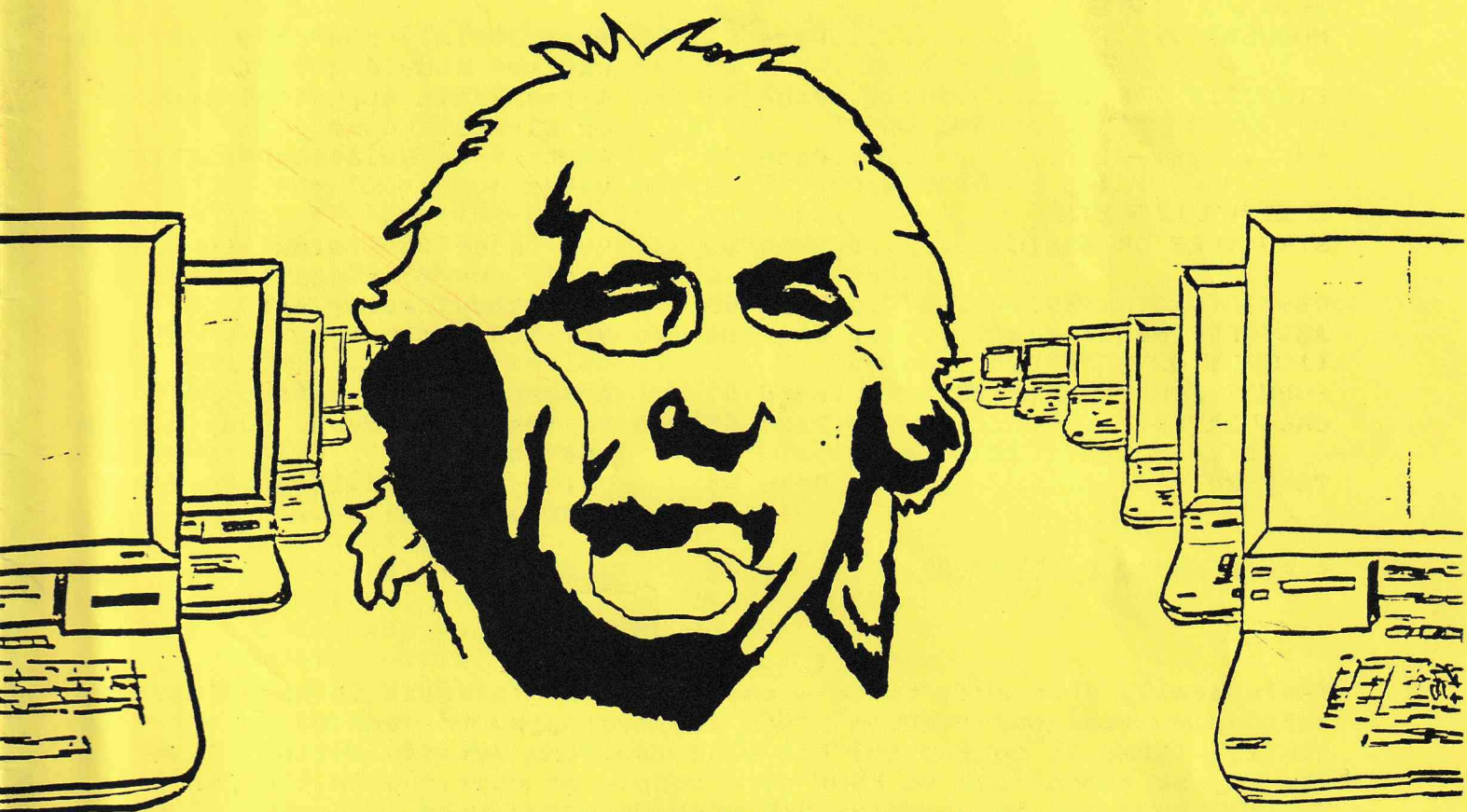


TATUNG Einstein



UK EINSTEIN USER GROUP
NEWSLETTER

Volume 2 Number 4 February 1987

CONTENTS

REGULARS

Editorial.....	Page 58	Getting there
Letters.....	Back Page	Your page for comments
Adventurers.....	Back Next Month	
BOOBS.....	Page 59	This will probably become a regular however hard we try.

REVIEWS

MODULA2.....	Page 72	What it is, how it works and why you should get it
EINSTEIN 256.....	Page 63	A realistic appraisal from an ex Einstein user.
AM.....	Page 75	Amstrad simulation without tears

FEATURES

STRUCTURE OF BASIC.....	Page 61	For those who wish to add their own routines to it
DISC DIRECTORIES.....	Page 63	What they really are
ALLSORTS NO 3.....	Page 64	Now it's up to you
UTILITIES		
SUBMIT FOR THE EINSTEIN.....	Page 67	A must for the serious
CALCULATOR.....	Page 65	Do 'BASES' confuse you, here's the answer
TEXT 25.....	Page 58	Using the 25th line to put your messages on
SPECIAL NOTE.....	Page 61	Read it

EDITORIAL

There really does seem to be a resurgence of interest in the Einstein, not only from our members, who are beginning to send us some good stuff, (keep it coming in) but also from the outside world. A lot of it may be rumour but we KNOW of a couple of upgrades in the pipeline. We will keep you informed as and when more concrete information comes our way.

The Amstrad Emulator in this issue looks to be well worth looking at along with the utilities from the same house. MATMOS at Lingfield can still supply 3" drives, single sided at £29.95 + vat and double sided at £39.95 + vat. Cables are £10 + vat. (Tel 0444 73830)

Now we change to Humble mode. Apologies for the print and photo copy quality. this is due to new ribbons on the printer and the photocopier over-running it's service date. (Beyond our control) If we can get another 50 members we should be able to go into PRINT so get out there and encourage everyone you know with an Einstein to join instead of lending them your copy!

No response to our suggestion about a bulletin board. Do you want one???

Did anyone note the deliberate mistake in last months issue. PAGE 49A. I don't know what came over me, I can usually count better than that. Thanks to Keith for coming to the rescue when he put it to bed. I would not have been at all popular if you had all been looking at a blank page. Enough of this drivel, lets get on with getting this one on the road and in the post and put to bed and through your letterboxes.

BOOBS

Diary.

Graham Higgins

I managed to get your Diary program working (UKEUG Newsletter November V2 No.1). I have some corrections and an improvement.

1. The variable N\$ never receives an assignment and is therefore redundant, thus;

Line 205 should be deleted and the N\$ in line 30 also deleted

2. Line 305 is an error and should be deleted.

3. In line 310, you have neglected to account for the case where $I+8>D$, i.e. where T\$ ("today") is, for example 12/28. The indexing of the D\$ and M\$ string arrays by the variable F should be replaced by $F \text{ MOD } 366$:

D\$(F MOD 366) and M\$(F MOD 366)

4. There are a couple of changes required for the FIRST run of the program, i.e. before the data file is created.

* the GOTO in line 20 should be changed to GOTO 500, (temporarily).

* line 190 should be temporarily renumbered as 191

* a new line 190 should be inserted:

190 CREATE "DIARY.DAT",FD\$:CLOSE

* line 500 should have :GOTO 80 appended

The program can now be run. AFTER the DIARY.DAT file is created- (which must be done by adding a message to the diary apparently),

the GOTO 500 in line 20 can now be replaced by GOTO 200

Line 191 can be renumbered to 190 and line 191 deleted.

lines 500 to end of file can be deleted.

Incidentally, here's a way of reducing some of the tedious DATA statement typing:-

Replace lines 00 to 980 with:-

500 DIM B\$(12)

510 DATA 31,28,31,30,31,30,31,31,30,31,30,31

520 FOR X=1 TO 12:READ B\$(X):NEXT:C=0

530 FOR X=1 TO 12:FOR Y=1 TO EVAL(B\$(X))

540 D\$(C)=STR\$(X)+"/"+STR\$(Y):C=C+1

550 NEXT Y:NEXT X:GOTO80

Two other changes are necessary, IOM 5,0 should be entered as line 5 and IOM 5,1 should be entered as line 194.

Ed's Note. Thank you very much for your comments, our only excuse is that the original program was written on and for a Video Genie and ported across to the Einstein on the RS232. Some changes were then made but only enough to get it working.

25 LINE TEXT by Stephen Eccleston

The following program allows the user to type, edit and display a message on the 25th line of the 80 column display. The message will be displayed so long as the screen is cleared only with 'HOME (1EH) and 'CTEOS' (16H), as detailed in the Einstein User, volume 2 No.1 page 4. It would also be useful to see a count of the number of characters typed in the message, on the screen, but I have not yet been able to turn the Hex value in the 'B' register into a decimal value and print it to the screen. Can you help with this, or perhaps a reader might have the answer.

;THIS PROGRAM ACCEPTS OPERATOR INPUT TO GENERATE A 25th LINE

ORG 100H

LOAD 5000H

TEXT: EQU 8000H

BDOS: EQU 0005H

LONG: EQU 50H

MESOUT: EQU 09H

ESC: EQU 1BH

GS: EQU 1DH

LINE24: EQU 17H

```

COL1:    EQU 0
SPACE:   EQU 20H
HOME:    EQU 1EH    ;Code to home the cursor
CTEOS:   EQU 16H    ;Clear to end of screen code
DOLLAR:  EQU 24H    ;$ = Terminating Char with MESSOUT
DEL:     EQU 19H
CURSOR:  EQU 5FH    ;Cursor Character '_'
                    ;START:RST 8
                    ;DB 0D3H;INITIALISE 80 COL CARD
                    ;
                    ;CLEAR TEXT AREA-FILL WITH SPACES 20H
                    ;
CLTEXT:  LD HL,TEXT    ;START OF TEXT BUFFER IN MEMORY
          LD A,SPACE    ;Character to fill buffer with
          LD B,LONG     ;Length of text buffer
NSPACE:  LD (HL),A      ;Store space in text buffer
          INC HL        ;increase pointer in memory
          DJNZ NSPACE   ;repeat until 'B' reg =0
          RST 8
          DB 0BFH;      DISPLAY EINSTEIN LOGO
          ;
DISP1:   LD DE,MESS1
          LD C,MESOUT
          CALL BDOS      ;PRINT OPERATOR PROMPT MESSAGE
                    ;ACCEPT 80 CHRS USING MCAL 9CH
          LD B,LONG      ;loop counter
          LD HL,TEXT     ;start of text buffer
          LD D,H         ;pointer for
          LD E,L         ;25th line MCAL
          LD (HL),CURSOR
          CALL PROG25     ;call the programming routine
LOOP:    RST 8           ;get char from keyboard
          DB 9CH         ;using MCAL
          CP 0DH         ;is it carriage return?
          JR Z,FINISH    ;YES so jump to end
          CP DEL         ;is it a DELETE character
          JR Z,DELETE    ;YES so go and delete
          LD (HL),A      ;ELSE store it
          INC HL         ;increase loop counter
          LD (HL),CURSOR ;
          CALL PROG25    ;call programming routine
          DJNZ LOOP      ;repeat till 'B' reg = 0
FINISH:  CALL PROG25     ;program the 80 col card
          LD DE,MESS2    ;message to CTEOS and HOME
          LD C,MESOUT    ;
          CALL BDOS      ;
          RET            ;return to system
          ;
PROG25:  PUSH HL        ;save text buffer pointer
          PUSH DE        ;save text pointer
          PUSH BC        ;save loop counter
          RST 8
          DB 0D4H
          POP BC
          POP DE
          POP HL
          RET
          ;
DELETE:  CALL CHKLH     ;at L.H. end of line ?
          JR Z,LOOP      ;YES so return
          LD A,SPACE
          LD (HL),A      ;erase cursor
          DEC HL

```



```

INC B
LD (HL),A      ;erase last character
LD (HL),CURSOR
CALL PROG25    ;program with new line
JR LOOP

;
CHKLH:  PUSH HL      ;save text buffer pointer
        PUSH DE      ;save start of text
        EX DE,HL     ;put pointer in de
        SBC HL,DE     ;set zero flag if at L.H.end
        POP DE       ;restore text start
        POP HL       ;restore text buffer pointer
        RET

;
MESS1:  DB ESC,GS,COL1,LINE24
        DB 'TYPE YOUR MESSAGE TO BE '
        DB 'DISPLAYED.....'
        DB 'MAXIMUM 80 CHARS. CR TO END..'
        DB '.....',DOLLAR
MESS2:  DB HOME,CTEOS,DOLLAR
;
        END
    
```

REPAIRS

SCS Components of 218 Portland Rd., Hove Sussex (Tel 0273 770191) will repair your Einstein. Either take it in or send it to them in the post and they will effect repairs as quickly as they can. They also supply all manner of bits and pieces and if they haven't got it they will get it.

THE STRUCTURE OF BASIC

Chris Giles

There are a number of facilities that are missing in BASIC, such as a REM FILL to remove all REM statements after completeng a program to make it run faster, or a routine to replace all the long variable names with single letter variables, again to make it run faster or give more working space once it is debugged. In order to write routines that alter a BASIC program in any way it is essential to know something about the way the program is stored in memory and hopefully this article will give some clues on just this. First of all the KEYWORDS are held in memory as TOKENS, i.e. each time you type in PRINT at the keyboard in a program line it is converted to Hex. A2 (decimal 162). This is done to save space but can be a bit confusing when looking at memory. To help you I have listed them all below along with a little routine to check them out yourself.

```

10 GOTO30
20 : :
30 FOR X=128 TO 245
40 POKE &3E0E,X
50 PRINTX,HEX$(X,2),:LIST20,1,20
60 NEXT
70 FOR X=&3E00 TO &3EBF
80 A=PEEK(X)
90 PRINTEX$(A,2),
100 NEXT
    
```

The action is in line 20. Line 40 pokes the TOKENS into location &3E0F which is the memory location where the KEYWORD between the two colons is stored in line 20. Line 50 Prints the TOKEN (X) and then Lists line 20 only, giving the actual keyword. We start at 128 because all the TOKENS have bit 7 set.

Here is the list.

80	AUTO	A0	POKE	C0	TI\$	E0	ERR
81	CHAIN	A1	POP	C1	TEMPO	E1	ERL
82	CLEAR	A2	PRINT	C2	VOICE	E2	EOF
83	CLOSE	A3	READ	C3	PSG	E3	FN
84	CLS	A4	REM	C4	ABS	E4	INCH
85	CONT	A5	RENUM	C5	ASC	E5	KBD
86	CREATE	A6	UNPLOT	C6	ATN	E6	MUL\$
87	DATA	A7	RESTORE	C7	CHR\$	E7	NOT
88	DEF	A8	RETURN	C8	COS	E8	PI
89	DEL	A9	RUN	C9	DEEK	E9	SIZE
8A	DIM	AA	SAVE	CA	EVAL	EA	234
8B	DOKE	AB	PLOT	CB	EXP	EB	235
8C	DRIVE	AC	STOP	CC	HEX\$	EC	236
8D	ELSE	AD	SWAP	CD	INP	ED	237
8E	END	AE	VERIFY	CE	INT	EE	238
8F	FOR	AF	WAIT	CF	LEN	EF	239
90	GOSUB	B0	FMT	D0	LN	F0	240
91	GOTO	B1	APPEND	D1	LOG	F1	241
92	HOLD	B2	DIR	D2	PEEK	F2	242
93	IF	B3	ERA	D3	POINT	F3	243
94	INPUT	B4	LOCK	D4	POS	F4	244
95	LET	B5	REN	D5	RND	F5	245
96	LIST	B6	UNLOCK	D6	SCRN\$	F6	246
97	LOAD	B7	MUSIC	D7	SGN	F7	247
98	MGE	B8	CALL	D8	SIN	F8	248
99	MOS	B9	IOM	D9	SQR	F9	249
9A	NEW	BA	NULL	DA	STR\$	FA	250
9B	NEXT	BB	PTR	DB	TAN	FB	251
9C	OFF	BC	SEP	DC	VAL	FC	252
9D	ON	BD	SPEED	DD	LEFT\$	FD	253
9E	OPEN	BE	WIDTH	DE	MID\$	FE	254
9F	OUT	BF	ZONE	DF	RIGHT\$	FF	0

The ones here are the Standard Reserved Words. The Auxiliary Reserved Words are not listed. (Try writing a routine to list them)

The second part of the program from line 70 to line 100 will list the program as it is stored in memory in hex.

This is the equivalent MOS tabulation:-

3E01	08 00 0A 00 91 33 30 00	10	GOTO30
3E09	08 00 14 00 3A 20 3A 00	20	:
3E11	12 00 1E 00 8F 20 58 7E	30	FOR X=128 TO 245
	31 32 38 20 72 20 32 34		
	35 00		
3E23	0E 00 28 00 A0 20 26 33	40	POKE &3E0E,X
	45 30 46 2C 58 00		
3E32	18 00 32 00 A2 58 2C CC	50	PRINTX,HEX\$(X,2),:LIST20,1,20
	28 58 2C 32 29 2C 3A 96		
	32 30 2C 31 2C 32 30 00		
3E4A	06 00 3C 00 9B 00	60	NEXT
3E4F	16 00 46 00 8F 20 58 7E	70	FOR X=&3E00 TO &3EBF
	26 33 45 30 30 20 72 20		
	26 33 45 42 46 00		
3E65	0B 00 50 00 41 7E D2 28	80	A=PEEK(X)
	58 29 00		
3E71	0D 00 5A 00 A2 CC 28 41	90	PRINTHEX\$(A,2),
	2C 32 29 2C 00		
3E7F	06 00 64 00 9B 00	100	NEXT
3E85	00		End Of Program Delimeter
3E86	00 00 00 00 00 28 7A 8E		Rest of memory

It all looks very confusing, doesn't it? Well hang on in there. The program listing actually starts at 3E01, so this is where to start to decipher the (very) cryptic code.

The first two bytes give the number of characters in the following line, so 08 00 gives 08decimal. The next two bytes together give the Line Number, so 0A 00 when converted gives 10decimal. The next byte is the TOKEN for GOTO followed by the ASCII code for the line, i.e. 30. The next character is the End of Line character which is Zero.

So our first line is made up of

- 1) the number of characters in the line
- 2) the line number
- 3) a TOKEN for the KEYWORD
- 4) the parameters relating to the KEYWORD in ASCII
- 5) the End of Line Delimeter.

the next line is also 8 bytes long, and so on.

NOTE The way to calculate line numbers (or line lengths) is to convert BOTH bytes to decimal, multiply the SECOND by 256 and add to the first. In this short program the second byte is always Zero so can be ignored but as an example the Line No. 1000 would be stored as E8 03. ($3 \times 256 = 768 + 232$) (E8hex is 232decimal)

DISC DIRECTORIES C.P. WALLIS February 1987

There is an error in the article on Discs in the November issue of the Newsletter which is reprinted in the December issue. Directory entries do not "form what is known as a File Control Block (FCB)". FCB's should normally only be found in RAM: they are in fact records which programs must format for the purpose of accessing files. The first 14 (and possibly the last 3) bytes of the FCB are written by the program and the remaining 19 bytes are filled in by DOS when the file is opened, by copying from the directory entry on the disc. A directory entry occupies 32 bytes and at least one must be continuously present for each file on the disc; an FCB is only required when the file is in use.

An FCB for an opened file is therefore almost a copy of the directory entry, so the article on Discs will be almost correct if you replace 'FCB' with 'directory entry'. Although there are minor differences which result from file processing, there is one major difference which the article implies does not exist: this concerns the first byte, which in an FCB specifies the drive to be read whereas in a directory entry the first byte is the user code. Program user codes can be any number from 0 to 31, but command user codes are restricted to values less than 16. The purpose of user codes is to be able to keep your files in separate pigeon-holes: you will normally only be able find (with DIR or when opening) files in the current user area.

Unfortunately you can't do much with user codes because (a) there are no commands to implement them and (b) Einstein DOS takes every opportunity to change your user code back to zero. I have a set of enhanced commands to implement user codes and I hope to complete debugging them shortly and make them available.

EINSTEIN 256

Vic Day

At the National Einstein User Show in November I was able to talk to Roy Clark and David Bell about the 256. As a result I put the old Einstein up for sale and a month later I bought the 256. It is, as you said at the show, not a great deal better in performance than the Einstein apart from the graphics which are really something, the screen, which is called Hi-Res is much better than the old Einstein for 80 column resolution and is easily readable, but in terms of resolution I think it should be medium Hi-Res, a true Hi-Res can

cost hundreds of pounds, but the 256 monitor is more than adequate. I have rewritten a few programs with the graphics which show the superiority of the 256 over the Einstein as you can have several colours within the pixel area, the Einstein sets the colour to whatever the last colour set in the area of the pixel.

Martin Page writing about his visit to the Tatung factory comments that he never saw any 256's on the line, well that is easily explained, the computers are made in Taiwan and the monitors only are British made.

He also mentioned that they are not in the shops, well I had no bother getting one from Screens. There does however seem to be a lack of information about the second disc and on the printer lead. Screens have been looking into it for some while without result, and a query to Tatung about the parallel interface has brought no result.

I did, however get a quick reply from David Bell about the compatibility with the Einstein with such programs as Wordstar, Cracker etc. When I first booted these up all I got on the screen was a series of graphic symbols where the text should be. The cure was simple, whilst still in MOS enter L1 to select the ASCII character set. It was unfortunate that David's letter said select L2 which selects the German character set which doesn't work, however it was obvious what the fault was.

From my six weeks experience with the 256 I find it much easier to use than the Einstein, it has proper cursor keys with no need to use SHIFT to select right or up. It took some time to find the RESET key which is not surprising since one doesn't exist, however by pressing Ctrl, Alpha Lock and Graph at the same time you get a RESET, a combination of keys that you are not likely to press accidentally.

A couple of minor grouses about the newsletters, firstly I wish you would put the authors name at the front of the article, sometimes a new article runs straight after a previous article without heading or the authors name, secondly the printing gets very faint at times making it difficult to read, which is even worse if it occurs during an listing.

ALLSORTS NO 3

Chris Giles

All sorting routines are a compromise between time, space and effort. The Exchange sorts, as we have seen, are small in size and easy to program but tend to take their time. Sorts based on Binary trees and recursion are very quick and efficient but are very difficult to program (I haven't attempted one yet).

The best all round sort that I have come across is the SHELL sort, named after it's inventor D.L.Shell. It is only slightly harder to program than the exchange sorts and takes up little more room, if any. The SHELL sort runs quickly because it makes fewer passes on average than other sorts. It does this by comparing half the table size apart, then it compares items a quarter of the table size apart, halving the distance apart of the items compared till the distance is 1, or adjacent items. For example if it is sorting a table of 64 items it will compare 1 and 33, then 2 and 34, then 3 and 35. When that pass is finished it will compare 1, 17 and 33, then 2, 18 and 35 and so on until the gap is 1. At this point it becomes a straight exchange sort but nearly all the items are sorted and only a few exchanges are needed to complete the sort. For greatest efficiency the gap needs to be chosen carefully. The ideal gap is between N and half N (N being the table size) so that when the gap is halved for the next pass it is an odd number. The way to get to this is the biggest power of 2 that will fit into the table minus 1.

Example:= table size	biggest power of 2 to fit	minus 1
100	64	63
200	128	127
400	256	255
1000	512	511

From BASIC this can be set by $GAP\% = 2^{\text{INT}(\text{LOG}(N)/\text{LOG}(2))} - 1$
 No I don't really understand the maths either but IT WORKS.

Now for the program:-

```

10 INPUT "NO OF ENTRIES "; N
20 DIM ENTRY$(N)
30 FOR ENT=1 TO N
40 READ ENTRY$(ENT)
50 NEXT
60 CLS
70 FOR ENT=1 TO N
80 PRINT ENTRY$(ENT); " ";
90 NEXT
100 TI$="000000"
110 PRINT:PRINT "SHELL SORT"; TI$: ENT=N
120 GAP%=2^INT(LOG(ENT)/LOG(2))-1
130 IF GAP%<1 THEN GOTO 300
140 FOR PNTR=1 TO ENT-GAP%
150 FOR PNTR1=PNTR TO 1 STEP -GAP%
160 REM
170 IF ENTRY$(PNTR1)>ENTRY$(PNTR1+GAP%) THEN SWAP
    ENTRY$(PNTR1),ENTRY$(PNTR1+GAP%):ELSE GOTO 280
180 REM
190 REM
200 REM
210 REM
220 REM
230 REM
240 REM
250 REM
260 REM
270 NEXT PNTR1
280 NEXT PNTR
290 GAP%=GAP%/2:GOTO 130
300 T$=TI$
310 PRINT "SORTED "; N; " ITEMS IN "; TI$
320 FOR ENT=1 TO N
330 PRINT ENTRY$(ENT); " ";
340 NEXT
    
```

No to sort	Time sorted list	Unsorted list
10	0	0
20	1	1
40	2	4
80	6	11

As you can see this compares very favourably with the unsorted list times for our previous sorts, but can be regarded as slow for a completely sorted list. I have put in REMs to make the program as long as the previous ones and there is some improvement when these are removed. The tests can be done in exactly the same way as last months tests. Try improving this one for the case of a sorted list?

CALCULATOR

A multi base calculator unashamedly filched by GEO (G3ZQS)
 Will perform addition, subtraction, multiplication and division in any or all of four BASES, being Decimal, Hex, Octal and Split Octal.

```

10 CLS: PRINT@05,10;CHR$(173);MUL$(CHR$(177),29);CHR$(181)
20 PRINTTAB(6);CHR$(181);" ** FOUR-BASE CALCULATOR ** ";CHR$(181)
30 PRINTTAB(6);CHR$(181);"UNASHAMEDLY FILCHED BY G3ZQS ";CHR$(181)
40 PRINTTAB(6);CHR$(189);MUL$(CHR$(161),29);CHR$(181)
50 FOR PAUSE=1 TO 4000:NEXT:CLS
60 PRINT:PRINT "TO USE THE CALCULATOR, PREFIX VALUES IN THE
    EXPRESSION WITH THE APPROPRIATE CODE"
70 PRINT:PRINT "CODES ARE:";TAB(15);"DECIMAL      T"
80 PRINT TAB(15);"HEXADECIMAL  H"
90 PRINT TAB(15);"OCTAL        Q"
100 PRINT TAB(15);"SPLIT OCTAL  S"
110 PRINT:PRINT "EXPRESSION: T1234+2345=H
    
```



```

120 PRINT"EXPRESSION IN DECIMAL: ANSWER IN HEX"
130 PRINT:PRINT"EXPRESSION: H00BA=T
140 PRINT"EXPRESSION IN HEX:ANSWER IN DECIMAL"
150 PRINT:PRINT"BASES MAY ALSO BE MIXED - THUS..."
160 PRINT"EXPRESSION: T1234+HB01E=Q
170 PRINT:PRINT"FUNCTIONS PERMITTED ARE + - * /"
180 PRINT:PRINT"PRESS SPACE TO CONTINUE"
190 X=INCH:IF X=32 THEN CLS:ELSEGOTO 190
200 BCOL14:TCOL6,14:CLS
210 PRINT:PRINT"CODES ARE:";TAB(15);"DECIMAL          T"
220 PRINT TAB(15);"HEXADECIMAL    H"
230 PRINT TAB(15);"OCTAL          Q"
240 PRINT TAB(15);"SPLIT OCTAL    S"
250 PRINT:PRINT
260 B=1:B$="DEC"
270 M=0
280 PRINT@0,10
290 INPUT" EXPRESSION: ";E$
300 IF LEFT$(E$,1)="H"ORLEFT$(E$,1)="Q"OR LEFT$(E$,1)="S"OR
    LEFT$(E$,1)="T"THEN330
310 PRINT:PRINT"YOUR VALUES ARE NOT PREFIXED."
320 PRINT"PLEASE TRY AGAIN !":GOTO 180
330 FOR A=1 TO LEN(E$)
340 X$=MID$(E$,A,1)
350 IF X$="T" THEN B=1:B$="DEC":GOTO 480
360 IF X$="H" THEN B=2:B$="HEX":GOTO 480
370 IF X$="Q" THEN B=3:B$="OCT":GOTO 480
380 IF X$="S" THEN B=4:B$="OCS":GOTO 480
390 IF X$="+" THEN M=1:GOTO 620
400 IF X$="-" THEN M=2:GOTO 620
410 IF X$="*" THEN M=3:GOTO 620
420 IF X$="/" THEN M=4:GOTO 620
430 IF B=4 AND X$=" " THEN 470
440 IF X$="=" THEN V=T:GOTO 620
450 IF (X$<"0" OR X$>"F")GOTO 660
460 IF (X$>"9" AND X$<"A") GOTO 660
470 H$=H$+X$
480 NEXT A
490 ON M GOSUB 580,590,600,610
500 IF E=1 GOTO 660
510 S=T
520 ON B GOSUB 1020,790,940,940
530 IF E=1 GOTO 660
540 PRINT
550 Z=LEN(E$)-2
560 PRINT"YOUR ";LEFT$(E$,Z);" EQUALS ";H$;" ";B$
570 T=S:H$="":GOTO 1030
580 T=V+T:RETURN
590 T=V-T:RETURN
600 T=V*T:RETURN
610 T=V/T:RETURN
620 IF LEN(H$)>0 THEN ON B GOSUB 1010,710,850,850
630 IF E=1 GOTO 660
640 H$="":GOTO 480
650 RUN
660 PRINT:PRINT"SORRY, ";E$;" IS NOT"
670 PRINT"A VALID EXPRESSION"
680 PRINT:PRINT"PRESS SPACE TO RE-START"
690 X=INCH:IF X<>32 THEN 690
700 E=0:CLS:GOTO 210
710 T=0:E=0:X=1
720 FOR I=LEN(H$)TO 1 STEP-1
730 H=ASC(MID$(H$,I,1))-48:IF H>9 THEN H=H-7

```

```

740 IF (H<0 OR H>15) THEN E=1:RETURN
750 T=T+(H*X)
760 X=X*16
770 NEXT I
780 RETURN
790 H$="":X=4096:E=0:IF (T<0 OR T>66535) THEN E=1:RETURN
800 H=INT(T/X):T=T-(H*X)
810 H=H+48:IF H>57 THEN H=H+7
820 H$=H$+CHR$(H)
830 X=X/16:IF X<1 THEN RETURN
840 GOTO 800
850 T=0:E=0:X=1
860 FOR I=LEN(H$) TO 1 STEP-1
870 H=ASC(MID$(H$,I,1))-48
880 IF H=-16 THEN X=256:GOTO 920
890 IF (H<0 OR H>7) THEN E=1:RETURN
900 T=T+(H*X)
910 X=X*8
920 NEXT I
930 RETURN
940 H$="":X=32768:E=0:IF (T<0 OR T>65535) THEN E=1:RETURN
950 IF B=4 THEN X=16384
960 H=INT(T/X):T=T-(H*X)
970 H=H+48
980 H$=H$+CHR$(H)
990 X=X/8:IF X<1 THEN RETURN
1000 GOTO 960
1010 E=0:T=VAL(H$):RETURN
1020 E=0:H$=STR$(T):RETURN
1030 PRINT:PRINT"ANY MORE? "
1040 Q=INCH
1050 IF Q=78 OR Q= 110 THEN BCOL4:TCOL15,4:CLS:END
1060 IF Q=89 OR Q= 121 THEN CLS:GOTO 200
1070 ELSE GOTO 1040

```

EINSTEIN SUBMIT

Introduction

The purpose of SUBMIT is to provide flexible chaining of commands without any operator intervention at runtime. This is mainly useful where the same sequence of commands is used repeatedly or if they have lengthy argument lists.

It is first necessary to write a file containing the commands exactly as they would be entered normally at the DOS prompt from the keyboard. Any editor can be used to produce this command file, which must then be converted to a submit file using one of the public domain submit editors. Some of these editors are capable of generating the submit file directly from the keyboard.

Under CP/M80 the SUBMIT file would be run automatically at this stage, providing it had been placed on the boot drive, and the commands in the file read until it was empty.

Specification

The Einstein SUBMIT is designed to minimise the alterations required by making extensive use of the existing software. It conforms to CP/M SUBMIT protocol (as far as I can ascertain without full documentation) with the following differences:

- (1) The submit file can be placed on any drive.
- (2) The run must be started with a short initialising program.
- (3) A capability to program return codes and a data pointer are available.
- (4) Warm booting occurs normally: it is neither forced nor pre-empted.
- (5) It is compatible with warm-booting from a silicon disc.

Input redirection (that is reading keyboard input requests from disc) is provided in a separate program which is only used when required. This program reduces the available (TPA) memory by 2.5Kb when it is activated.

It should be possible to call any program from a submit file provided that (a) it runs on the Einstein from a normal DOS prompt command and (b) that it does not stop until it is finished. Where input redirection is required, there are some restrictions.

Installation

Reformat a spare disc and try out the changes before copying to other discs.

It is first necessary to alter the instruction sequence in DOS. The changes are specified in SUBPATCH.TXT and can be inserted using MOS R, M and W instructions, or more readily using a disc editor. If you don't understand how to do this spend an evening finding out what these instructions do or ask someone who has already tried patching.

There are several alternative instructions in SUBPATCH.TXT:

- (A) a dummy instruction at 0E200H is required until you have installed INSUB.
- (B) if you wish to warm-boot from a silicon disc use 03H here (I am assuming that you have already organised code to load the operating system on to the silicon disc).
- (C) the ERA command can be made to continue without stopping for confirmation by including this modification. If you do not insert the modification you will either have to type 'Y' at the appropriate stage or make use of INSUB.

Note that the sequence of code starting at 0E163H is relocatable and if you already have code here, it can easily be moved to a higher address by adjusting the two entry point jumps. The altered operating system should now be tested by putting the disc in drive 0 and entering a variety of commands. There should be no discernible difference. If you should need to put another disc in drive 0, press CONTROL-BREAK after putting the altered disc back for the next test.

When you are satisfied, the submit routine in EINSUB.OBJ can be installed. Type:

```
LOAD 0:EINSUB.OBJ
MOS
R AB00 ACFF 0500 0
C 0100 01FF AB00
W AB00 ACFF 0500 0
```

I find it useful to use the T instruction to check that the code has been placed where intended.

INSUB.OBJ is installed in the same way, except that the arguments for R and W are AA00 ABFF 0 0.

NOTES on installation

- (1) Do not alter the operating system in use at the top of memory: loading addresses should be in the range 4000H to 0DFFFH.
- (2) If the alterations in SUBPATCH.TXT are written to drive 1, you can install them in stages. It is probably better to make a backup file first, then LOAD it and write it to disc in one operation.
- (3) The whole operation can be done from MOS (if you can find the files) but there are numerous utilities which assist the procedure: disc editors, linkers, debuggers etc. Their capabilities vary greatly, so use only those with which

you have experience and you will know where they can help.

- (4) I find that ZEN provides most of the required facilities in a simple form, and you can toggle in and out of MOS for reading and writing (ZEN -> MOS: G038H, MOS -> ZEN: Y).
- (5) These instructions, and the object files supplied, are for DOS version 1.31. For DOS version 1.11 you will have to assemble the source files and test them with the conditions altered appropriately. Note that the destination and length in the C instruction above must also be altered.

Instructions

Use your submit editor to generate a submit file from an existing command file or from the keyboard. Make sure that the submit file is on a disc which has EINSUB installed (EINSUB is not strictly necessary on other discs but life will be simpler if you also have EINSUB in drive 0).

Now, if the file has been placed on drive 1, type:

SUBSTART 1:

The drive may be omitted as usual if the file is on the default drive.

SUBSTART is a short program, and if you want an automatic start, it should not be difficult either to add it to the exit routine of your submit editor or possibly to chain it.

If EINSUB stops in the middle of a run (possibly as a result of changing a disc), try continuing by calling SUBSTART. A submit run may be aborted (for example if you have used the wrong file) by pressing SHIFT-BREAK.

Extended facilities

EINSUB allows programs to return (a) a code and (b) a pointer when they finish to provide parameters to a following program. The code must be written to address 0FAFDH. Certain values are reserved. The full range is:

- 0 - EINSUB not active (program running from keyboard)
- 1 - EINSUB activated: initial default value
- 2-127 - available to programs as return codes (not altered or used by EINSUB during the run)
- 128-254 - request for submit to be aborted at end of current program
- 255 - command error

It is the responsibility of the programmer to ensure (1) that any codes set are set back to 1 (or some other available code) when they have been read and (2) that no code is set when the program is run independently from the keyboard. This is done by testing address 0FAFDH: zero indicates keyboard control (no alteration allowed), non-zero indicates SUBMIT control.

The pointer address is 0FAFEH. Under keyboard control it contains 0FFFFH, which is initialised to zero by SUBSTART. A simple sequence to determine the pointer status is:

```
LD  A,(0FAFFH) ; test high byte
      OR  A
      JR  Z,K
      INC A
      JR  Z,I
L:    ; address loaded
K:    ; initialised, no address loaded
I:    ; under keyboard control (but can still be altered)
```


Of course if you need to point to a base page address something more elaborate is needed.

The pointer will normally be used to locate a data area, but it could equally well be used as a chaining address or as a jump vector to select a routine. A short driver program could be called from the command file to read the code and make alterations accordingly, leading to the possibility of constructing conditional command sequences and loops.

Input redirection

It is not possible to read from disc answers to prompts using EINSUB: an optional routine INSUB is provided for this purpose. INSUB diverts CP/M keyboard input requests to the submit file, but version 1 does not divert MOS function input requests. This means that many programs written specifically for the Einstein cannot use INSUB (Note: COPY is a CP/M program and can be used).

A further restriction with INSUB is that it reduces the available TPA memory by 2.5Kb. This should not affect programs directly unless they are a very tight fit, but a number of CP/M programs manipulate memory in a similar way. For example it should be possible to call a debugger during a submit run but INSUB could not be used from within the debugger.

Using INSUB

Your submit editor needs to be able to handle control characters (usually in the form ^M for carriage return, CONTROL-M will normally put you into the next command). If it does not, the only alternative is to put in something readily recognisable such as '%' and patch in the appropriate value. The initial command string must be terminated by the null character ^@. All characters required for input by the program must follow on directly after the null. Do not insert any spaces unless they are expected by the program.

Distinguish carefully between single character input which is read immediately it is typed, and string input which is not read until it is terminated. String input must be terminated by ^M in the command file, which is converted to carriage-return in the submit file (but not written to the input buffer). Do not add a line-feed as it will be read by the next input request. The last input character for the program is followed by a carriage-return (ENTER) which is interpreted as end-of-command and not written to the submit file.

Example:

```
COPY^@1:prog.* TO 3:<V>^MYNY0:*.COM TO 3:<V>^MA^M
```

Note here that if any files are locked and you have not allowed for this the input will get out of phase (use SHIFT-BREAK to start again). Note also the final ^M is required here to exit from the program's '*' prompt. It is simpler to split up the command using the alternative format for COPY:

```
COPY 1:prog.* TO 3:<V>^@YNY
COPY 0:*.com to 3:<V>^@A
```

Function 11 (Console Status) presents some problems. The logical reply during a submit run is zero (no character waiting at keyboard) and this works in most cases. However some programs use the function to wait until function 6 can return a character, so a zero value returned would cause an indefinite wait. Comments please from anyone with programs of this kind.

PATCH FOR INSTALLATION OF EINSUB

Options - see EINSUB.DOC

- +++ (A) Not required when Insub is installed
 £££ (B) Warm-boot from silicon disc
 *** (C) Makes ERA skip request for confirmation

DOS 1.31

DOS 1.11

Original				Modified			
E10C	3A 30 FA			E10C	3A D4 FA		
E10F	FE 04			E10F	D6 3E		
E111	CA 03 E6			E111	32 08 E3		
E114	3E 01			E115	28 4D		
E116	B7			E116	31 FF FC		
E117	D3 23						
E119	D3 24			E119	D3 24		
E11B	C3 E1 00			E11B	C3 DF 00		
-----				-----			
E163	FF			E163	31 1C EB		
..				E166	21 3D EB		
..				E169	71		
..				E16A	AF		
..				E16B	ED 67		
..				E16D	5E		
..				E16E	77		
..				E16F	CD 00 E4		
..				E172	0E 0D		
..				E174	CD 05 00		
..				E177	CD 72 E4		
..				E17A	C3 DC E5		
-----				-----			
E200	FF FF FF	+++		E200	C3 11 EC		
-----				-----			
E300	C3 07 E6			E303	C3 63 E1		
-----				-----			
E603	AF			E603	3A FD FA		
E604	32 08 E3			E606	B7		
E607	31 1C EB			E607	28 1B		
E60A	C5			E609	AF		
E60B	79			E60A	3D		
E60C	1F			E60B	18 17		
E60D	1F			E60D	FF FF		
E60E	1F						
E60F	1F						
E610	E6 0F			E60F	3A FD FA		
E612	5F			E612	B7		
E613	CD 43 E4			E613	CA 7C E4		
E616	0E 0D			E616	F1		
E618	CD 05 00			E617	C3 6C E8		
E61B	C1						
E61C	79						
E61D	E6 0F						
E61F	32 3D EB						
E622	CD B8 E4						
E625	3A 08 E3			E61A	3A 08 E3		
E628	B7			E61D	B7		
E629	20 09			E61E	20 14		
E62B	31 1C EB			E620	3A FD FA		
				E623	B7		
				E624	31 1C EB		
				E627	28 05		
				E629	CD 56 EB		
				E62C	18 09		
-----				-----			
				Original			
				E15F	FF		
				..		E15F	AF
				..		E160	32 08 E3
				..		E163	31 D5 EA
				..		E166	21 F6 EA
				..		E169	71
				..		E16A	AF
				..		E16B	ED 67
				..		E16D	5E
				..		E16E	77
				..		E16F	CD 00 E4
				..		E172	0E 0D
				..		E174	CD 05 00
				..		E177	CD 72 E4
				..		E17A	C3 DC E5

				E200	FF FF FF	+++	E200 C3 11 EC

				E300	C3 C9 E5		E300 C3 63 E1
				E303	C3 C5 E5		E303 C3 5F E1

				E5C5	AF		E5C5 3A FD FA
				E5C6	32 08 E3		E5C8 B7
				E5C9	31 D5 EA		E5C9 28 1B
				E5CC	C5		E5CB AF
				E5CD	79		E5CC 3D
				E5CE	1F		E5CD 18 17
				E5CF	1F		E5CF FF FF
				E5D0	1F		
				E5D1	1F		
				E5D2	E6 0F		E5D1 3A FD FA
				E5D4	5F		E5D4 B7
				E5D5	CD 00 E4		E5D5 CA 39 E4
				E5D8	0E 0D		E5D8 F1
				E5DA	CD 05 00		E5D9 C3 28 E8
				E5DD	C1		
				E5DE	79		
				E5DF	E6 0F		
				E5E1	32 F6 EA		
				E5E4	CD 75 E4		
				E5E7	3A 08 E3		E5DC 3A 08 E3
				E5EA	B7		E5DF B7
				E5EB	20 09		E5D0 20 14
							E5E2 3A FD FA
							E5E5 B7
				E5ED	31 D5 EA		E5E6 31 D5 EA
							E5E9 20 05
							E5EB CD 0F EB
							E5EE 18 09

				E6B8	C3 ED E5		E6B8 C3 C5 E5

				E810	CD B8 E3	+++	E810 CD D1 E5

EAB2 C3 ED E5	EAB2 C3 E2 E5
E6F6 C3 2B E6	E6F6 C3 03 E6

E852 CD 7C E4	*** E852 CD 0F E6

EAF9 C3 2B E6	EAF9 C3 20 E6

FAD2 AF	FAD2 45
FAD3 47	FAD3 4D
FAD4 4F	FAD4 3E 00
FAD5 CF A4	FAD6 CF A4
FAD7 CD 6A FA	FAD8 CD 6A FA
FADA 7E	FADB 7E
FADB FE 04	FADC BD

FAPD 57	FAPD 00
FAPF 09 0A	FAPF FF FF

FAC0 AF	FAC0 45
FAC1 47	FAC1 4D
FAC2 4F	FAC2 3E 00
FAC3 CF A4	FAC4 CF A4
FAC5 21 04 00	FAC6 21 04 00
FAC8 7E	FAC9 7E
FAC9 FE 04	FACA BD

FAPD CF	FAPD 00
FAPF CF CF	FAPF FF FF

MODULA-2

My first program on the Einstein was written in BASIC, and it ran out of memory when it was half completed. My second program was in C and it ran out of disc before it was finished. Even on mainframes, the same problem occurs because of the need to service other users promptly, but it is mitigated by the ability to set up programs consisting of a number of object files, only a few of which are in use at any one stage. Why not have a system like this on the Einstein? I have been searching for software which makes use of this modularity principle, not only on space grounds but also because it simplifies testing and handling of programs. The language which is used is immaterial as long as it provides the features required, but Modula-2 implementations must be contenders because the language is designed to make use of modularity.

The reasons for choosing a modern computer language are concerned more with the use of your programs than with the actual writing. Firstly, maintenance (i.e. making alterations after you have forgotten how the program works) is much easier; secondly portability is greatly improved (fewer changes when your Einstein wears out) and thirdly there should be more scope for maximising the space available for data. Most languages allow for the required facilities: what makes the difference is how the software implements them. I shall therefore concentrate on the way that the implementations work rather than on the language.

Modula-2 is an extension of Pascal and anyone who has used Pascal should have no difficulty in taking up Modula-2. If not, then an introductory textbook is essential. Modula-2 is designed specifically to implement the use of short procedures (about 1 page on average) which can not only be built up into programs but can also be saved in libraries for use in other programs.

Compilers

There are now two Modula-2 compilers available for the Einstein at prices well below the usual CP/M 'only' '200'. The basic requirements are two drives and an intention to write longer programs (an 80-column screen is desirable but not essential initially). These implementations, which have appeared within the last year, are FTL Modula-2 and Borland Turbo Modula-2. FTL provides the source code of most of the system and occupies both sides of three discs. You need two working system discs since the full set of system files occupies more than two sides but you can minimise disc switching by careful grouping. Borland supplies an integrated

package with no system source code which can all be fitted comfortably on one side of a disc. The package includes its own operating system, modeled on the UCSD p-system, but lacking in style. The menu looks like a Wordstar file seen with DISP, presumably to back up Borland's advice to learn to skip the menu. The only redeeming feature of the operating system is a neat I/O redirection mechanism which is in any case part of the language. (Redirection means that when the program is expecting to read from the keyboard or write to the screen, you can make it use a file by means of an additional command line string).

The difference of approach in the two implementations is illustrated by the manuals. Borland provides 500 pages of print as small as you expect to find on the bottom line of a copy protection contract. The FTL manual has 200 pages of print the size of Borland's chapter headings. The cause of this disparity is that Borland gives a detailed specification of each system procedure and variable, while FTL refers you to the source code. As a result you have to keep piles of printout beside your FTL manual. Borland's manual starts with a short introduction, but then becomes formal and throws you in at the deep end with "a digit is an element that can be combined with other digits to form a number that is an element at a different level of abstraction". If you know what this means, stop reading, go and get Borland's package - it's what you have been waiting for. The FTL manual is much more chatty and concentrates on explaining the difficult bits that are glossed over by Borland. In spite of this, FTL provides a clear picture of the system-dependent limitations, while the only limit admitted by Borland is a maximum of 16 elements in a set (which rules out the well-used "IF ch IN {'A'..'Z'}").

Both packages provide screen editors with Wordstar-style commands which return to the corresponding line of the source file when errors are found. The FTL editor is clearly superior and it has a neat menu, but it does not like the Einstein's inverse video toggle - I shall have to put in a Boolean to keep a check for it. The FTL editor also allows you to edit 2 (or 3) files at once (very useful for switching a procedure into another module) and it provides for command repetition and command strings (macros).

FTL follows Wirth's standard for Modula-2 quite closely (at least as far as single user micros are concerned) and has only a few minor extensions. It allows only one library, which does not include .SYM files, and you will quickly build up masses of files requiring very careful organisation. Borland provides for user-defined libraries to hold both types of object file (.MCD and .SYM) which keep your discs much tidier. This is a major improvement: the only other software which does this adequately is UCSD p-system and Borland has produced a more convenient procedure for calling modules from libraries. In addition you can compile either to M-code or to self-contained .COM files or to .COM files with overlays (FTL only produces self-contained .COM files).

Overlays

The last point needs some elaboration. Many earlier compilers have made use of intermediate codes which are run by interpreters. For example BCPL is a language which compiles to O-code and provides a high degree of modularity (although at the expense of omitting many important features). The O-code is very compact (much shorter than machine code), but runs more slowly since the code has to be interpreted. However it is considerably faster than BASIC. Intermediate code is also relocatable (i.e it does not contain any absolute program addresses) and can be loaded and moved anywhere in the available memory. It is therefore the usual method for

implementing dynamic overlays, where parts of the program are read from disc only when needed, and parked anywhere in available memory. Nevada Pascal provides a neat implementation of this, again at the expense of important facilities. UCSD p-system is the only complete implementation of dynamic overlays, with a correspondingly complex calling structure. I am still looking for an implementation of dynamic overlays in machine code. Pascal MT+ provided a blunderbuss implementation of static overlays in machine code, by specifying the overlay address in the linker command tail. Borland has now organised this approach and you only need to specify the name of the module which can be overwritten while the overlay module is in use. This is clearly a rather inflexible arrangement, but overlays are the only way to increase the available data space unless you move on to hardware expansion.

Extensions

Neither package provides much help with debugging. FTL exhorts you to fill the source code with write statements, but does at least have an interface for a 'do-it-yourself' debugger. FTL also provides a number of utility programs such as Gaussian pivoting for solving equations. Borland provides LONGINTEGERS (4 bytes) Pascal-type I/O and introduces exceptions. In standard Modula-2 and in FTL if a system procedure fails for some reason (e.g. no file, string too long) a Boolean variable is usually set and can be tested on return. This means incorporating numerous tests for unlikely (but not impossible) situations into the no-error path. Borland has borrowed the idea of exceptions from Ada (so the answer to your next question is 'yes').

If there is only one error condition it is simple to incorporate statements such as

```
REPEAT
```

```
    ReadInt (i);
```

```
UNTIL Done;
```

and the meaning is intuitive. Anyone who has tried to write a user-friendly program to input assorted information will know that testing for numerous errors at different levels, which have to be returned to different restart points, using the standard method (a number for each error) can easily become both lengthy and complicated, and this slows down the correct response. The use of exceptions not only formalises the return codes by naming each error and providing specific instructions for use at each level but also removes any further tests from the normal program path. An exception is actually only a label in a case statement inserted at the end of a procedure, and a RAISE statement (= GOTO ON ERR) jumps to the exception when the error occurs. The coding of single errors gets a little tedious, but there are 5 error conditions in the FILES module alone, and using exceptions does simplify organising the return codes. You can of course define your own exceptions if you still feel like it. We shall be hearing more of exceptions because Ada, like UCSD p-system is having an influence out of proportion to the number of users.

I have not said a great deal about FTL, because it is a basic system which you can add to or adapt for your own purposes. It is a hacker's program written for hackers: if your favourite disc is full of patches, you will enjoy FTL. In contrast Borland's package is written by professionals for professionals who want to get on with the job without messing about: if your favourite disc is a straight copy of the distribution disc you should choose Borland. Sources: If your local dealer says "Is that a database?", try Grey Matter, Ashburton.

C.P. WALLIS

February 1987.

Scorpio Software announces

***** AMSCOPY *****

Version 2.0

This program enables discs from an Amstrad 6128 or 8256 to be read by the Einstein and automatically copied onto Einstein discs in standard Einstein format. The normal Einstein operating system is used. No additional software or hardware is required. Operation is simplicity itself - simply insert the disc into the default drive and type AMSCOPY. You are asked to specify which drive will be used for the Amstrad disc and which drive will receive the copied files, thereafter the process is totally automatic. The program tests for the recording format of the Amstrad disc. There is no requirement to specify the format of the Amstrad disc. Single disc drive machines can be used but there will be the inevitable disc changing to perform.

It is believed that most CP/M 2.2 software copied in this way will run on the Einstein as well as some CP/M Plus software. Please note that AMSCOPY will NOT cope with Amstrad 'protected' programs.

If the copied software does not have an Install program then it will probably be necessary to run it using the Amstrad Emulator. Indeed you will probably find that the keyboard and screen handling will be superior when using the Emulator. This is certainly the case with Wordstar which was used for testing the Emulator (80 col only).

NOTE: NON STANDARD CP/M PROGRAMS COPIED FROM AN AMSTRAD ARE MOST UNLIKELY TO RUN ON THE EINSTEIN (eg games).

A wide range of Amstrad CP/M software has been copied and run successfully on the Einstein.

***** AMRUN *****

THE AMSTRAD EMULATOR PROGRAM (80 col only)

Version 2.0 This is the high speed version. The Einstein screen handling has been re-written and operates in RAM. This in itself gives an increase in speed but the code is for 80 column only which also gives a considerable gain in speed. This version also includes a keyboard buffer which eliminates any possibility of lost characters when typing at speed.

AMSCOPY is used to copy CP/M programs from Amstrad 6128 and 8256 discs. This program enables the copied Amstrad programs to run on the Einstein without making any alterations of any kind.

TO USE:

Type:- AMRUN FILE1.COM FILE2.TYP

where FILE1.COM is the copied Amstrad program and FILE2.TYP is the name of a second file IF this is required.

E.g. AMRUN WORDPRO.COM EDITFLE.DOC

If there is no requirement for a second file then type:-

AMRUN PROGRAM.COM

No further action is required to use AMRUN.

(N.B most CP/M 2.2 programs and some CP/M plus programs copied from Amstrad 6128 discs, in either Data or System format, will run on the Einstein using AMRUN)

The disc containing AMSCOPY2 and AMRUN costs £24.95 total, from

B & H Ltd.,
Bank Top Works,
Southowram,
Halifax,
Yorkshire.
Tel (0422) 52905

B & H Ltd. give a discount of 10% on Amscopy and on certain other items of hardware and software to UKEUG members.

BACKPAGE INFORMATION

As you can see from the front page the running of the group has now changed hands. Mike Smallman and myself, Graham Bettany, are now in the hotseats. It was intended that the first magazine produced by us would be the June issue but due to many and varied reasons the U.K.E.U.G. newsletter was somewhat behind time and we will start from July. I may live to regret this but we will not be late!!

We shall be calling the newsletter Einstein Monthly as of July, and as we have just enough members now to use a professional printers you should have no problems with the print quality. We do need more members for the group so do tell other users of our existence if you can. The monthly competitions will start next month, first prize will be a Tatung 4 band radio and there will be some runner up prizes. Screens, B+H, and Mike bayliss have agreed to sponsor the competitions and also take part in a National Exhibition later this year, similar to the one held last November. If you have any ideas on how the competition should be run please send them in, it is no mean task to set a competition that is interesting.

We are at present sorting the Public Domain software and will have a complete list in the first issue of Einstein Monthly. If you have any programs that we could use to expand the library then please send them in. We want programs written in any language, I'm sure you have composed something. Einstein Monthly will have a section for ONELINERS so please send any online programs you have written, the best entry will receive a disc containing most of the U.K.E.U.G. programs that have been published in the past issues.

A disc copy of each magazine's programs is available for 5 pounds if we supply the disc, or 2 pounds if you send a disc. Backcopies of the newsletter can be had for 1 pound 25 pence, including postage.

I'm sure you're bored of being told but we do need your input to produce a worthwhile magazine, so do send in anything that could help, articles, questions, comments, wants, swaps etc,etc

Membership enquiries :-

Letters, articles, comments:-

U.K.E.U.G.
GRAHAM BETTANY
80 DALES ROAD
IPSWICH
SUFFOLK
IP1 4JR

U.K.E.U.G.
MIKE SMALLMAN
7 SHEEPEN PLACE
COLCHESTER
ESSEX
CO3 3LD

Tel: 0473 49507

Tel: 0206 540540

IMPORTANT: IF YOU REQUIRE A PERSONAL REPLY YOU MUST ENCLOSE A S.A.E.