of a place for computers on the curriculum between the introductory course and O levels is made up for by the unlimited access to the Apple systems during the daily two and a half hour activities period.

Taylor estimates that about 20 per cent of the pupils take advantage of this opportunity on a regular basis while another 30 per cent do so occasionally.

He is in favour of an unstructured approach which enables pupils to explore the subject – and the machines – at their own pace.

"The only thing we lose this way is formality, but I believe we gain more than we lose", he said.

He is rigorous in his approach to the use of the Apple IIs, overseeing his pupils' efforts at Basic in an attempt to instill structure into their programs and forbidding computer games during activities time.

"The pupils' awareness tends to be of home computers. I make them aware of computers as serious application tools", says Taylor.

On the Apple IIe and IIc systems at the college pupils use word processing packages for writing essays, and then Appleworks as an introduction to other business applications and to the concept of integration.

Taylor is more than satisfied with the investment the college has made in the Apple systems, finding the IIe a natural and easy-to-use tool for learning physics and programming and looking to the IIc for different advantages.

"I find the serial ports very useful and I am very impressed with the mouse input device that we are using on one of the IIc machines. This opens up whole new areas, particularly in graphics applications", he said.

What to do with all those Page 3 routines

DAVID HAYNES offers an elegant solution for those with a 16k language card

AS an *Apple User* reader you may well have a large number of useful machine code programs gleaned from the magazine's pages.

A problem often arises, however, if you want to use more than one of them in any given Basic program. This is because so many of them are written to reside in Page 3 of memory – at $300 or 768 decimal onwards – and only one can do so at any time.

The obvious solution is to BLOAD each one from disc and CALL it as required. But the BLOAD is slow and it interrupts the flow of the program.

Another solution might be to relocate the routines in adjacent areas at the top of memory, alter HIMEM appropriately and CALL them from their new addresses.

However relocation usually entails modification of the routine to suit the new location. And useful memory for Basic programs is wasted with this solution.

This article describes a more elegant solution which makes use of the language card. This involves loading all your Page 3 routines on to it as they stand.

Basic can then CALL the short machine code program shown below to rapidly copy the routine of your choice from the language card into Page 3 where it belongs and execute it.

This CALL can be repeated as often as required to access any of your routines from within the same Basic program.

The machine code program P3 ROUTINES.OBJ0 will reside at $3AD – decimal 941. This allows your Page 3 routines to be up to 173 bytes in length. It also avoids using the range $3DO to $3FF which is used by Basic and DOS.

Each Page 3 routine should be loaded into the language card so that it begins at a page boundary. Forty eight different routines, numbered 0-47, can be accommodated starting at $D000.

Basic can CALL the routines by number or by name:

```
CALL 941,36
```

or

```
ROUTINE=36
CALL 941,ROUTINE
```

If the Page 3 routine needs parameters P1, P2, . . . then add them to the call like this:

```
CALL 941,36,P1,P2, . . .
```

Now for the loading of the language card. I suggest you prepare one large disc file of all the Page 3 routines you want to use.

One way to do this conveniently would be to enter the monitor – CALL –151 – BLOAD each Page 3 routine in turn – at $300 – and move it to a page above $2000 say, for the fifth routine, counting from 0.

```
*BLOAD ROUTINE4
*2400<300.3ACM
```

Finally BSAVE the file of routines using:

```
*BSAVE FILENAME,
A$2000,L$LENGTH
```

where LENGTH = $100 times the number of routines.

Your Basic program then must BLOAD FILENAME into the language card with some lines of code such as:

```
10 RM=12*4096+8*16+1
20 POKE RM,1:POKE RM,1
30 PRINT CHR$(4)"BLOAD
   FILENAME, A$D000"
```

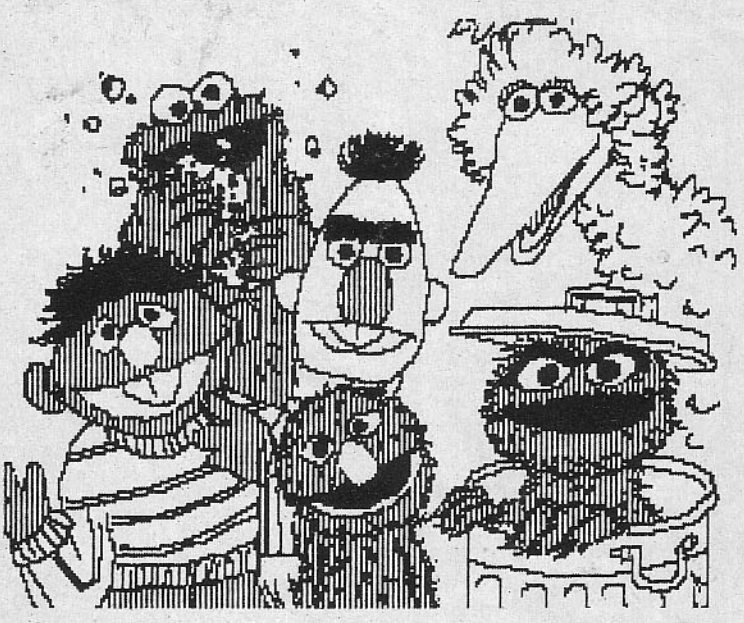
If you actually have more than 48 Page 3 routines then an extra 16 can be accommodated if you modify the P3 ROUTINES machine code program to use the other 4k bank of RAM from $D000 to $DFFF on the language card as well.

The disadvantage of this would be a limitation on the length of Page 3 routines which could be handled because P3 ROUTINES.OBJ0 would then take up more space.

The method described works very quickly, requires no modification of your Page 3 routines and uses none of the RAM your Basic programs might need. I hope you find it useful.

```
SOURCE FILE: P3 ROUTINES
C080:              1 RAMRD   EQU  $C080      ;Read enable RAM on 16k card
C082:              2 ROMRD   EQU  $C082      ;Read enable ROM on motherboard
DEBE:              3 EATCM   EQU  $DEBE      ;Eat comma
E6F8:              4 GETNUM  EQU  $E6F8      ;Value from BASIC line to X
D000:              5 DUMMY   EQU  $D000      ;Modified by this routine
0000:              6 ;
----- NEXT OBJECT FILE NAME IS P3 ROUTINES.OBJ0
03AD:              7         ORG  $3AD
03AD:              8 ;
03AD:20 BE DE      9         JSR  EATCM
03B0:20 F8 E6     10         JSR  GETNUM     ;Find page
03B3:8A           11         TXA
03B4:18           12         CLC
03B5:69 D0        13         ADC  #$D0
03B7:8D C1 03     14         STA  ADDR+2     ;Set DUMMY to address of routine
03BA:A2 00        15         LDX  #$00
03BC:AD 80 C0     16         LDA  RAMRD      ;Read enable RAM
03BF:BD 00 D0     17 ADDR    LDA  DUMMY,X    ;Modified address
03C2:9D 00 03     18         STA  $300,X     ;Target
03C5:E8           19         INX
03C6:E0 AD        20         CPX  #$AD       ;Max size of page 3 routine is $AD
03C8:D0 F5        21         BNE  ADDR
03CA:AD 82 C0     22         LDA  ROMRD      ;Read enable ROM
03CD:4C 00 03     23         JMP  $300       ;Execute page 3 routine

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```



Learning with the Sesame Street gang

SOME time ago Apple decided it was not going to release any more software under its own name. Rather, it would give more support to third party developers.

You may remember the Apple Special Delivery series that came to an end as a result of this decision. Personally, I didn't mourn the demise of that particular exercise.

However one series which Apple did "sponsor" and which deserved to do better was "Apple Presents . . .", produced in conjunction with the Children's Television Workshop (CTW).

If you know CTW at all the chances are that you relate it to the Sesame Street TV programmes. Not surprisingly, then, the four packages I'm about to describe feature Sesame Street characters.

You may wonder why I'm bothering to describe software that's been around for some time. The reason is that it's just been made readily available again in this country and at an amazingly low price.

On one of his trips to the States, Pete Fisher must have picked up a real bargain because P&P are offering these packages at £9.95 each. Considering that I recently saw the packages advertised at £34 each, the P&P price is very low.

The question is: Are the packages worth £9.95 each or is this a case of software dumping?

To start off with, the packages are nicely presented with beginner-level manuals which also include follow-up activities and additional information.

Each pack contains not only a master disc but also a separate back-up. This was one of the few aspects of the Special Delivery range which I appreciated, and is an excellent policy with software designed to be used by children.

The **Mix and Match** package is described as being appropriate for the whole family. It contains four separate games and a word editor.

The Mix and Match game itself allows the child to choose a Muppet head, body and legs to create a new Muppet. A new name is created as a pastiche of three names.

The Animal game is a version of Animals from the old DOS 3.3 Master Disc. It's nicely done with large text. The child can teach the program about various beasts and the new knowledge can be saved to the disc for future sessions.

Raise The Flags is an excellent implementation of Hangman, noteworthy for the fact that it is the only non-violent version I've ever encountered.

Letters are run up a set of flag-poles to make a word. It's really clear which letters have been used and the animation is simple but effective.

The program contains two lists, one containing food words and the other nature words. However this is where the Word Editor comes in. With it you can make up your own list of words to be used by the game.

The Layer Cake is a thinly-disguised Towers of Hanoi problem with three layers of cake and three plates. There are a few bells and whistles when you solve it.

The **Spotlight** package is described as being for 9 to 13 year olds. It comprises four games with a distinct educational flavour.

In Reflect you simulate shining a torch at a mirror. A paddle or joystick is used to change the angle of the mirror in order to do various things – light candles on a cake, hatch an egg and so forth. A performance score is given at the end in terms of number of times the target was hit and number of attempts.

Spotlight is effectively a more complicated version of Reflect. You are given 10 shots at shining the spotlight to hit Steve as he walks about a stage.

There's a screen blocking part of the stage and a fixed mirror which can be used in conjunction with the moveable mirror.

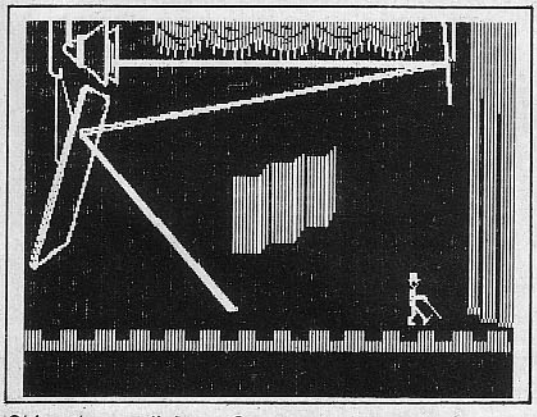
Again, the game is non-violent – when Steve's hit he performs his song and dance routine.

Hot Stuff is a Cows and Bulls variant which involves guessing three numbers and their order. The feedback is given in the form of "cold-warm-hot", with cold being a wrong number and hot being a number in its correct position.

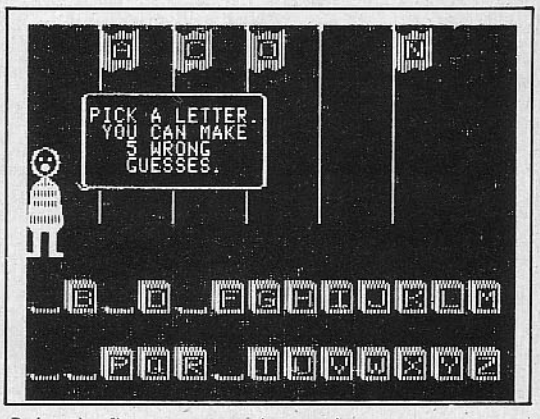
There's a bit of animation to add a little extra interest and the meaning of the clues is clearly stated.

The final game in this package is Boxed In, a 6 x 6 version of Reversi/Othello. Coordinates are used to specify moves. It doesn't play a very strong game but it's good enough for the target age-range.

The **Instant Zoo** package is aimed at 7 to 10 year olds.



Shine the spotlight on Steve



Raise the flags to guess the word

Again there are four games and a word editor.

The game of Instant Zoo is a lo-res animal guessing game. Random blocks of the screen are filled and the children have to press Esc when they want to guess the animal.

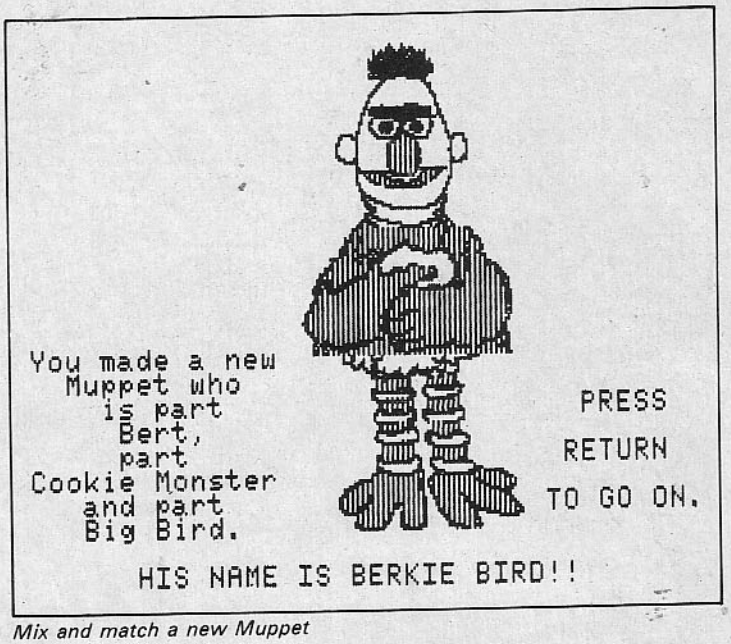
Some of the animals are clearer in colour, but most of them are recognisable in monochrome. My junior play-testers seemed to have no trouble with a green screen monitor.

Star Watch is a reaction time tester. Some stars are drawn on the screen and after a few moments one of them starts to move.

The child's task is to press the Space bar as soon as any movement is seen. The faster the reaction, the more stars appear on the screen.

Quick Match is a version of Snap based on words and incorporating a time factor. There are three levels of difficulty, and on the hard level the words are similar enough to be quite challenging – such as 'clan' and 'clam'.

Alternatively you can use the word editor to add your own list. The words are added in pairs and the program simply chooses one of the pair and displays it twice when it wants to 'snap'.



Mix and match a new Muppet

Scramble, as the name suggests, is an anagram game. The program contains two lists of words – sport and parts of the body – but once again the word editor can be used to input your own choice of words.

If you don't guess the answer within the time, the letters are rearranged into an order closer to the word. Points are related to the speed with which the answer is given.

The fourth package in the series, **Ernie's Quiz,** is aimed at 4 to 7 year olds.

The Guess Who game is very similar to Instant Zoo but this time the characters that appear are Muppets rather than animals. This has the disadvantage that you need to know the Muppets a little. If, like me, you can't tell your Bert from your Big Bird this game may give you a few problems the first few times through.

Of course the popularity of the Muppets means that most children would have less trouble than me with the game.

Jelly Beans is a counting game using Cookie Monster to reinforce right answers. The numbers may be too high – say 25 – for some 4 year olds but 7 year olds should have no problems.

Face-It is a sort of lo-res Identikit in which you build up a face from offered parts. Paddles, or a joystick, are necessary to cycle through the available bits, with Return being used to select a feature.

Ernie's Quiz starts by showing pictures of several Muppets. Word clues are then given and you must choose one of three Muppets on the basis of the clue. When you get the right answer the response is in keeping with the Muppet. For example, Ernie will play a trick on you and show himself on the text screen version of lo-res.

There is a nice sense of humour in all the packs. They make good use of the Apple's graphics and the only way they show their age is in sometimes taking a while to load. Once loaded, all four packages ran without a hitch.

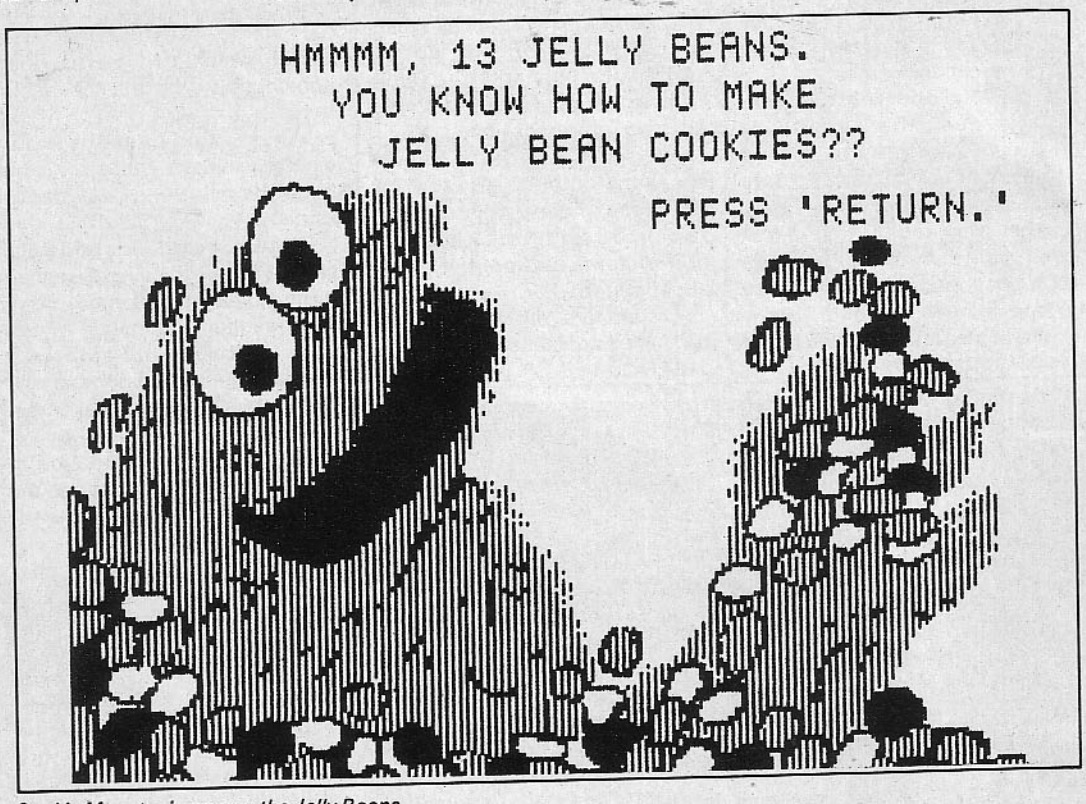
So, are the packages worth £9.95 each? I certainly think so. In fact, they'd be reasonably priced at twice that amount.

Although my children very rarely watch Sesame Street, even they knew enough to have fun with the Muppet-based games, and there's enough variety and challenge to keep everyone occupied.

All in all the packages display the thoughtful approach which typifies the Children's Television Workshop. I only wish more educational software was of such quality.

If Apple had sponsored more of this and less of the Special Delivery series it might not have been forced out of the software market.

Cliff McKnight

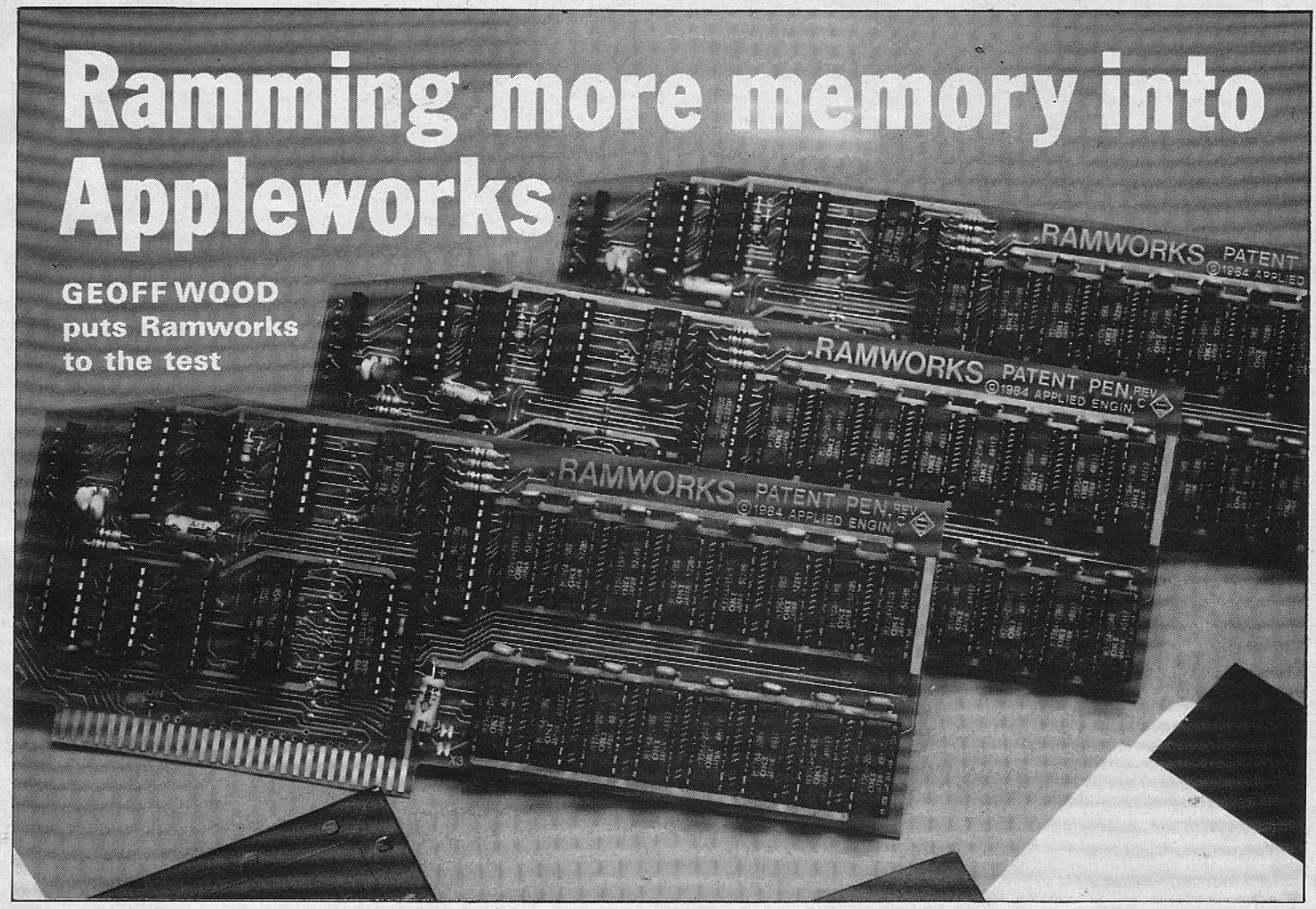


Cookie Monster in among the Jelly Beans

Titles: Mix and Match/ Spotlight/Instant Zoo/ Ernie's Quiz
Authors: Children's Television Workshop
Publisher: Apple Computer
Requirements: 48k Apple II family

Ramming more memory into Appleworks

GEOFF WOOD puts Ramworks to the test



APPLEWORKS offers three integrated programs – database, spreadsheet and word processor – for less than the price of three separate programs.

It can hold up to 12 different files in RAM simultaneously and easily switch between them. It is also easy to transfer information from the database or spreadsheet into the word processor.

Above all, Appleworks is easy to use and almost foolproof.

But it has some limitations. On a 128k Apple it offers only 55k of RAM in the Desktop for files, whereas Visicalc leaves 95k for a spreadsheet. The Appleworks database offers fast sorting and selection but only about 800 records in one file. The word processor offers 15 to 20 pages of typescript but some people would like more.

Another drawback with Appleworks is that if you want to switch frequently between spreadsheet, database and word processor, you have to wait for it to load in the relevant part of the program each time.

But now the scene has changed. Ramworks is an extended 80-column card made by Applied Engineering who specialise in Apple peripherals. It is slightly larger than the Apple 80-column card but it offers memory sizes up to 1mbyte.

Prices in Britain range from £299 for the 128k size to £499 for 256k, £799 for 512k and £1,199 for 1024k.

In turn these RAM sizes offer Appleworks Desktop sizes of 101k, 183k, 367k and 736k respectively.

If you create a file that is too large to fit on one disc (136k), Ramworks will automatically segment the file to spread it over more than one disc. Alternatively, large files can be saved on a hard disc.

The price of Ramworks includes a disc which modifies a copy of Appleworks to enable it to recognise the extra RAM. The modification process offers two options.

One version simply recognises the extra memory, the other automatically loads the three Appleworks programs into Ramworks at the boot stage so it can switch rapidly between programs. However the second version needs 256k or more on the Ramworks card.

Using a 256k version I found that Ramworks made a world of difference to Appleworks. I loaded in a database file of 500 records occupying 39k on disc and expanded it to 2,000 records by cutting and pasting.

I then tried sorting it into alphabetical or numerical order on various columns and found that the sorting times varied from 30 seconds to 55 seconds. Sorting the original file of 500 records took between seven and 12 seconds.

The Find command and the Selection command using three criteria took no more than 10 seconds.

Larger files may take much longer to perform these operations but Appleworks with Ramworks must be one of the fastest database programs available in terms of speed of sorting and selecting records from large files.

Larger files take proportionately longer times to save and load. My file of 2,000 records occupied 152k on two discs and took 100 seconds to save and 85 seconds to reload. The original version with 500 records took 27 seconds to save and 23 seconds to reload.

The spreadsheet too offers amazing capacities. Appleworks offers 999 rows and 127 columns (126,873 cells) but no hope of filling them all. With Ramworks I filled almost 16 columns of 999 rows with 5 digit numbers – 15,689 cells – before the memory ran out.

The normal version of Appleworks filled only four and a half columns, which is 4,428 cells.

Of course, formulae in the cells will reduce the number of usable cells but few people need as many cells as Ramworks offers. However large spreadsheets can be cumbersome and prone to error. The

〉

time to recalculate large spreadsheets can be frustrating unless you take a tea break.

Ramworks offers larger files for the word processor but the Appleworks program sets a limitation of 2,250 lines. I created a file with 2,250 lines and it ran to 39 pages and almost 20,000 words, occupying 128k on disc.

You can flick through the file quickly using the Open-Apple-1-9 command. The Find and Replace command took only a few minutes to replace a word that occurred 1,000 times in the text.

However I am not enthusiastic about large files for word processing. Having written hundreds of thousands of words with Applewriter – which I still prefer to Appleworks for word processing alone – I find it better to keep the files short for editing and then link them for printing.

But the extra memory of Ramworks is not just for creating huge files. It can also be used to hold several files in RAM and switch between them.

With the normal Appleworks Desktop of 55k you are limited to, say, two files of about 25k each or 12 smaller files of about 4k each. With the 256k Ramworks you could have two files of about 90k each or 12 files of about 15k each.

With the 1mbyte Ramworks you could have 12 files of about 60k each.

The speed of switching between the files depends on two factors – whether the files are all of the same type, such as spreadsheets, and whether the Desktop is nearly full.

If all the files are of the same type, the switch from one file to another, using the Open-Apple-Q command, is almost instantaneous.

However if the Desktop is nearly full the switch takes longer because it has to refer to the program disc in order to load in the necessary part of the program. If you want to switch rapidly, don't fill too much of the Desktop.

One version of the modified Appleworks disc preloads the three Appleworks programs in the Ramworks so that it can switch rapidly between them. But this version of Appleworks takes about 90 seconds to boot up compared with about 20 seconds for the normal version.

In practice there is not much difference in overall time between the two versions. Both recognise the extra memory.

One loads the three programs in as you boot up, the other loads them in as you need them. But once the programs are loaded, you can switch almost instantly between files of the three different types.

However if you fill too much of the Desktop with files, Appleworks refers to the program disc between changes. The version that preloads the programs in at the boot stage seems to drop the programs if you use too much of the Desktop.

The merits of Ramworks are not confined to Appleworks. There is also a pre-boot disc for Visicalc costing £29, which offers up to 437k of memory. There is a £29 Ram Drive disc for turning Ramworks into a solid state disc drive.

There is a CP/M version of this Ram Drive for £29, and an option for RGB output for £129.

Ramworks comes with a 20 page manual for programmers who want to use the auxiliary memory. If you buy the 256k version you can extend the memory later by adding more chips.

Ramworks will be a boon to many users of Appleworks, especially those who want to create large files or to switch rapidly between files of different types.

Doing the impossible with a Thunderclock card

When, a year or two ago, I bought a Thunderclock card for my Apple II+ I didn't know that it was eventually to become the official Apple clock card.

Although I always missed the current year being available, it kept excellent time and I was otherwise very pleased with it.

So when I eventually updated from DOS 3.3, I was delighted to find that ProDOS read the card, automatically making note of the date and time.

What really astonished me, however, was that ProDOS managed not only to read the current month and day – as well as the usual hours, minutes and seconds – from my clock card, but also appeared to know the year which was then 1984.

As in two years I had never managed to read this information from the clock card, I naturally thought that ProDOS had the year detail programmed in, and expected that it would remain as 1984.

On the offchance though, I took time out from New Year celebrations to check my Apple. Was ProDOS still assuming it was 1984? Well, no, it wasn't. To my amazement, at midnight the screen changed to January 1, 1985.

How did the Apple manage to get more information from my Thunderclock card than even the manual said was in there?

Why had I never managed to find it from Basic, in the fairly lengthy time I'd been using the card?

The answer, I finally found, was that the current year is not available from the Thunderclock. ProDOS uses an algorithm to calculate it, based on the day of the week the current year started. When you think about it, in a normal year the 365 days will divide into 52 weeks and one day, so that January 1 will be one day later each year.

For example, it falls on Monday in 1984, Tuesday in 1985, Wednesday in 1986 and Thursday in 1987.

If you can find out on which day of the week January 1 fell, it is therefore possible to work out which year from 1984 to 1987, is the current one.

1988 is a leap year with 366 days, so ProDOS's algorithm will fail on 29 February, 1988. In that year, of course, the Apple II will be 10 years old.

Once I knew it was possible to make my Thunderclock give me the year detail as well as the rest of its information, I had to write a program which did it.

The listing, which will run under either DOS 3.3 or ProDOS, shows an example of the way in which the year can be found. Rather than write a program which reads the card, I have allowed the current month, day of month and day of week details to be read from the keyboard.

This allows various different dates to be fed in, testing that the program actually works.

When you are satisfied that it does, substitute the new lines 300-350 for those in the original listing, making sure you change the slot number to the slot in which you have your Thunderclock. The routine is then ready to be included in your larger program.

You will now be able to automatically print the full date and time – just like ProDOS.

If you are still using your Apple in 1988 it would be a simple matter to reset the baseline of my program to read the year from the next leap year cycle, 1988-1991 – then you'll be right, and ProDOS wrong.

What a pity we'll have to wait until 1988 to see it.

Duncan Langford

```
100 REM Year-from month demo
110 REM by Duncan Langford
120 REM All date variables are
    as in the Thunderclock
    manual. !! Remember -- it
    only works until 1987 !!
130 :
140 REM Set variables
150 :
160 DIM M(12)
170 FOR I=1 TO 12
180 READ M:M(I)=M(I-1)+M
190 NEXT I
200 DATA
    31,28,31,30,31,30,31,31,30,3
    1,30,31
210 :
220 REM M (Month)=accumulated
    days so far, at month's end
230 :
240 FOR I=1 TO 3
250 Y(I)=1984+I
260 NEXT I
270 :
280 REM Y() holds the year --
    1985-1987
290 :
300 REM Get current date
310 :
320 TEXT:HOME
330 VTAB 10:INPUT"Month
    (1-12)?";MO
340 VTAB 12:INPUT"Day of month
    (1-31)?";DT
350 VTAB 14:INPUT"Day of week
    (0=Sun, 6=Sat)?";DW
360 DW=DW+7*NOT DW:REM Make
    Sunday 7 rather than 0
370 :
380 REM Calculation
390 :
400 Q=M(MO-1)+DT: REM Total
    days
410 X=Q-(INT(Q/7)*7): REM Less
    than a week?
420 IF X>DW THEN X=X-DW:GOTO
    470
430 X=DW-X: REM Count back
440 :
450 REM And the answer is...
460 :
470 VTAB 16:PRINT"The year is
    ";Y(X)
480 END

300 REM Replace lines 300-350
    with these to get data from
    the clock card
310 PRINT CHR$(4)"PR#2":REM
    Card in Slot #2
320 PRINT CHR$(4)"IN#2"
330
    INPUT"#";MO,DW,DT,HR,MN,SEC
340 PRINT CHR$(4)"PR#0":REM
    Restore screen
350 PRINT CHR$(4)"IN#0"
```

...and poking the date into DOS

A very useful procedure is to poke the current date within DOS.

As long as DOS is not rebooted the date will be available for any program prepared to use it.

Two unused bytes in both DOS 3.3 and Fast DOS are 47097 and 47098. Your HELLO program can include the lines shown on the right.

```
100 HOME: VTAB 10: HTAB 5:
    PRINT "Enter today's
    date: DD/MM";: HTAB 25:
    INPUT "";DA$
110 IF LEN(DA$) <> 5 AND DA$
    <> "" THEN 100
120 IF DA$ = "" THEN HOME:
    GOTO ... (rest of
    program)
130 POKE
    47097,VAL(LEFT$(DA$,2)):
    POKE
    47098,VAL(MID$(DA$,4,2))
```

Haris Courouclis

Figure I

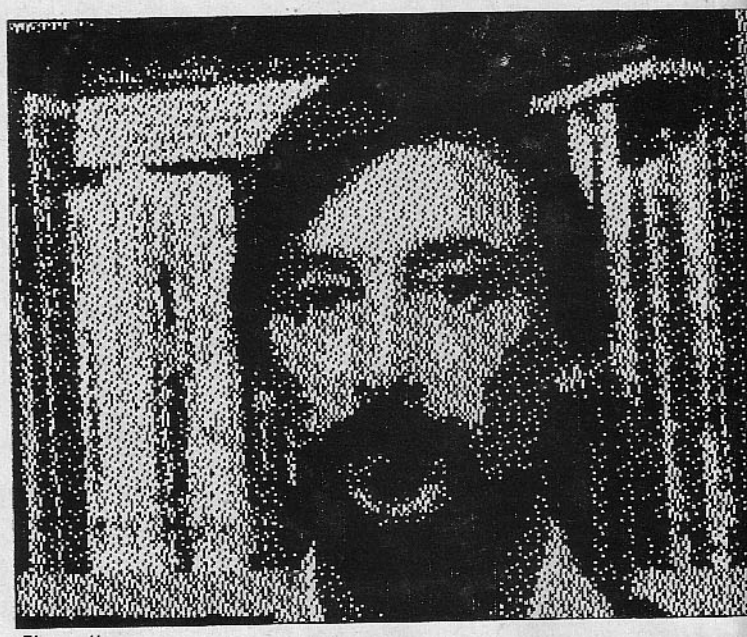


Figure II

Oh what a digitised picture...

CLIFF McKNIGHT reviews two digitising packages for the Apple II and Macintosh

ONE of the standard side-shows at any computer exhibition is the digitised picture stall. You have your photo taken, and it is then dumped to an Epson printer.

The technique for doing this is usually referred to as digitising. Recently I was given the chance to look at two such systems, one for the Apple II range and one for the Macintosh.

The system for the Apple II is called Computereyes, a terrible pun even by my standards. It describes itself as a video acquisition system and is manufactured by Digital Vision. It's marketed in this country by Stem Computing.

The system consists of a small (4in × 3in × 1.5in) black box bearing two knobs on one side and an RF input socket on another. One knob is labelled Sync and the other Brightness.

A ribbon cable runs from the box and terminates in a 16-pin connector which plugs into the game I/O socket.

All you need to do is plug the connector into the game I/O, plug a video source into the RF input and boot the supplied software.

Product: Computereyes.
Distributor: Stem Computing, 3 Blackness Avenue, Dundee DD2 1ER.
Price: £139.
System: Apple II family with 48k.

It's simplicity itself, especially if like me you have a break-out box from your game port. Even if you've never opened your Apple before, the manual provides ample details and photographs.

For the video source I used a cctv camera fitted with a zoom lens, both of which were kindly loaned by Elliot Kahan of Heyden Datasystems. Lighting was by bare 200 watt bulb.

Once the software is booted an 11-item menu is presented. This is one more item than the manual shows but the copyright date on the menu is 1985 while the manual bears a 1984 date. Hence, I'd guess that the software has been upgraded.

The first menu item is Help. Selecting this allows you to call up information regarding all the other menu functions.

The other utility menu items are: save to, load from or catalog disc; demo; and exit. All these are self explanatory.

The first thing to be done is select the adjust sync item and do just that. An arrow is shown on the screen and you turn the sync knob in that direction until the words In Sync appear. Once again, simplicity itself.

With sync adjusted you're ready to start capturing a video image. There are three types of capture available: a normal black-and-white image, a four-level and an eight-level image.

These latter two allow grey-scale pictures to be built up, one using four separate scans and the other eight.

For comparison I've taken the same shot using each method and without altering the brightness control – see Figure I,, II and III.

The black-and-white could be improved slightly by using the brightness control.

A normal scan takes just under five seconds. The four-level and eight-level captures use the corresponding number of slow-scans so you can appreciate the difficulty in getting a subject to sit still for about 35 seconds.

In addition to the video camera I also tried the system using an old, well-worn JVC video tape recorder as the source.

This presented two main problems. Firstly, the vtr pause facility doesn't present an absolutely still image so there's a certain amount of "noise" being input.

Secondly, with a scan taking about five seconds you can't run the vtr and select a shot to digitise.

What I did therefore was to pause the vtr at an appropriate point during a black-and-white movie and connect it to the digitising system. The result is

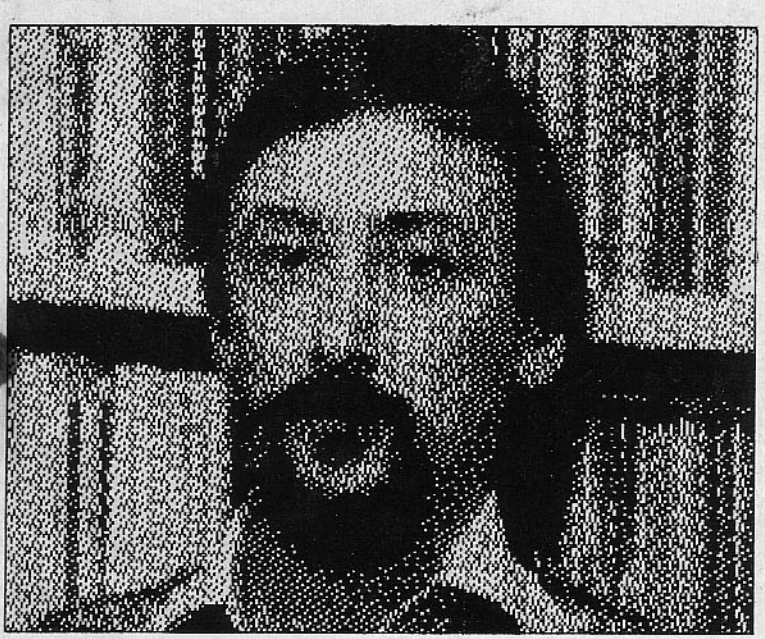


Figure III

Figure IV

shown in Figure IV – a prize for anyone who can guess the movie!

The pictures are saved to disc in either the normal or packed form. The normal form is the standard 8k dump of hi-res page 1 and uses 34 sectors. In this form the pictures can be accessed by many other packages and could therefore be subjected to considerable manipulation.

In packed form the pictures occupy less space on a disc and can be reloaded from the main menu.

The manual also provides information on using the system from within your own program, both as CALLs from Basic or JSRs from assembly language.

Overall the Computereyes system has to be one of the easiest pieces of hardware to connect, and using it is no harder than connecting it.

MAC MAGIC

THE Macintosh system is called Magic. It describes itself as a graphics input controller, is manufactured by New Image Technology and is marketed over here by Heyden Datasystems.

Setting up the hardware is simple. Apart from power leads, it's simply a case of video source to Magic box and Magic box to Macintosh.

I used the same camera, lens, light source and subject as with the Computereyes system, not so much for comparison as convenience.

The basic black-and-white picture is shown in Figure V. The scanning rate is much faster than the II-based system and is also continuous. Combined with the Mac resolution, this gives reasonable pictures.

The fun comes in trying to produce grey-scale pictures. These are produced using a pattern bar to decide the threshold levels for the various levels of grey.

It's called a pattern bar because you don't need to use just different density dots to produce the picture – you can define your own patterns (à la MacPaint) and substitute them on the pattern bar.

The thresholds can also be dragged about with the mouse so the possibilities are almost endless.

Combine this with the facilities to expand or reduce a picture, save and load pictures in MacPaint file format and you have a very powerful system indeed.

The problem with endless possibilities is that it's easy to get lost in them or to spend hours looking for the exact combination you want.

In order to produce the picture shown in Figure VI I reverted to almost default settings, but I had a great time trying the other options.

I also attached the JVC video

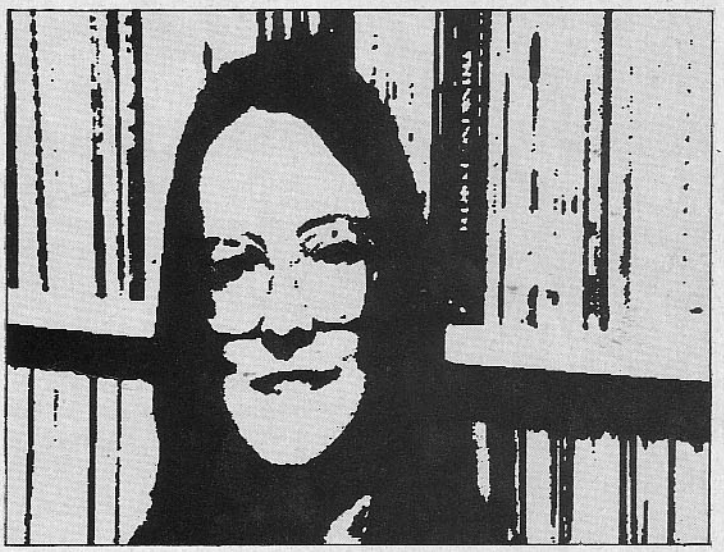


Figure V

Figure VI

tape recorder to the system. Because the scan rate is quite fast (2.5 frames a second) it was possible to watch the movie as a rapid series of slices and pick a shot. The result is shown as Figure VII. Have you guessed the movie yet?

In principle, you could capture a picture from live television but I didn't try. To be honest, there's not much on TV I'd want to capture.

The manual provided with the review system was only a draft and is apparently undergoing considerable revision. To be fair, just about everything was in there if you looked. The difficulty I found was in translating the words on the page into the desired effect on the screen.

Product: Magic.
Distributor: Heyden Datasystems, Spectrum House, Hillview Gardens, London NW4 2JQ.
Price: £695.
System: 128k Macintosh.

Overall I found the Magic system was easy to use but easy to get lost in. The cost of power is complexity, but there's an enormous range of possibilities.

CONCLUSION

Both the systems were used under less than optimal lighting conditions. By stretching the camera lead into our kitchen, which is lit by flourescent light, a better quality image was obtained.

The problem was that I couldn't be bothered doing the washing-up and I'd hate you to see the place as it was! The only other fluorescent-lit room was the bathroom . . .

I tried taking pictures of the kids but encountered a different set of difficulties. They wouldn't sit still long enough for he Computereyes system to scan them, and lighting seemed to be much more critical. Their little round faces don't provide as many angles (and therefore information points) as a craggy old editor's face.

Figure VII

One application I would have liked to try is to print a T-shirt from a captured image. There are ribbons available for the Imagewriter that will produce an iron-on transfer of anything dumped to the printer.

Also, Denise is now involved in converting a four-level Computereyes picture into a tapestry. She says the form of the picture lends itself to conversion this way.

In fact we can think of hundreds of applications for digitisers around the home, neglecting all the "serious" applications.

I mean, personalised notepaper could be really personalised and Readers' Wives need never be quite the same again.

appletip

Flip your screen over

There are some packages around which give loads of neat graphics routines, but cost a bomb. Here's one routine for free.

It mirror-images a block of hi-res, which can be the whole screen or just a small box. The routine relies heavily on three relatively large tables.

To save you time, and to make sure that there are no mistakes in the tables, I have written routines for making them.

To get everything into the computer, you should type the following:

```
]NEW
](now type in Listing I)
]RUN
]CALL -151
*4000:(now type in the hex
   dump in Listing II)
*40006
*BSAVE FLIP,A$4000,L$3FF
```

To use the routine just type in **CALL 16433**. In its present form, the routine will mirror the whole screen. The following POKEs may be used to change the size of the window:

```
]POKE 16434,(top of window)
]POKE 16462,(right of window
   / 7)
]POKE 16468,(left of window
   /7, -1)
]POKE 16488,(bottom of
   window)
```

The left and right coordinates must be divisable by seven, due to the Apple's screen format.

If the difference between the left of the window and the right is an even number, there will be colour changes in the image. This is unavoidable when mirror imaging left-right.

To use the routine in the future, just type in **BLOAD FLIP** and then use **CALL 16433** and the POKEs as above.

Julian Brewer

```
10 DATA 0,128,40,168,80,208
20 C = 17152
30 FOR I = 0 TO 2
40 READ A,B
50 FOR R = 0 TO 3
60 FOR T = 0 TO 7: P = C +
   (R * 16) + (I * 64) + T:
   POKE P,A: POKE P + 8,B:
   NEXT T
70 NEXT R,I
80 C = 16896
90 FOR J = 0 TO 2
100 FOR I = 32 TO 35
110 FOR F = 0 TO 7: D = I +
   (4 * F): POKE C + F,D:
   POKE C + F + 8,D: NEXT F:
   C = C + 16
120 NEXT I,J
```

Listing I

```
4000: A0 00 A9 00 85 01 98 29 4040: 40 A6 02 BD 00 43 85 00
4008: 80 85 00 98 A2 06 4A 85 4048: BD 00 42 85 01 A0 27 B1
4010: 03 90 09 BD 2A 40 05 01 4050: 00 48 88 C0 FF D0 F8 A0
4018: 85 01 A5 03 CA 10 EF A5 4058: 27 68 AA BD 00 41 91 00
4020: 01 05 00 99 00 41 C8 D0 4060: 88 C0 FF D0 F4 A5 02 C9
4028: D9 60 01 02 04 08 10 20 4068: BF F0 05 E6 02 4C 41 40
4030: 40 A2 00 86 02 AD 4E 40 4070: 60
4038: 8D 58 40 AD 54 40 8D 62
```

Listing II

By PETER GORRY

Bring your histograms right up-to-date

Part XVI of the Apple User Graphics library

WAY back in the March, 1984 issue of *Apple User,* in the second instalment of the Graphics Library, I presented a series of histogram routines capable of plotting single or multiple sets of data, with or without spaces horizontally or vertically.

The graphics library has come a long way since then and it is now time to bring the routines fully up to date.

The annotation routines in the July and August issues of *Apple User* are designed to handle all the histogram variations, but it requires a new histogram controller routine and a few modifications to the box and histogram plotter routines to achieve the desired results.

The new controller routine allows us to adopt the same data structure as the rest of the library although it will need a little explanation of how "grouped data" is handled. The modifications are given in Listing I, the new controller and an example of its use is given in Listing 2.

The routines make extensive use of the *Apple User* Graphics Library and will only function if most of the library is present.

The range finding and axis drawing/annotation is performed using the existing routines and you should refer to the relevant *Apple User* issues for full details of all the options available. The histogram option is activated by setting ZG=1.

The histogram routines were specifically designed to handle symbolic or nominal data in which one has a list of labels with associated values.

In the simplest case we have a label for each value. For instance, it could be SALES FIGURES by MONTH. This is the sort of symbolic data we can already plot and produce pie charts for. The values are stored in the ZY() array and the labels in the ZP$() array.

The Histogram would contain 12 groups (months) each having one bar (the sales figure). In this simple case we can produce a bar chart just by setting ZH(1)=12 and ZH(2)=1 and calling the histogram plotter instead of the point plotter.

In the case of grouped data we must be careful how the data is set up. The necessity for having grouped data arises if we want to lump several items into a common catagory.

For instance, if you want to show the profits from each of three companies over the last four years we would want four groups of bars, each group containing three bars, as shown in Figure I. We only want to label the GROUPS this time, not each individual bar.

The convention adopted by the histogram controller in this case is that the group labels are stored in ZP$() and the values of the bars in ZY() as before. The only difference is that there will be more entries in ZY() than ZP$() this time. The values are stored in the order they appear across the screen.

In the example above ZY() would hold the three profit figures for 1981 first, then the three figures for 1982 . . . and so on. This can be seen explicitly by examining lines 605 and 610 in the example program.

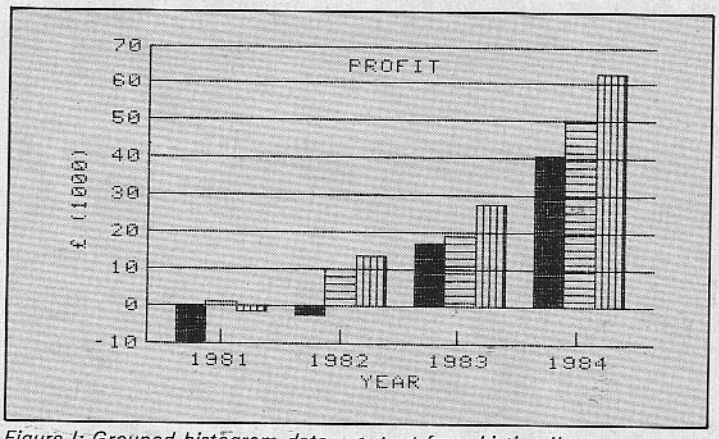


Figure I: Grouped histogram data – output from Listing II

The option for having horizontal bars is also supported and set using the same parameter as before – ZH(8)=1.

This can be demonstrated simply by inserting a line, before line 150, setting this value.

This now completes the integrated plotting functions of the library routines and brings them to using a common format.

In all the instalments so far I have geared the routines to ones which would be installed in the body of another program in order to provide flexible graphical output.

It has been up to the program to provide the necessary control and display options before calling the routines.

An alternative to this is to ask the user at run time to decide which of the various options are required.

This leads to the "asking" routines I promised so long ago. The first of these are also provided in this month's listing.

Up till now the data being displayed in the programs has always been held internally in the programs or generated by a formula or random number generator.

The routine at line 56000 provides a simple data input routine that supports numeric, symbolic and histogram formats in a form ready for the plotting routines. The routine at 56500 gets the graph labels.

As long as we are willing to accept default values for everything else on the graph this is all the information we require to plot the data.

Lines 180-230 thus represent a very simple graph

plotting program. It is not meant to be a serious tool but it illustrates how easily the library can be used to plot data.

Line 170 waits for you to press the space bar after performing the histogram plot shown in Figure I.

Line 180 then presents you with the data input menu and you can choose numeric, symbolic or histogram data as desired. The input is fully "prompted" so you should have no trouble using it.

The routine sets a "data type flag" – ZD$ – to show which type of data has been input. The symbol set is N for X and Y values both numbers, S if the X values are labels, and H if the data follows the histogram format.

Lines 210-230 then plot the data accordingly.

● *In another issue I will provide a routine which replaces 210-230 with something which provides a lot of choice about how the graph should look.*

```
40845  IF ZH < 0 THEN ZY = ZY + Z
       H:ZH =  ABS (ZH)
40846  IF ZW < 0 THEN ZX = ZX + Z
       W:ZW =  ABS (ZW):
41150 ZX = (ZH(4) - 1) * ZW + ZW(
       1)
41164  IF ZH(0) = 0 THEN 41170
41165 ZX = ZH(9) - ZW / 2
41220 ZY = (ZH(4) - 1) * ZH + ZW(
       3)
41234  IF ZH(0) = 0 THEN  GOTO 41
       240
41235 ZY = ZH(6) + ZH(5):ZY = ZY -
       ZH / 2:ZW = ZH(9) - ZH(5)
```

Listing I: Modifications/additions to BOX and HISTOGRAM PLOTTER routines

appletips

When programming in machine code or manipulating binary files it is nice to have a completely blank memory area.

Here is a simple way of clearing the area from $800 to DOS, using Applesoft commands only.

Type **FP** then **DIM A (7267)** and the main user memory now contains all zeros, apart from a few bytes at 803-809.

Derek Turner

The following statement, if inserted into an Applesoft program, will show how much memory remains in terms of elements in a single array of real numbers:

```
10 PRINT INT ((( FRE (0) +
( FRE (0) < 1) * 65536) / 5) – 3)
```

Derek Turner

To see filenames previously deleted under DOS 3.3 **POKE 44505,234: POKE 44506,234.** Deleted files, unless saved over, will be marked with an inverse letter when catalogued.

Jim Davies

```
80  DIM ZX(50),ZY(50),ZP$(50),ZD(
    50)
100  REM

EXAMPLE PROGRAM

110  GOSUB 42400: REM  SHAPE TABL
    E LOADER
120 ZN = 12: GOSUB 600: REM  SET
    UP DATA
130 ZG$(1) = "PROFIT":ZG$(2) = "Y
    EAR":ZG$(3) = "£ (1000)": REM
     SET GRAPH LABELS
140 ZF = 1: GOSUB 49400: REM  DEF
    AULT GRAPH/PAGE
145 ZH(1) = 4:ZH(2) = 3: REM   4
    YEARS, 3 ITEMS PER YEAR
146 ZG(0) = 2:ZG(4) = 2:ZG(5) = 1
    : REM  AUTOMATIC RANGES WITH
     ZERO AND HORIZONTAL LINES
150 ZG = 1: GOSUB 450: REM   DRAW
     HISTOGRAM
170  GET A$: IF A$ <  > " " THEN
     GOTO 170
180  GOSUB 56000: GOSUB 56500

190 ZG(0) = 2: REM  AUTO RANGES
200 ZG = 0: IF ZD$ = "H" THEN ZG =
    1
210  IF ZD$ = "N" THEN  GOSUB 550

220  IF ZD$ = "S" THEN  GOSUB 500

230  IF ZD$ = "H" THEN  GOSUB 450

280  END :

450  REM  CLEAR PAGE, SET RANGE,
    DRAW AND LABEL - HISTOGRAM D
    ATA
460  GOSUB 49400: GOSUB 49500: GOSUB
     48600: GOSUB 41400
470  RETURN
500  REM  CLEAR PAGE,SET RANGE,DR
    AW AND LABEL - SYMBOLIC DATA

510  GOSUB 49400: GOSUB 49500: GOSUB
     48600:ZT = 2: GOSUB 42600
520  RETURN :

550  REM  CLEAR PAGE, SET RANGE,
    DRAW AND LABEL - NUMERIC DAT
    A
560  GOSUB 49400: GOSUB 47600: GOSUB
     48600:ZT = 1: GOSUB 42600
570  RETURN
600  REM  DATA ROUTINE
605 ZP$(1) = "1981":ZP$(2) = "198
    2":ZP$(3) = "1983":ZP$(4) =
    "1984"
610  FOR I = 1 TO ZN:ZY(I) = I *
     I / 2 -  RND (1) * 15: NEXT
620  RETURN :

41400  REM

HISTOGRAM CONTROLLER

41410  REM  CONTROLS PLOTTING OF
     COMPLETE HISTOGRAM
41420  REM  USES INFO IN ZH ARRAY

41430  FOR ZK = 1 TO ZH(1): REM
     GROUPS
41440  FOR ZJ = 1 TO ZH(2): REM
     BARS IN GROUP
41450 ZA = ZJ
41460  IF ZA > 3 THEN ZA = ZA - 4
     : GOTO 41460: REM  SELECT SH
    ADING
41470 ZH(7) = ZA: REM  SHADING
41475  IF ZH(8) = 1 THEN ZH(6) =
     ZX((ZK - 1) * ZH(2) + ZJ) -
     ZH(5): GOTO 41484
41480 ZH(6) = ZY((ZK - 1) * ZH(2)
     + ZJ) - ZH(5)
41484  IF ZH(0) = 0 THEN 41490
41485 ZH(10) = ZY((ZK - 1) * ZH(2
    ) + ZJ)
41486 ZH(9) = ZX((ZK - 1) * ZH(2)
     + ZJ)
41490 ZH(3) = ZK:ZH(4) = ZJ: REM
     GROUP AND BAR NO
41500  GOSUB 41000: REM  PLOT HIS
    TOGRAM
41510  NEXT ZJ: NEXT ZK
41520  RETURN :

56000  REM

GET DATA

56005  TEXT : HOME : INVERSE : PRINT
     "DATA INPUT": NORMAL : PRINT
     : PRINT
56010  PRINT "1  NUMERIC ONLY": PRINT
     "2  SYMBOLS/NUMBERS": PRINT
     "3  HISTOGRAM DATA"
56020  PRINT : INPUT "WHICH ?";ZB

56030  IF ZB > 1 THEN  GOTO 56100

56040  PRINT : INPUT "NUMBER OF D
    ATA POINTS ? ";ZN
56050  PRINT : PRINT "NOW INPUT D
    ATA IN PAIRS: X,Y ": PRINT
56060 ZU(1) = 0:ZV(1) = 0: REM  N
    UMBERS
56070  FOR ZI = 1 TO ZN: PRINT ZI
    ;"  ";: INPUT ZX(ZI),ZY(ZI):
     NEXT
56080 ZD$ = "N": REM  DATA TYPE F
    LAG
56090  GOTO 56300
56100  IF ZB = 3 THEN  GOTO 56200
56110  REM  SYMBOLIC DATA
56120  PRINT : PRINT : INPUT "NUM
    BER OF DATA POINTS = ";ZN
56130  PRINT : PRINT "NOW INPUT L
    ABEL,VALUE PAIRS"
56140  FOR ZI = 1 TO ZN: PRINT ZI
    "  ";: INPUT ZP$(ZI),ZY(ZI):
     NEXT
56150 ZD$ = "S": REM DATA TYPE FL
    AG
56160 ZU(1) = 1:ZV(1) = 0: REM  S
    YMBOLS/NUMBERS
56170  GOTO 56300
56200  REM  HISTOGRAM DATA
56210  PRINT : INPUT "NUMBER OF G
    ROUPS ? ";ZH(1)
56220  PRINT : INPUT "NUMBER OF B
    ARS IN GROUP ?";ZH(2)
56230  FOR ZK = 1 TO ZH(1)
56232  IF ZH(1) = 0 THEN  GOTO 56
    260
56240  PRINT : PRINT "LABEL FOR G
    ROUP ";ZK;: INPUT ZP$(ZK): PRIN

56250  PRINT "NOW INPUT BAR HEIGH
    TS FOR THIS GROUP": PRINT
56260  FOR ZJ = 1 TO ZH(2)
56262  IF ZH(1) > 0 THEN  GOTO 56
    270
56263  INPUT "LABEL,VALUE ?";ZP$(
    ZJ),ZY(ZJ)
56264  GOTO 56280
56270  PRINT ZJ;"  ";: INPUT ZY((
    ZK - 1) * ZH(2) + ZJ): NEXT

56280  PRINT : PRINT : NEXT
56290 ZD$ = "H": REM  DATA TYPE F
    LAG
56295 ZN = ZH(1) * ZH(2): IF ZH(1
    ) = 0 THEN ZN = ZH(2)
56300  RETURN :

56500  REM

TITLE INPUT

56510  TEXT : HOME : INPUT "GRAPH
     TITLE ? ";ZG$(1)
56520  PRINT : INPUT "X AXIS TITL
    E ? ";ZG$(2)
56530  PRINT : INPUT "Y AXIS TITL
    E ? ";ZG$(3)
56540  RETURN
```

Listing II

As a matter of factorials...

S. EVESON provides a USR routine to help with statistical problems

IT is often useful, when working on statistical problems, to have a factorial function in Basic. This 50-byte program for values up to 33! uses the USR function – a little-used, but often useful, feature of Applesoft.

When a Basic program encounters a USR control is passed to $000A, with the value of the argument in the floating-point accumulator, FAC, at $9D.

The first part of this program, SETUP, at $0300, places a JMP instruction at $000A, to FACT, at $030D, and returns to Basic, ready for calls to the routine.

For the multiplication routine used later to work, XORFPSGN – the eXclusive OR of the floating-point SiGNs – must be set to zero if the numbers are of the same sign or one if they are different. As all the terms to be multiplied are positive, it is set to zero.

The program then calls GETADR at $E752 which converts the argument (in FAC) to a 16-bit integer (of which only the lower eight bits are used) in LINNUM, at $50.

GETADR was chosen because negative numbers in FAC produce their complements in LINNUM, which will be sufficiently large to cause an OVERFLOW ERROR message from Applesoft, consistent with the factorials of negative numbers being infinite.

The A and Y registers are then loaded with the address of the floating-point constant, 1.

A call to CONUPK, at $E9E3, loads this value into ARG, the secondary accumulator, at $A5.

A check is then made for the argument being zero. If it is, as 0!=1, and one has already been loaded, the calculation loop is bypassed, and the answer 1 is returned.

The calculation loop uses LINNUM as its loop counter. The Y register is loaded with LINNUM, and this value is floated into FAC by SNGFLT at $E301.

The A register is then loaded with the contents of FACEXP, the exponent byte of FAC, at $9D, for the call to FPMULT at $E982, which multiplies FAC and ARG, placing the product in FAC. FACARG then moves this product into ARG, ready for the next multiplication.

LINNUM is then decremented, and, if the result is non-zero, the program loops back to LOOP for the next multiplication.

When the loop has been completed the program returns via ARGFAC at $EB53, which moves ARG to FAC, ensuring that the result is in FAC. This is only necessary when the argument is zero, in which case the answer 1 is in ARG and the argument zero still in FAC.

Note that fractional arguments are rounded down by GETADR and for arguments greater than 255 the factorial of the argument, modulo 256 is returned.

```
0300- A9 4C 85 0A A9 0D 85 0B
0308- A9 03 85 0C 60 A9 00 85
0310- AB 20 52 E7 A9 13 A0 E9
031B- 20 E3 E9 A5 50 F0 11 A4
0320- 50 20 01 E3 A5 9D 20 82
0328- E9 20 63 EB C6 50 D0 EF
0330- 4C 53 EB
```

Hexadecimal dump

```
0800                    1  ***************
0800                    2  *             *
0800                    3  * FACTORIALS *
0800                    4  *             *
0800                    5  * S.EVESON   *
0800                    6  *             *
0800                    7  * 4-4-1985   *
0800                    8  *             *
0800                    9  ***************
0800                   10  *
0800                   11  *
000A                   12  USRADR   EPZ $0A
004C                   13  JMP      EPZ $4C
0050                   14  LINNUM   EPZ $50
009D                   15  FACEXP   EPZ $9D
00AB                   16  XORFPSGN EPZ $AB
E301                   17  SNGFLT   EQU $E301
E752                   18  GETADR   EQU $E752
E913                   19  ONE      EQU $E913
E982                   20  FPMULT   EQU $E982
E9E3                   21  CONUPK   EQU $E9E3
EB53                   22  ARGFAC   EQU $EB53
EB63                   23  FACARG   EQU $EB63
0800                   24  *
0800                   25  *
0300                   26           ORG $300
0300                   27           OBJ $800
0300                   28  *
0300                   29  *
0300 A9 4C             30  SETUP    LDA #JMP          ;SET UP USER HOOK
0302 85 0A             31           STA USRADR
0304 A9 0D             32           LDA #FACT
0306 85 0B             33           STA USRADR+1
0308 A9 03             34           LDA /FACT
030A 85 0C             35           STA USRADR+2
030C 60                36           RTS
030D                   37  *
030D                   38  *
030D A9 00             39  FACT     LDA #$00          ;ALL TERMS
030F 85 AB             40           STA XORFPSGN      ;POSITIVE
0311 20 52 E7          41           JSR GETADR        ;GET ARGUMENT
0314 A9 13             42           LDA #ONE          ;SET ARG
0316 A0 E9             43           LDY /ONE          ;TO 1
0318 20 E3 E9          44           JSR CONUPK
031B A5 50             45           LDA LINNUM        ;0! = 1
031D F0 11             46           BEQ END           ;SO DROP THROUGH
031F A4 50             47  LOOP     LDY LINNUM        ;MOVE NEXT TERM
0321 20 01 E3          48           JSR SNGFLT        ;INTO FAC
0324 A5 9D             49           LDA FACEXP        ;REQUIRED FOR FPMULT
0326 20 82 E9          50           JSR FPMULT        ;MULTIPLY FAC AND ARG
0329 20 63 EB          51           JSR FACARG        ;MOVE FAC -> ARG
032C C6 50             52           DEC LINNUM        ;NEXT TERM TO MULTIPLY
032E D0 EF             53           BNE LOOP          ;CONTINUE IF NOT FINISHED
0330 4C 53 EB          54  END      JMP ARGFAC        ;MOVE ANSWER TO FAC
0333                   55           END
```

The mechanics of the 6502 assembly line

STUART BELL explains how to write, assemble and link assembly language programs under Apple Pascal

WRITING 6502 assembly language programs under the Apple Pascal system has much in common with writing Pascal programs. You use the same editor, the same filer and, when a syntax error is encountered, control can be passed to the editor, just as with Pascal programs.

Those who have used other assemblers on the Apple will find that the Apple Pascal assembler has a number of special characteristics of which you should be aware before using it.

The most important of these is that you never know – or need to know – where in memory the machine code program will be located.

Fundamental to the concept of the UCSD P-System is that the actual hardware locations are invisible to the programmer. Programs, whether produced from Pascal or assembly language, are relocatable – that is, they can be run anywhere in memory.

In the case of assembly language programs, a special loader fixes all the references to particular memory locations when the program is run. This can be disconcerting if you are used to knowing where in memory your program is, but such knowledge is usually unnecessary.

The second difference – one that has caught me out a few times – is a difference in the syntax of instructions using indirect addressing. Whereas most assemblers use the forms:

```
LDA (FRED),Y
LDA (EPRT,X)
JMP (EXITPTR)
```

the Apple Pascal assembler uses slightly different forms:

```
LDA @FRED,Y
LDA @ERPT,X
JMP @EXITPTR
```

Also the Pascal assembler expects 'numbers' (whether literals or addresses) to be in hexadecimal but needs a zero in front of them if the first character is not in the range 0 to 9. Decimals are indicated by a trailing decimal point.

So much for the problems. The Apple Pascal assembler is a very powerful software development tool, giving macros, conditional assembly and a host of other directives.

Experienced assembly language programmers should refer to Page 157ff of the Operating System manual.

The rest of this tutorial will be devoted to illustrating to less experienced programmers the stages in writing, assembling and linking 6502 assembly language programs.

You will need a disc in drive 2 (#5:) with the following files on it:

```
SYSTEM.ASSMBLER
SYSTEM.COMPILER
SYSTEM.LINKER
6500.OPCODES
6500.ERRORS
```

Obviously if any of these are on drive 1 (#4:) then you need not have them in drive 2. It's worth keeping a special disc for developing assembly language programs as this is the only time that most of these files are needed.

Assuming you have the editor and filer on your disc in drive 1 (#4:), the stages of development are as follows:

1. Use the editor and compiler to produce the Pascal program which will call the assembly language program. An example program is given in Listing II. When it has compiled without errors, save the program. In the case of the example program, save it as CALLKPRESS.

2. Clear the workfile from within the filer.

3. Use the editor to type in your assembly language program. A specimen program is given in Listing I. We shall examine it more closely later.

4. Quit the editor, updating the workfile. Now type A to assemble it, just as you would type C to compile a Pascal program. Unless you want a listing of the assembly, key Return when prompted for an output file.

But be warned: Output to the printer assumes that it is 132 columns wide. If an error is found, re-edit the file, and repeat the process until all errors have been removed.

5. The final assembly will have produced SYSTEM.WRK.CODE, the 6502 machine code program. Now use the Filer to save the workfiles.

To avoid confusion, I always name my assembly language files with .MC in them, such as: KPRESS.MC.

6. Machine code files cannot be run directly – they must be linked into a Pascal program which will call them. Fortunately the system gives us a very elegant way of passing data between Pascal and assembly language programs, and so lets us use the two languages for tasks at which they are best.

We can use Pascal for input, output and complex logic, and assembly language for speed-critical stuff, and driving hardware directly.

7. We now invoke the Linker to join the two programs together. Type L at command level.

8. To the prompt HOST FILE? reply with the name of the Pascal program file, such as: CALLKPRESS.

9. To the prompt LIB FILE? reply with the name of the assembly language program file, such as: KPRESS.MC. At the second prompt LIB FILE? just hit Return.

10. To the MAP FILE? prompt, hit Return.

11. To the prompt OUTPUT FILE? give the name of the file to hold the combined program, such as: KEYPRESS. The output file now contains a mixture of p-code instructions, produced by compiling the Pascal program, and 6502 machine code instructions produced by the assembler.

Control and data will pass from one to the other without the user being aware of the change.

12. Now from the Command level type X and then the name of the output file, such as KEYPRESS, to execute the final program.

Unless you are an experienced assembly language programmer, I suggest you try the above process on the example programs before proceeding further.

Let us now consider the mechanics of the calling process in a little more detail.

In the Pascal program – Listing II – you will notice the following two lines:

```
function keypress:boolean;
external;
```

This means that the function called 'keypress' returns a Boolean value – a 'true' or a 'false' – and that it is external to, or outside of, the main program. This tells the compiler that the function will be linked in later.

The assembly language listing is rather more complex – Listing I – and some of the lines are peculiar to the Apple Pascal System.

The line:

```
.FUNC KEYPRESS,0
```

is particularly significant. It indicates that the code is a function as opposed to a procedure, that it is called KEYPRESS – this must match the name in the Pascal program – and that there are no parameters passed from the Pascal program to this function.

```
        ;This function duplicates the KEYPRESS function supplied
        ;as part of the Applestuff Unit in the system library.
        ;Using this function, rather than Applestuff, reduces memory
        ;requirements - as unwanted unwanted routines in Applestuff
        ;are not loaded, and avoids the need for the library.

        ;Warning; it calls specific locations in the Apple Pascal
        ;system, and so will only work with Apple Pascal 1.1, the
        ;version that was distributed from 1980 to 1984 and Pascal 1.2
        ;which is now distributed.

        .FUNC KEYPRESS,0

RETURN  .EQU  0         ;the return address will be stored at loacation 0
CKCONSL .EQU  0BF0A     ;the location in the BIOS of the console check routine
KEYBD   .EQU  0C000     ;the hardware location of the keyboard strobe
READPTR .EQU  0BF18     ;the 'read pointer' of the type-ahead buffer
RITEPTR .EQU  0BF19     ;the 'write pointer' of the type-ahead buffer

                        ;code proper starts here
        PLA             ;get return address off stack & save it
        STA   RETURN
        PLA
        STA   RETURN+1
                        ;now pull 'spare bytes' off stack
        PLA
        PLA
        PLA
        PLA
        LDA   #0        ;irrespective of result (true/false), we must return
        PHA             ;the top byte of the result as zero, so do it now

        LDA   KEYBD     ;check the strobe port of the keyboard: if top bit
        BMI   YES       ;is set, then jump to label YES

        JSR   CKCONSL   ;call the console check routine, to see if key was
                        ;pressed earlier, but strobe cleared by the system
                        ;checking the keyboard
        LDA   RITEPTR   ;get the 'write-pointer' to the type-ahead buffer
        CMP   READPTR   ;if the different, then character in buffer
        BNE   YES

        LDA   #0        ;no key pressed, return false (0)
        PHA             ;push false onto stack
        BEQ   EXIT      ;branch always taken, as accumulator always 0

YES     LDA   #1        ;key has been pressed, return true (1)
        PHA             ;push true onto stack

EXIT    LDA   RETURN+1  ;restore the return address previously saved
        PHA
        LDA   RETURN
        PHA
        RTS             ;return to the Pascal calling program
        .END            ;end of the assembly
```

Listing I

```
program testkeypress;
(* silly little program to test keypress routine *)

var ch:char;
    i:integer;

function keypress:boolean;
external;

begin
  repeat
    repeat
      write('.');
      for i:=1 to 100 do; (* nothing- just slows
                             the loop down a bit *)
    until keypress;
    read(keyboard,ch);
    writeln
  until ord(ch) = 27 (* ie wait for ESC *)
end.
```

Listing II

We shall look at parameters in more detail next month.

Next we note the 'pulling' of a return address off the stack. This is the location in the Pascal program which will be jumped to when the routine has been completed. The four PLAs are needed with functions to make space on the stack for the result returned by the function.

At the end of the routine we note the pushing on to the stack of the return address, as RTS looks at the stack to see where it should go, and then the essential .END directive to tell the Assembler to stop assembling.

The Apple Pascal system provides a very powerful environment for the development of assembly language programs. It can be used to assemble very large and complex programs and, despite my introductory comments, can be used, at your own risk, to produce routines to run at a specific location.

Do not be afraid to experiment. It's certainly much easier and more secure than trying to link Basic programs to assembly language routines.

Try some of the example programs in the Operating System manual, and then find the effects of modifications to them.

Assembly Language programming opens up a whole new world to the Apple owner, and once you've got the Apple Pascal system, it's all free.

● *Next month, we shall look in more detail at the passing of data between Pascal and assembly language programs and then see how assembly language routines can be added into the system library.*

appletip

Locating monitor's LIST command

I have often thought it would be useful if I could locate and use the Apple monitor's LIST command. This LIST routine disassembles 20 lines of memory each time it is called. It starts disassembling at the memory location whose address is held in 58/$3A (lo-byte), 59/$3B (hi-byte). The actual list routine is located at 65118/$FE5E in memory. The following Basic program shows how to use it.

Jason W. Smith

```
5 TEXT: HOME
10 LO=0: HI=224
15 POKE 58,LO: REM LO-BYTE
   OF ADDRESS
20 POKE 59,HI: REM HI-BYTE
   OF ADDRESS
30 HOME: CALL 65118: REM
   DISASSEMBLE 20 LINES
35 HTAB 1: VTAB 23: PRINT
   "PRESS 'Q' TO END DEMO
   :";
45 GET A$: IF A$ = "Q" THEN
   END
50 GOTO 30
```

Upper and lower case in an instant

DAVID WELLS describes how to make lower case a transparent part of the Pascal system

```
387             PHA             ;Push video char on to stack
388             CMP #00         ;00 = Cursor
390             BEQ LOOP1       ;Branch to output if cursor
392             BIT E0          ;Check Ctrl-E flag
394             BPL LOOP1       ;Branch to output if Uppercase only
396             LDA F6          ;Get ASCII Code of output character
398             AND #7F         ;Clear bit 7
400             CMP #41         ;41 = "A" in ASCII
402             BCC LOOP1       ;Branch to output if < "A"
404             CMP #5B         ;5B = "[" in ASCII
406             BCS LOOP1       ;Branch to output if >= "["
408             PLA             ;Get video value from stack
409             EOR #80         ;Invert bit 7
411             PHA             ;Push inverted video value
412   LOOP1     LDA F4          ;Output character position
414             SEC             ;
415             SBC BF11        ;
418             BMI LOOP2       ;
420             CMP #28         ;
422             BCS LOOP3       ;If >= 40 (Decimal) then branch
424             TAY             ;Transfer output position to Y
425             PLA             ;Pull video value from stack
426             BNE LOOP4       ;Branch if not cursor
428             LDA (F0),Y      ;Get current screen character
430             EOR #80         ;Invert bit 7
432   LOOP4     STA (F0),Y      ;Store inverted character on screen
434             RTS             ;Exit
435   LOOP2     CLC             ;
436             ADC #28         ;Add 40 (Decimal)
438             JMP DBBC        ;Jump out of routine
441   LOOP3     SEC             ;
442             SBC #28         ;Subtract 40 (Decimal)
444             TAY             ;Transfer output position to Y
445             PLA             ;Pull video value from stack
446             BNE LOOP5       ;Branch if not cursor
448             LDA (F2),Y      ;Get current screen character
450             EOR #80         ;Invert bit 7
452   LOOP5     STA (F2),Y      ;Store character on screen
454             RTS             ;Exit
```

Listing I: Original Apple Pascal output routine

THIS article is a development of the technique described by J.P. Lewis in an issue of *Apple User's* predecessor, *Windfall.*

The modification described allows the lower case facility to become a transparent part of the Pascal system. In order to integrate this feature into the system it is necessary to modify the screen output routine in System.Apple.

The relevant portion of System.Apple (for Pascal 1.1) was disassembled by hand and its operation unravelled before attempting any changes. Listing I shows the assembler form of the output routine along with comments.

Please note that addresses are the byte number in block 5 of System.Apple with the first byte being 0.

Points to notice in Listing I are:

- On entry the accumulator always contains either 00, signifying a cursor output, or the code for the character to be output which may be a control character with bit 7 set for upper case.
- E0 is a flag indicating if Ctrl-E or Ctrl-W have been used – that is, whether the display should show upper and lower case.
- 409 – this portion is only executed if the character to be output is upper case and the screen is displaying both. Upper case characters are converted to inverse video, 81 = "A" in normal video EOR #80 gives 01 which is A inverse.
- 412-423 decide which text screen to write to.
- 424-434 and 444-454 are the output sections and only differ in the zero page location they use to indirectly address the screen.
- If the cursor is being output then the current character is fetched from the screen and bit 7 inverted. This changes normal video to inverse and inverse to normal.

Once the above operation is understood, modifying the routine to use the lower case chip appears relatively simple.

Replace the check for upper case characters with one for lower case and change the command at 409 to EOR #E0 to produce the code for lower case. This does display lower case but only if Ctrl-E has been pressed.

A more serious problem is that placing the cursor over a lower case character gives a strange display. Consider the character "a" with code $E1. Placing the cursor over it results in bit 7 being inverted to give $61 which is the video code for "!" in flashing mode.

The problem is due to there not being a set of inverse lower case characters in the lower case generator.

The solution given in Listing

```
387             PHA             ;Push video char on to stack
388             TAY             ;Sets Z flag = to accumulator
389             BEQ LOOP1       ;Branch to output if cursor
391             LDA F6          ;Get ASCII code of output character
393             ASL             ;Get rid of bit 7
394             CMP #C0         ;C0 = 60 shifted left. 60 = "'"
398             PLA             ;Get video value from stack
399             ORA #E0         ;Set bits 5,6,7 so Lowercase code
401             PHA             ;Push lowercase value
402     LOOP1   LDA F4          ;"Note: This section is as the
404             SEC             ; original except that the addresses
405             SBC BF11        ; and branch offsets have been
408             BMI LOOP2       ; changed."
410             CMP #28         ;
412             BCS LOOP3       ;
414             TAY             ;Transfer output position to Y
415             PLA             ;Pull video value from stack
416             BNE LOOP4       ;Branch if not cursor
418             LDA (F0),Y      ;Get current screen character
420             AND #40         ;Mask all but bit 6
422             SEC             ;Set carry flag
423             ROR             ;Move right, i.e. bit 7 = carry, 5 = 6
424             EOR (F0),Y      ;Combine character and mask
426     LOOP4   STA (F0),Y      ;Store character on screen
428             RTS             ;Exit
429     LOOP6   ROR             ;Comes here from 453
430             EOR (F2),Y      ;Combine character and mask
432     LOOP5   STA (F2),Y      ;Store character on screen
434             RTS             ;Exit
435     LOOP2   CLC             ;Same address as original
436             ADC #28         ;Add 40 (Decimal)
438             JMP DBBC        ;Jump out of routine
441             SEC             ;
442     LOOP3   SBC #28         ;Subtract 40 (Decimal)
444             TAY             ;Transfer output position to Y
445             PLA             ;Pull video value from stack
446             BNE LOOP5       ;Branch if not cursor
448             LDA (F2),Y      ;Get current screen character
450             AND #40         ;Mask all but bit 6
452             SEC             ;
453             BCS LOOP6       ;Always branch
```

Listing II: Modified Pascal output routine

```
program lowercase;
var buf: packed array [0..511] of 0..255;
      f: file;
      i: integer;

procedure firstbit;
  begin
    buf[387]:= 72;
    buf[388]:=168;
    buf[389]:=240; buf[390]:= 11;
    buf[391]:=165; buf[392]:=246;
    buf[393]:= 10;
    buf[394]:=201; buf[395]:=192;
    buf[396]:=144; buf[397]:=  4;
    buf[398]:=104;
    buf[399]:=  9; buf[400]:=224;
    buf[401]:= 72;
    buf[402]:=165; buf[403]:=244;
    buf[404]:= 56;
    buf[405]:=237; buf[406]:= 17; buf[407]:=191;
    buf[408]:= 48; buf[409]:= 25;
    buf[410]:=201; buf[411]:= 40;
    buf[412]:=176; buf[413]:= 28;
    buf[414]:=168;
    buf[415]:=104;
    buf[416]:=200; buf[417]:=  8;
  end;

begin (start of main program)
  reset(f,'system.apple');  (open file to be modified)
  i:= blockread(f,buf,1,5); (get relevant section of file)
  close(f);
  firstbit;
  buf[418]:=177; buf[419]:=240;
  buf[420]:= 41; buf[421]:= 64;
  buf[422]:= 56;
  buf[423]:=106;
  buf[424]:= 81; buf[425]:=240;
  buf[426]:=145; buf[427]:=240;
  buf[428]:= 96;
  buf[429]:=106;
  buf[430]:= 81; buf[431]:=242;
  buf[432]:=145; buf[433]:=242;
  buf[434]:= 96;

  buf[447]:=240;

  buf[450]:= 41; buf[451]:= 64;
  buf[452]:= 56;
  buf[453]:=176; buf[454]:=230;

  (Output routine is now modified in memory.)
  reset(f,'system.apple');
  i:= blockwrite(f,buf,1,5); (Now write the modified code to disk)
  close(f);
end.
```

Listing III: After executing this program the machine must be turned off to boot it with the modified System.Apple. A reset does not reload the System.Apple file.

II gets around this by displaying the corresponding upper case character in flashing video when the cursor is over a lower case character.

Listing II shows the final modified code in assembler form along with comments. The main points to notice are:

- 388 – TAY causes the Z flag to reflect the accumulator state and only uses one byte instead of two for a CMP.
- 391 – No check on Ctrl-E flag always displays upper and lower case.
- 393 – ASL removes bit 7 using only one byte instead of two for AND.
- 399 – For example, "A" = C1 ORA #E0 gives E1 = "a".
- 402 – Note LOOP1 is now at a different address and so all branch offsets have been altered.
- 414-428 and 444-453 + 429-434 are the two output sections. The second is split to allow the code at 435-444 to remain at its original address.
- If the output is the cursor then the current screen character is fetched and masked to bit 6.

This bit is only set in lower case characters.

The ROR instruction means that the two states the mask can take are: 80 for upper case and A0 for lower. So:

81 = "A" EOR 80 = 01 = "A" inverse video

E1 = "a" EOR A0 = 41 = "A" flashing video

Listing III is the Pascal program which modifies the System.Apple file to the above form.

Once the program is compiled it should be run with the boot disc (A COPY) in drive 1. After the program has finished turn off the power then reboot the machine.

So there it is – instant upper and lower case.

Talking of Apples . . .

THE Voice Master speech, voice recognition and music synthesiser system is now available for the Apple II+, IIe and IIc from Convox.

The speech synthesiser is basically a digital tape recorder with up to 64 different words, phrases or other sounds, which can be in memory at any one time.

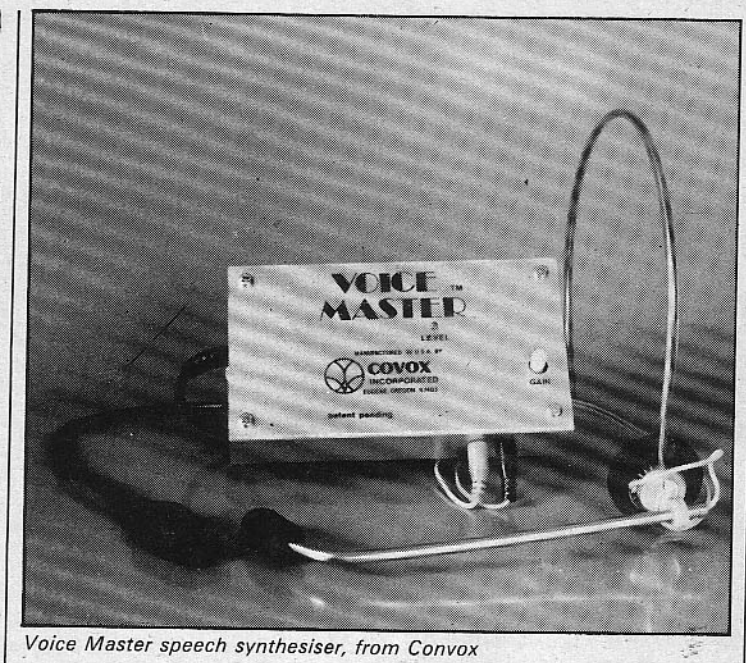
Speech recognition is achieved by storing the words or phrases needed and "training" the system to recognise them.

The Voice Harp allows music to be composed and performed in real time by humming or whistling.

Also available is Soundmaster, a 3-voice plug-in music/sound synthesis board for Apple II, II+ and IIe.

The price is $89.95 for Voice Master disc software and $39.95 for Soundmaster.

● *Convox, 657-D Conger Street, Eugene, Oregon 97402, USA. Tel: 0101-503 342 1271.*



Voice Master speech synthesiser, from Convox

Learn a language

A RANGE of language learning programs has been developed for the Apple II series which its manufacturer claims cuts teaching time by more than two-thirds.

From Linkword, the courses cover basic vocabulary and grammar and come in several versions – French, German, Spanish, Italian, Russian, Greek, Dutch and Portuguese.

Priced £29, each package includes an audio cassette.

Unlike other language course Linkword's deviser Mike Gruneberg has used psychological principles rather than normal teaching methods.

And one company willing to testify to the success of these is Thomson Holidays who agreed to four of its managers acting as guinea pigs. "It took around 12 hours to teach them about 400 words and a basic grammar, a regime that would normally take about 40 hours using traditional techniques", said a spokesman.

Gruneberg explained that his method involved linking foreign words to accoustically similar English words and giving a visual rather than verbal association.

So for playa, the Spanish word for beach, students are told to imagine a pair of pliers. Other pictures are used to denote gender.

● *Linkword, 100 Baker Street, London W1M 1LA. Tel: 01-935 1470.*

Prolog available

THE programming language Prolog, chosen as the basis of the Japanese fifth generation computer system project, is now available for the Apple IIe and IIc from Logic Programming Associates.

Micro-Prolog is designed to process knowledge. The programmer describes the problem to be solved by stating facts and rules about the problem which micro-Prolog then uses to search for possible solutions.

The price of £85.00 includes a reference manual and copy of the book "Start Problem-Solving with Prolog" by Tom Conlon.

● *Logic Programming Associates, Dept SPR/1, Studio 4, The Royal Victoria Patriotic Building, London SW18 3SX. Tel: 01-871 2016.*

Combination pack

A COMBINED package consisting of its CP/M Plus system and Wordstar-Mailmerge is on offer to Apple IIc owners from Cirtech.

The company has set the price at £280 but normally the word processing package alone would cost about £400.

Cirtech's system itself enables the IIc to run any standard CP/M programs including Calcstar and dBase, and to utilise the machines full 128k RAM.

It also offers printer, disc drive and modem support.

Other features include ToolKey and MouseKey functions, a disc-based help system and support for a high capacity external disc drive.

The system consists of a Z80 hardware module and an advanced version of the CP/M Plus operating system configured especially for the machine.

It fits entirely inside the computer and is only activated when a CP/M disc is booted. The system is also fully compatible with Softcard CP/M version 2.23.

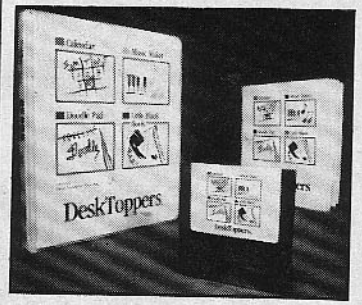
● *Cirtech (UK), Currie Road Industrial Estate, Galashiels, Selkirkshire, Scotland TD1 2BP. Tel: 0896 57790.*

Desk accessories

DESIGNED to improve personal productivity, Desk Toppers is a set of four accessories developed for the Macintosh by Harvard Associates.

A calendar, doodle pad, music maker and "a little black book" make up the set.

The calendar displays any month from the years 1904 to 2003 and the little black book



replaces the address and phone number book.

Music can be made on a xylophone, clarinet, trumpet or organ using Music Maker and ideas can be sketched out on the Doodle Pad.

The desk accessories can be copied on to any Macintosh software discs.

Price is £46.

● *P&P Micro Distributors, Todd Hall Road, Carrs Industrial Estate, Haslingden, Rossendale, Lancs. BB4 5HU. Tel: 0706 217744.*

C compiler and toolkit

A C COMPILER and tookit have been developed for the Macintosh by Consulair.

Mac C is a C compiler or

Space on disc

MAX PARROTT presents a Basic way to find out how many free sectors are left on a disc

BACK in the summer and autumn of 1982, *Windfall* carried a number of articles and comments on DOS patches which enabled the CATALOG command to print the number of free sectors left on a disc.

More recently a couple of readers have written asking how the same information can be accessed from within a Basic program. Now I'm sure there's more than one way of doing it but for what it's worth this is mine.

The listing is of a completely relocatable machine code subroutine which when called will give an output such as:

```
AVAILABLE:38 FULL:522
```

The routine is only assembled for DOS 3.3 in a normal 48/64k machine. It uses a Basic routine called LINPRT at $ED24 which prints out the decimal equivalent of a two byte hexadecimal number with the high byte in the A register and the low byte in the X.

This routine first stores the values in $9E and $9F. To save some space I have used these locations to store the number of free sectors – hence the slightly unusual method of storing high byte at lower address than low byte for the label FREE which has its value printed by LINPRT+4. The label USED is used more conventionally.

The bit pattern of the sectors is obtained from the VTOC by the routine $AFF7 which reads it into the DOS buffer. This is so that the sector map starts at $B3F3 (START).

I have used parts of the DOS error message routines to print the words "available" and "full" in order to save space and the Basic routine CHARPRT ($DB5C) to print characters so that the output can be in inverse, flashing or normal as required by the programmer.

The routine is relocatable so that it can be loaded from disc in to any area of memory and used. Alternatively it could be POKEd in from, or tacked on the end of, the Basic program.

A CALL to the start of it will activate the current drive and start the routine. No attempts at error messages have been incorporated. I think these are best left to the Basic program to handle.

The routine should only be used from Basic because of the zero page locations which it uses.

Because of its relocatable routine I think it is probably best to tag it on to the end of your Basic program.

To this end I have produced the Basic program shown in Listing I.

Type this in and save it to disc without running it.

Then run it and delete lines 10-26. Add the line:

30 CALL DISK

and run it again with a write-protected disc in the current drive in order to test it out.

If you get the correct response then everything is okay – you can continue typing in the rest of your listing.

To access the disc at any time just CALL DISK as in line 30. If the program did not give the correct response then you have probably made a typing error. Type FP, reload the first program, edit it and start again.

If you have already created your Basic program and wish to add this routine to the end of it follow this course.

Enter the monitor with a

→

```
10  DATA 169,0,133,159,133,158,133,157,133,156
11  DATA 169,8,133,155,32,247,175,162,137,189
12  DATA 243,179,160,7,74,144,8,230,159,208
13  DATA 10,230,158,208,6,230,155,208,2,230
14  DATA 156,136,16,236,202,240,12,169,255,69
15  DATA 157,133,157,208,220,202,202,208,216,32
16  DATA 251,218,162,143,32,6,167,169,58,32
17  DATA 92,219,32,40,237,162,104,32,6,167
18  DATA 169,58,32,92,219,165,156,166,155,32
19  DATA 36,237,76,251,218
20  RESTORE : LOMEM: 32 * 256
21 DISK = PEEK (175) + 256 * PEEK (176)
22  FOR I = DISK TO DISK + 94
23  READ Y: POKE I,Y
24  NEXT
25 H = INT (I / 256):L = I - 256 * H
26  POKE 175,L: POKE 176,H
27 DISK = PEEK (175) + 256 * PEEK (176) - 95
```

Listing I

```
1000- A9 00 85 9F 85 9E 85 9D
1008- 85 9C A9 08 85 9B 20 F7
1010- AF A2 89 BD F3 B3 A0 07
1018- 4A 90 08 E6 9F D0 0A E6
1020- 9E D0 06 E6 9B D0 02 E6
1028- 9C B8 10 EC CA F0 0C A9
1030- FF 45 9D 85 9D D0 DC CA
1038- CA D0 D8 20 FB DA A2 8F
1040- 20 06 A7 A9 3A 20 5C DB
1048- 20 28 ED A2 68 20 06 A7
1050- A9 3A 20 5C DB A5 9C A6
1058- 9B 20 24 ED 4C FB DA
```

Hex dump

CALL-155. Type AF.B0 and you will see two bytes. Suppose what you get is:

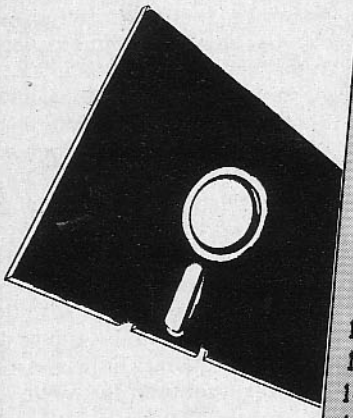
```
00AF- AD
00B0- 1E
```

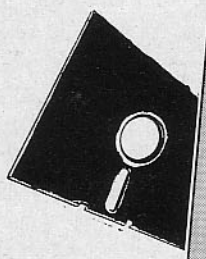
This means that the end of the Basic program is at the address $1EAD. Start to enter the routine at this address or BLOAD it from disc to this address if you have already BSAVED it.

Add 95 decimal to the address obtained from AF.B0. In our example that is $1EAD + $5F which gives 1F0C.

Now set the end of program pointer to this new address by typing AF:0C 1F Return to BASIC by typing Ctrl-C and SAVE your program.

The new code will be saved along with the Basic and can always be accessed by adding the lines 27 to the start of your program and a line such as 30 wherever you want to know the space.





```
009B                    1  USED     EPZ $9B
009D                    2  COUNTER  EPZ $9D
009F                    3  FREE     EPZ $9F
AFF7                    4  VTOCRD   EQU $AFF7
B3F3                    5  START    EQU $B3F3
ED24                    6  LINPRT   EQU $ED24
DAFB                    7  CRDO     EQU $DAFB
DB5C                    8  CHARPRT  EQU $DB5C
A706                    9  TEXTPRT  EQU $A706
0800                   10  *
1000                   11          ORG $1000
1000                   12  *
1000 A9 00             13          LDA #0
1002 85 9F             14          STA FREE
1004 85 9E             15          STA FREE-1
1006 85 9D             16          STA COUNTER
1008 85 9C             17          STA USED+1
100A A9 08             18          LDA #8
100C 85 9B             19          STA USED
100E 20 F7 AF          20          JSR VTOCRD
1011 A2 89             21          LDX #137
1013 BD F3 B3          22  LOOP1   LDA START,X
1016 A0 07             23          LDY #7
1018 4A                24  LOOP2   LSR
1019 90 08             25          BCC NOTFREE
101B E6 9F             26          INC FREE
101D D0 0A             27          BNE CONTINUE
101F E6 9E             28          INC FREE-1
1021 D0 06             29          BNE CONTINUE
1023 E6 9B             30  NOTFREE  INC USED
1025 D0 02             31          BNE CONTINUE
1027 E6 9C             32          INC USED+1
1029 88                33  CONTINUE DEY
102A 10 EC             34          BPL LOOP2
102C CA                35          DEX
102D F0 0C             36          BEQ DONE
102F A9 FF             37          LDA #$FF
1031 45 9D             38          EOR COUNTER
1033 85 9D             39          STA COUNTER
1035 D0 DC             40          BNE LOOP1
1037 CA                41          DEX
1038 CA                42          DEX
1039 D0 D8             43          BNE LOOP1
103B 20 FB DA          44  DONE    JSR CRDO
103E A2 8F             45          LDX #$8F
1040 20 06 A7          46          JSR TEXTPRT
1043 A9 3A             47          LDA #$3A
1045 20 5C DB          48          JSR CHARPRT
1048 20 28 ED          49          JSR LINPRT+4
104B A2 68             50          LDX #$68
104D 20 06 A7          51          JSR TEXTPRT
1050 A9 3A             52          LDA #$3A
1052 20 5C DB          53          JSR CHARPRT
1055 A5 9C             54          LDA USED+1
1057 A6 9B             55          LDX USED
1059 20 24 ED          56          JSR LINPRT
105C 4C FB DA          57          JMP CRDO
105F                   58          END

***** END OF ASSEMBLY
```

;READ VTOC INTO START

;PRINT X,A AS NUMBER
;PRINT CARRIAGE RETURN
;PRINT A AS CHAR
;PINCHED FROM DOS ERROR ROUTINE

;FIRST BYTE WITH DATA
;PICK UP BIT PATTERN

;LOOK AT EACH BIT

;DO ALL BITS

;DONE 1 BYTE OR TWO?

;SKIP TWO BYTES
;MORE TRACKS

;A COLON

Assembly listing

Call this progress?

APPLE'S new ProDOS is bringing us welcome benefits, or so we're supposed to believe.

One benefit claimed for the awkward operating system is faster working programs.

OK then. So why can't someone write a halfway decent spelling checker for my newfangled, ProDOS style, word processor files?

The other day I put a 2,225 word article on a DOS text file through my ancient – 1982 – Sensible Speller 3.0.

I timed how long it took to get the file, count the words and compare it against the dictionary.

Crusty ol' Sensible Speller 3.0 zipped through the job in 69 seconds. Not bad.

Then I converted the article to an Appleworks file and put it through a friend's newer Sensible Speller for ProDOS.

Faster still? Well not exactly. It clocked up 149 seconds, more than double the DOS 3.3 version time.

Not only that, but using the program was decidedly messy. It required a knowledge of ProDOS conventions for a start. A friendly little number!

Anyway, having negotiated the first part of the checker, I thought I should at least add some Kiwi words to the dictionary disc. So I followed the tortuous instructions, and was finally invited to build my enhanced dictionary on a new pre-formatted disc.

Unfortunately I wasn't given the option of doing it on both my disc drives.

"Put the old dictionary in the drive/put the new dictionary in the drive/old/new/old/new" – on and on it went. Must have been at least 50 swaps before I gave up and bowed to Amurican spellin.

There's got to be a better way, I told myself as I trotted off to a friendly computer store in search of other ProDOS spelling checkers.

There was only one on offer – Megaworks, by the Megahaus people. Now they have glossy ads in A+ and InCider. Should be good, and a snip at NZ$300.

A snip? Well, in weak defence of the hefty price I should point out that the program also does mail merging – not that I often feel an urge to merge (mail that is, Hortense!).

I brought out my 2,225-word file and booted up Megaworks with eager anticipation. This early in the financial year the office budget could easily stand $300 for a really worthwhile product.

Right from the start things looked better. The screen format was almost identical to Appleworks and you didn't have to pussyfoot about with pathnames.

But wait! And wait. And wait . . .

And wait some more. To be exact, wait for 568 seconds. Shiny new Megaworks did the job, and it only took eight times longer than my antediluvian Sensible Speller 3.0, running on obsolescent DOS 3.3.

This is progress? I didn't wait to try megamerging. – **John MacGibbon, New Zealand.**

Pascal printing

I HAVE an Olympia ESW 100-RO printer connected to an Apple II+.

In order to alter the horizontal pitch of the above printer, one has to use a short series of instructions in Applesoft:

```
10 PR#1
20 PRINT CHR$(9);
   CHR$(75); CHR$(13)
30 PRINT CHR$(27);
   CHR$(13);CHR$(05)
40 PR#0
```

to produce 12-pitch – the '05' in line 30 specifying the number of sixtieths of an inch for spacing.

However when I do the equivalent thing in Pascal (or maybe it's not equivalent!) it does not work:

```
REWRITE(F2, 'PRINTER:');
WRITELN(F2,CHR(9),
   CHR(75),CHR(13));
WRITELN(F2,CHR(27),
   CHR(13),CHR(05));
CLOSE(F2);
```

where F2 is defined to be a text file.

Can anyone throw any light on the problem? – **Christopher R. Harding, Cotham, Bristol.**

● It's difficult to know without access to a system like yours.

The only thing we can think of is that the printer is expecting the high bit of the codes set which is what Applesoft sends, whereas it's reset with Pascal.

Are there any switches or codes to change on the printer or interface?

Macclub Benelux

WE are a fully independent, international Macintosh user group with members in Belgium, Holland, Luxembourg, France, Switzerland and Germany. We are interested in having contacts with other user groups all over the world.

We don't publish a printed newsletter – our information is on disc. The "Macclub Diskette" is published in Dutch, German and French, and if there is enough interest from abroad, we will also start an English version.

On the Macclub Diskette we give tips, tricks, latest reviews and descriptions of hardware and software, public domain software and many more.

We would be very happy if you could inform your readers about our activities. Thanks a lot. – **Hubert Savelberg, Macclub Benelux, Wirtzfeld Avenue 140, B-4761, Büllingen, Belgium.**

Basic WP

I HAVE written a word processor for my ITT 2020 computer, which has Applesoft ROMs, in Applesoft Basic.

As it stands I am pleased with my work but I wish I could use my program's features on my Applesoft programs to make it a sophisticated program editor.

Is there any simple way, without using machine code, that I could load an Applesoft program into my word processor, as though it were a text file, edit it and then write it out again and the updated version be stored as a Basic program and not as a text file?

I have also recently bought the Apple language card for my ITT but when I type "?FRE(0)/1024" to see how much memory I have available the same amount is returned as though I still have a 48k machine.

How do you use the language card so that I have an extra 16k free memory?

Now for some praise, your magazine is very good, I have been buying it since it changed its name from Windfall.

The Appletips are great and I'm glad you now have more of them.

It's also good to see that you have a good selection of Macintosh and Apple II articles because we II owners like to know what's happening with the 'old faithful' as well as what's new on the scene. – **Duncan Eadie, High Wycombe, Bucks.**

● We're sorry but there is no easy way to load a Basic program into your word processor and then put a revised version back to disc because the program is not stored completely as text.

The Basic keywords are in a tokenised form where 1 byte – with high bit set – stands for each word, line numbers are in a hexadecimal format and text and line numbers in GOTO's and GOSUB's are in Ascii format (high bit reset).

You will have to write a translator program to do what you want.

The language card's RAM is in parallel – that is, shares the same address space – with the Basic ROMs so it cannot easily be used.

There have been *Apple User* articles on its use from Basic but its primary use is for other languages and operating systems.

UNFORTUNATELY, the phone number for Stephen Kearon's Dublin-based bulletin board (see *Feedback*, September 1985) ended up on the cutting-room floor.

The number is Dublin 885634. We also managed to spell Stephen's name wrongly, for which we apologise.

Memory map, please

I WOULD like any information on the memory layout of the IIe. If the machine has an extra 16k of RAM over the II+, how can it have the same memory locations?

It appears that to access the speaker on the IIe from Basic you need to make machine code subroutines and then CALL them.

As I don't consider myself a machine language expert, is there an easier way to do this?

No matter how hard I look, I still can't seem to get the control over the speaker that I was hoping to get. Is there any way I can create my own machine code subroutines without learning the whole damn language?

I am basically familiar with it and I've done a few bits here and there, but I don't think I know nearly enough to write my own subroutines which I think need to use the speaker.

If you can't help me, maybe you could suggest a few books to look at as I haven't seen any books on the subject, either.

I use Mousepaint a fair bit, but I do wish there was some way to add your own fonts like you can on the Mac.

One thing I have tried is to add my own fonts by renaming, say, the font TORONTO on a backup disc to TORONTO. ORIGINAL or something, then substitute my own font by saving it as TORONTO.

Then when I run my backup of Mousepaint, the font I have created will be there instead of the original TORONTO font.

This gets rather complicated, and besides, it doesn't work – to my dismay!

Is there any way I can customise Mousepaint to my own fonts and make it print on an Epson printer with a Grappler card from within the program, instead of going to the slave disc, loading the program, and printing it that way?

Not knowing much about machine language is very frustrating, as you can see from my problems. Could you please come to my rescue? – **Steven McIntyre, New South Wales, Australia.**

● We've covered the question of the IIe's extra 16k recently – see *Feedback*, August 1985.

For sound routines, you could start with Mark Bowyer's Ampersand routines from the June 1985 issue.

We don't know how to change fonts in Mousepaint, having neither a card or the software. Perhaps one of our readers can help?

Trouble with the accounts

I AM having terrible trouble trying to get the Sage Accounts program to run satisfactorily on both of my Apple II+ systems.

We are using Rosco Z80 cards, and the software manufacturers say the problem exists with the hardware.

The hardware has been checked out by our Glasgow Apple dealer and would appear to be working properly.

The software suppliers also say, however, they have no records of customers to whom they have supplied Sage Accounts to run on Apples.

Might I use your columns to ask whether any readers are using Sage Accounts on Apple II machines successfully or not, and whether they would be interested enough to contact me direct at 041-331 2834? – **James Cuthbertson, Glasgow.**

Better solution

I AM a 16-year-old reader of your great magazine here in The Hague, and while at school I use an Apple IIe.

Now that all my examinations are finished, I would like to call attention to the fact that there is a better solution to J.P. Holden's problem with Apple Pascal 1.1 than the one proposed by Max Parrott (June 1985).

My solution, outlined below, is the one I use at school, and is really a modification of that described as the "official solution" in InCider magazine.

You need, in short, all four discs – not forgetting to have duplicated them all – in the Apple Pascal 1.1 package, and two blank discs.

I assume that Mr Holden is in possession of the Apple Pascal Operating System Manual, and has it open so that he can see how to transfer files using the T command.

Now, first boot up with APPLE1: and when you see the welcome message, press F for Filer.

Then transfer FORMATTER.CODE from APPLE3: on to APPLE1:.

Put APPLE1: back into #4 – I shall use Pascal drive addresses throughout – and press QX, then FORMATTER (Return).

When it asks "Which drive?", put one of the blank discs in #4 and press #4 (Return).

Do the same with the other disc, and when both discs have been formatted, type 0 (Return) to quit the program.

Then type FC. Then type in BLANK:, put one of the newly formatted discs in #4, then Return, and finally APPLE4: (Return). Do not forget the colons.

Do the same with the other newly formatted disc. This done, write on the label of one of them APPLE4:(a), and on the other APPLE4:(b).

Now press T for transfer, and then transfer SYSTEM.PASCAL and SYSTEM.MISCINFO from APPLE0: on to both APPLE4: discs, making them the first files on both discs and placing them in the same order.

Now transfer SYSTEM.APPLE from APPLE3: on to APPLE4:(a) only. Then press T for transfer again, then APPLE0:SYSTEM.? (Return), and finally APPLE4:$ (Return).

Respond with a Y to all questions except "Transfer SYSTEM.PASCAL?" and "Transfer SYSTEM.MISCINFO?". If you prefer to programme in assembly language, you can respond with an N to "Transfer SYSTEM.COMPILER?" and then transfer 6500.OPCODES and 6500.ERRORS from APPLE2: on to APPLE4:(b).

Let me remind you that the transfers described in the previous two sentences are only to involve APPLE0:, APPLE2: and APPLE4:(b).

Now you can boot up with APPLE4:(a) and then, after the welcome message, put APPLE4:(b) into 4 and start writing, compiling and debugging programs. – **Seija C. Teramoto, The Hague, The Netherlands.**

Pal talk

I HAVE friends who say there is no such thing as an 80 column card with 64k for an Apple II+.

I have other friends who say there is. I would be most grateful if you could shed light on the matter! – **Matt Mick, London.**

● We have friends like that too! We don't know of one – does anybody have a friend who's definitely got one?

Disc space

I HAVE been hunting high and low for a program which can automatically show sectors free at the beginning of the menu.

My Appleplot discs do have this facility, but somehow I cannot get them to list this particular program which automatically changes as new items are added or subtracted.

Any hope of a simple listing for such a program please? – **F.E. Brooks, Melaka, Malaysia.**

● You're in luck! See Max Parrott's article on Page 53 of this issue.