9 Trapping Other Errors

Chapters 7 and 8 described how two of the errors generated by BASIC could be trapped, and used to add new commands, or to overlay procedures and functions. This section gives a couple of examples of recovering from other errors.

9.1 Bad MODE recover

If an attempt is made to change mode inside a PROC or a FN, a ‘Bad MODE’ error (error number 25) is generated. When a PROC or FN is in operation, there will be data on the BASIC STACK, which it will use when it returns (see section 5.3).

A MODE change alters HIMEM and resets the BASIC STACK pointer to this new value of HIMEM. If this was reset inside a PROC or a FN, the BASIC STACK contents would be lost, and BASIC would crash when the call returned.

However, by trapping this error, changing MODE inside a PROC or a FN can be allowed, providing that the bottom of the new MODE is above the current HIMEM. If it is, HIMEM can be left as it is, and the BASIC STACK pointer left unchanged. For example, changing from MODE 3 to MODE 6 would be allowed, as the bottom of screen is higher for MODE 6 than MODE 3.

The prevailing conditions on a ‘Bad MODE’ error are:

Stack contents:	RTI information	3 bytes
	&16 MODE change char.	1 byte

PTRA	points at statement delimiter
&2A	prospective MODE number

If it is possible to change MODE without moving the STACK, this routine will print the MODE change command and continue executing the program. It will not reset HIMEM or the STACK, although the normal MODE change routine will continue to do so whenever the MODE change is made outside a FN or PROC. This means that after this routine has been called, there may be a gap between HIMEM and the bottom of the screen.

```
  10 REM *** Program to allou MODE change inside PROCS ***
  12 REM
  14 REM        M D Plumbley         1984
  16 REM
  18 REM This program traps the "Bad MODE" error (ERR = 25)
  20 REM
  22 REM If there is enough room to change MODE above
  24 REM  HIMEM, tfithout disurbing the BASIC stack, then
  26 REM  MODE can be changed, even if the stack is in use
  28 REM  (i.e. there is a FN or PROC active at the time)
  30 REM
  32 REM "Bad MODE" will still be given if you are changing
  34 REM  to a mode which requires HIMEM to be lower than
  36 REM  the current setting (untess you are not in a
  38 REM  FN/PROC).
  40 REM
  42 REM For BASIC 1, replace EQUs as in chapter 7.
  44 REM
  99
 100 PROCsetup :REM Set up correct ROM entry points
 490
 495 REM *** OS routines and vectors ***
 500 OSWRCH = &FFEE
 505 OSBYTE = &FFF4
 550 BRKV   = &0202
 590
 595 REM *** Allocate workspace ***
 600 worksp = &0070
 605 svbrkv = worksp
 690
 695 REM *** BASIC system variabtes ***
 700 Lomem = &0D00
 705 Heap  = &0002
 710 Stack = &0004
 715 Himem = &0006
 720 Top   = &0012
 725 Count = &001E
 799
 900 start% = &0C00 :REM Assemble into user char space
 905
 910 FOR opt% = 0 TO 3 STEP 3
 920 P% = start%
 950 [OPT opt%
1000 .init
1005     LDA &8015            \Test that the correct
1010     CMP #baschr          \ version of BASIC is
1015     BEQ basok            \ in the ROM.
1016
1020     BRK                  \If it isn't, print an
1025     EQUB 60              \ error message.
1030     EQUS "Not BASIC "    \ (baschr set by PR0Csetup)
1035     EQUB baschr
```

```
1040     EQUB 0
1041
1045 .basok
1050     LDA BRKV             \Load the current BRK vector
1055     LDX BRKV+1           \ into A and X.
1056
1060     CMP #newbrk MOD &100 \If this routine is already
1065     BNE ntsavd           \ set up, don't change BRKV.
1070     CPX #newbrk DIV &100
1075     BEQ saved
1076
1078 .ntsavd
1080     STA svbrkv           \It has not been set up
1085     STX svbrkv+1         \ atready, so save old
1090     LDA #newbrk MOD &100 \ BRKV, and set up the new
1095     STA BRKV             \ one.
1100     LDA #newbrk DIV &100
1105     STA BRKV+1
1106
1110 .saved
1115     RTS
1190
1192    \ *** This is the new BRK handting routine  ***
1200 .newbrk
1205     PHA                 \Save A and Y on 6502 stack
1210     TYA
1215     PHA
1216
1220     LDY #0              \Get error number
1225     LDA (&FD),Y
1226
1230     CMP #25             \If ERR = 25 ("Bad MODE"), then
1235     BEQ badmde          \ try to correct it
1236
1240 .giveup
1245     PLA                 \Restore A any Y from 6502 stack
1250     TAY
1255     PLA
1256
1260     JMP (svbrkv)        \Go to old BRK handter
1261
1490 \ *** If we get here, a "Bad MODE" error has      ***
1492 \ ***  occurred. This was either caused by a      ***
1494 \ ***  non-empty BASIC stack, or not enough room. ***
1500 .badmde
1505     LDX &2A             \Get requested mode number from
1510     LDA #&85            \ IntA, and find out what HIMEM
1515     JSR OSBYTE          \ would be in that mode.
1516
1520     CPX Himem           \If new HIMEM woutd be below the
1525     TYA                 \ current HIMEM, then the STACK
1530     SBC Himem+1         \ is in the way.
```

```
 1535     BCC giveup
 1536
 1540     CPX Heap            \If new HIMEM would be below the
top
 1545     TYA                 \ of the variables heap, there is
 1550     SBC Heap+1          \ not enough room for the MODE.
 1555     BCC giveup
 1556
 1560     CPX Top             \If HIMEM woutd be below TOP,
there
 1565     TYA                 \ is not enough room for the MODE.
 1570     SBC Top+1           \ This test is in case LOMEM had
 1575     BCC giveup          \ not been set to TOP yet.
 1576
 1580     PLA                 \Discard saved vatues of Y and A
 1590     PLA                 \ from 6502 stack
 1591
 1600     PLA                 \Discard RTI information from the
 1605     PLA                 \ 6502 stack. This is pushed by
 1610     PLA                 \ the BRK instruction.
 1611
 1615     LDA #0              \Zero COUNT (a MODE change leaves
 1620     STA Count           \ the cursor at start of line)
 1621
 1625     PLA                 \Pop "mode change" byte from stack
 1630     JSR OSWRCH          \ (pushed by MODE command), and
 1631                         \ print it
 1632
 1635     LDA &2A             \Get mode number from int ace, and
 1640     JSR OSWRCH          \ print that
 1641
 1645     JMP cont            \Command compteted, so execute the
 1646                         \ next statement.
 1647
 8000 ]
 8010 NEXT
 8015 @%=0
 8020 PRINT'"Code length =&"~P%-start%
 8190
 8200 PRINT'''''"** WARNING: Once assembled, the code"
 8210 PRINT"generated by this program is not"
 8220 PRINT"transferable between different BASICs"
 8230 PRINT
 8300 PRINT"Execute ""CALL &"~init""" to initiatise."'
 8310 END
 8990
 8992 REM *** Set up R0M entry points, allowing for ***
 8993 REM *** BASIC I and BASIC II. ***
 9000 DEFPROCsetup
 9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
 9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
 9030 IF $&8009=basic1$ THEN PROCset1 :ENDPROC
 9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
 9050 PRINT "NOT BASIC 1 OR 2
 9060 END
```

```
9290
9292 REM *** Set up BASIC 1 entry points ***
9300 DEFPROCset1
9305 baschr = ASC"1":REM Used by init routine
9310 cont   = &8B0C :REM Cont execution at next statement
9320 ENDPROC
9490
9492 REM *** Set up BASIC 2 entry points ***
9500 DEFPROCset2
9505 baschr = ASC"2":REM Used by init routine
9540 cont   = &8B9B :REM Cont execution at next statement
9550 ENDPROC
```

The initialising and BRK handling parts of this routine are very similar to the programs in chapter 7. In fact, there is not really a lot to the program at all.

This routine could be modified to copy the BASIC stack bodily if a MODE change was made which required HIMEM to be lower than its current setting. This could also be used anyway, to ensure that the least amount of memory was being used for each MODE.

Performing a MODE change, and shifting the stack, may be one way of allocating more memory if a ‘No room’ error is generated. However, this is only possible with BASIC 2, as this error does not use the BRK error generating mechanism in BASIC 1 (see chapter 11 for more on ‘No room’)

9.2 Bad program salvage

One of the more annoying error messages that BASIC can produce is ‘Bad program’. You may have just waited 10 minutes for a long program to load from tape, or spent the last 2 hours typing something in, to be greeted by this message because the program got corrupted somehow. This section describes how the bad program, or as much of it as possible, can be salvaged into an editable form.

Program storage

Program lines are stored in the following format:

00	MSB of line number
01	LSB of line number
02	total length of line (= XX)
03	first character of line text
04	etc.
XX−1	&0D (carriage return) line end marker
XX	MSB of line number of next line
XX+1	etc.

The first byte stored at PAGE is a &0D (carriage return), followed by the MSB of the first line number. The end of the program is marked by an &FF byte after the carriage return on the end of the last line.

The length byte of the line number is used to speed up the search for line numbers in a GOTO or GOSUB. However, if one of these gets corrupted, so that there isn't a &0D where BASIC thinks the end of the line should be, it will give a ‘Bad program’ error. This could also be caused if the carriage return has been corrupted.

By scanning through the program, re-linking all these length bytes, the program can be savlaged. It may not be completely correct, but at least it will be possible to edit it again.

The salvage routine

This routine can be assembled and the code saved onto disc or cassette by using ‘*SAVE’. It assembles into the user defined character area, so the code can be loaded in and executed if a ‘Bad program’ occurs, without disturbing the program to be salvaged.

The program can be loaded and run by typing

```
*LOAD SALVAGE
CALL &C00
```

assuming that it was assembled from &C00 onwards. If the DFS, or any filing system which operates from a paged ROM, is used to load the routine, it should not be run by using ‘*SALVAGE’. If this was used, the DFS ROM, rather than the BASIC ROM, would be paged in while the routine was operating, and the BASIC ROM routines which the are called would not be available. To get round this, the ROM routines required could be duplicated in the salvage routine itself.

```
   4 REM **       Bad program salvage routine       ***
   6 REM
   8 REM             M D Plumbley 1984
  10 REM
  12 REM This routine will scan through the BASIC program
  14 REM  at PAGE and re-set any link pointers which have
  16 REM  been corrupted.
  18 REM
  20 REM Before using with BASIC 1, the EQUs shoutd be
  22 REM  replaced with their equivalents:
  24 REM  "EQUB X"   => "]?P%=X:P%=P%+1:[OPTopt%"
  26 REM  "EQUS A$"  => "]$P%=A$:P%=P%+LEN$P%:[OPTopt%"
  90 REM
  99
 100 PROCsetup :REM Set up correct ROM entry points
 490
 495 REM *** OS routines and vectors ***
 510 osrdch = &FFE0
 590
 600 worksp = &0070
 605 line   = worksp
 610 ytemp  = worksp+2
 690
 695 REM *** BASIC system variables ***
 700 page   = &0018
 710 inta   = &002A
 799
 900 start% = &0C00 :REM User defined character area
 905
 910 FOR opt% = 0 TO 3 STEP 3
 920 P% = start%
 950 [OPT opt%
 990
 995\ ** Salvage routine entry point ***
1000 .slvage
1005     LDA page           \Set "line" to point to the
1010     STA line+1         \ first byte of the program
1015     LDY #0             \ at PAGE.
1020     STY line
1025
1030     LDA (line),Y       \If it is a CR, jump to start
```

```
1035     CMP #&0D            \ checking through the lines.
1040     BEQ strtok
1045
1050     JSR pmess                   \Othertfise, print an
1055     EQUS "No CR at start"       \ error message and
1060     NOP                         \ exit.
1065 .end
1070     RTS
1075
1100 .escape                 \This is used to give an
1105     BRK                 \ "Escape" error if the
1110     EQUB 17             \ necessary
1115     EQUS "Escape"
1120     EQUB 0
1125
1195 \ ** Start looking through lines ***
1200 .strtok
1205     JSR pnewl           \Start on a new line
1210
1215     BIT &FF             \If an escape condition is
1220     BMI escape          \ pending, handle it.
1225
1230     LDA line+1          \Print out the address of the
1235     JSR phex            \ current line.
1240     LDA line
1245     JSR phexsp
1250
1255     LDY #1              \If we are at the end of the
1260     LDA (line),Y        \ program, exit.
1265     BMI end
1270
1275     STA inta+1          \Otherwise, print out the
1280     INY                 \ line number.
1285     LDA (line),Y
1290     STA inta
1295     JSR plnum5
1300
1305     LDY #3              \Get the length byte from the
1310     LDA (line),Y        \ line. If it is zero, the
1315     BEQ flink           \ link has failed, so fix it.
1320
1325     TAY                 \Get the byte on the end of
1330     LDA (line),Y        \ the line.
1335
1340     CMP #&0D            \If it is not a CR, the link
1345     BNE flink           \ failed, so fix it.
1350
1355     TYA                 \Transfer the length into A
1360
1365 .newlna
1370     CLC                 \Add the length of the line
1375     ADC line            \ (in A) to the line pointer,
1380     STA line            \ so it now points to the
1385     BCC strtok          \ line, and go back to
1390     INC line+1          \ "strtok" to handl-e the next
1395     BCS strtok          \ line.
1400
1990 \ ** If we get here, the link has faited ***
2000 . flink
2005     JSR pmess           \Print a message
2010     EQUS " Failed link"
2015     NOP
2020
2025     LDY #3              \Scan from the start..
2030
2035 .cscan                  \ for control characters
2040     LDA #&1F            \ (i.e. less than &20)
2045     INY
2050
2055 .loop                   \loop round until a control
2060     CMP (line),Y        \ character is found. If it
2065     BCS fixlnk          \ is, go to fix the link.
2070     INY
2075     BNE loop
2080
2085     DEY                 \If the end wasn't found, set
2090     STY ytemp           \ the "end" to be used at 255
2095
2100     JSR pmess                   \ and print the
2105     EQUS " End not found: F/T"  \ message.
2110     NOP
2115
2120     JSR osrdch          \Read a character, and exit
2125     BCS escape          \ if ESC was pressed.
2130
2135 .notasc                 \Check for a "T".
2140     CMP #ASC"T"
2145     BNE noterm
2150
2155     LDA #&FF            \If it was, set the MSB of
2160     LDY #1              \ the current line to &FF
2165     STA (line),Y        \ to terminate the program,
2170 .nforce                 \ and exit.
2175     RTS
2180
2200 .noterm                 \If it wasn't, check for an
2205     CMP #ASC"F"         \ "F".
2210     BNE nforce
2215
2220     LDY ytemp           \If it uas, set the character
2225 .force                  \ where scanning stopped to
2230     LDA #&0D            \ be a CR, and ...
2235     STA (line),Y
2240
```

```
2245     TYA                 \ set the length byte,
2250     LDY #3              \ and ...
2255     STA (line) ,Y
2260
2265     JMP newlna          \ go to the next line.
2270
3000 .fixlnk                 \If the controt character
3005     LDA (line),Y        \ that was found was a CR,
3010     CMP #&0D            \ force the length byte to
3015     BEQ force           \ point to it.
3020
3025     STY ytemp           \Otherwise, save the offset,
3030
3035     JSR pmess                   \ and print the
3040     EQUS " Control. char A/F/T" \ message.
3045     NOP
3050
3055     JSR osrdch          \Read the character input,
3060     BCS jesc            \ and exit if ESC pressed.
3065
3070     CMP #ASC" A"        \Check for "A" .
3075     BNE notasc
3080
3085     LDY ytemp           \If it was, force the
3090     LDA (line),Y        \ control char to be a letter
3095     ORA #&40            \ by ORing it with &40, and
3100     STA (line),Y        \ jump back to continue
3105     JMP cscan           \ scanning the line.
3110
3200 .jesc                   \Jump the the "Escape" error.
3205     JMP escape
8000 ]
8010 NEXT
8015 @%=0
8020 PRINT'"Code length =&"~P%-start%
8190
8200 PRINT'''''"** WARNING: Once assembled, the code"
8210 PRINT"generated by this program is not"
8220 PRINT"transferable between different BASICs"
8230 PRINT
8300 PRINT"Execute ""CALL &"~start%"""  to use"'
8310 END
8990
8992 REM *** Set up ROM entry points, allwing for ***
8993 REM ***  BASIC 1 and BASIC 2.                ***
9000 DEFPROCsetup
9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
9030 IF $&8009=basic1$ THEN PROCset1 :ENDPROC
9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
9050 PRINT "NOT BASIC 1 OR 2"
9060 END
```

```
9290
9292 REM *** Set up BASIC 1 entry points ***
9300 DEFPROCset1
9305 plnum5 = &98F5 :REM Print line number (field 5)
9310 pmess  = &BFCB :REM Print message fottenfing JSR
9315 pnewl  = &BC42 :REM Print a new line (CRLF)
9320 phex   = &8570 :REM Print A as 2-digit HEX no.
9325 phexsp = &856A :REM Print HEX no. then space
9330 ENDPROC
9490
9492 REM *** Set up BASIC 2 entry points ***
9500 DEFPROCset2
9505 plnum5 = &9923 :REM Print line number (fietd 5)
9510 pmess  = &BFCF :REM Print message following JSR
9515 pnewl  = &BC25 :REM Print a new line (CRLF)
9520 phex   = &B545 :REM Print A as 2ASNdigit HEX no.
9525 phexsp = &B562 :REM Print HEX no. then space
9600 ENDPROC
```

The general operation of the routine is as follows:

1 It first checks that there is a carriage return at the start of the program. If there isn't, it prints a message and exits. If this happens, either there was no BASIC program at all, or the routine can be re-started after '?P AGE=13' has been typed.

2 The start address of the current line, and its line number, are printed. If the program is so bad that this savlage routine cannot cope with it properly, this infornation may help if a hex dump program needs to be used to patch up the program.

3 If the end of the program has been found, the routine exits.

4 If the length byte points correctly to the carriage return on the end of the line, the routine moves on to the next line, and jumps back to stage 2.

5 The message 'Failed link' is printed after the line number, and the line is scanned until a control character is found.

6 If the control character found was a carriage return, the length byte is fixed, and the routine jumps back to continue checking the rest of the program.

7 If the end of the line was not found, or the control character found was not a carriage return, the routine gives the option of forcing the control charater to be a letter, forcing the end of the line to be at this point, or marking the end of the program at this line.

The ESC key can be pressed at any time while the salvage operation is underway, and the routine will stop when it is about to do the next line.

The routine may think that it has reached the end of the program before it should have, because it found a negative byte as the MSB of the next line number. It can be forced to continue by typing ‘END:?(TOP−1)=0’ to force the end marker to zero before re-starting the salvage routine.

This routine will cope with most things, but if the program is really bad, the following hex dump program may be useful to examine it by hand. It should be loaded in by setting PAGE above the top of the corrupted program (give plenty of room, just in case), and then just LOADing in as normal.

```
   5 REM **       Hex dump program           **
   6 REM
  10 REM        M D Plumbley    1984
  15 REM
  20 REM Press <space>   to stop listing
  25 REM       <return>  to continue
  30 REM         "Q"     to quit
  35 REM
 100 len% = 8 :REM length of line (bytes)
 200 INPUT"START ADDR :&"input$
 210 start% = EVAL("&"+input$)
 220 INPUT"END   ADDR :&"input$
 230 end% = EVAL("&"+input$)
 400 REPEAT
 410   PROCline(start%)          :REM Hexdump 1 line
 420   start% = start%+len%      :REM Next line
 430   key$ = INKEY$(0)
 440   IF key$=" " THEN PROCuait
 450   IF key$="Q" THEN END
 460   UNTIL start%>end%
 470 END
 998
 999 REM *** Print hexdump of 1 line ***
1000 DEFPROCline(addr%)
1010 @%=4:PRINT~addr%" ";        :REM Addr at start of line
```

```
1015 @%=3
1017 text$ = ""                 :REM Clear text string
1020 FOR offset = 0 TO len%-1
1030 byte% = addr%?offset        :REM Get byte
1040 PRINT ~byte%;               :REM Print hex byte
1045 valid = (byte%>=&20 AND byte%<&7F)
1046         :REM Is it a character?
1050 IF valid THEN chr$=CHR$(byte%) ELSE chr$="."
1060 text$ = text$+chr$          :REM Add char to text string
1070 NEXT offset
1080 PRINT" " text$
1090 ENDPROC
1998
1999 REM *** Wait for <CR> or "Q" to be pressed ***
2000 DEFPROCwait
2010 REPEAT
2020   key$ = GET$
2030   UNTIL key$=CHR$(13) OR key$="Q"
2040 IF key$="Q" THEN END
2050 ENDPROC
```

9.3 Error listing

Sometimes it is not very easy to spot an error in a line of BASIC, especially when it is in the middle of a multi-statement line. The routine in this section will LIST out the line that any error occurred on, together with 2 markers pointing out the possible sources of the error. These represent the positions of the two BASIC text pointers, PTRA and PTRB, at the instant of the error.

For example, if the following line is typed in:

```
>PRINT"HELLO"; REM ShouLd be a ":"
```

the response will be:

```
HELLO
PRINT"HELLO"; REM Should be a ":"
                 ^
                 ^
No such variable
```

The top arrow represents the position of PTRA, and the bottom one represents the position of PTRB. In this case, they both point to the same position (just after the REM token), but in most cases they will be different.

This can also be used to check the position of the pointers, if certain errors are to be intercepted.

```
   5 REM ***        Error Listing routine           ***
   7 REM
  10 REM           M D Plumbley 1984
  15 REM
  20 REM When an error occurs, this routine will print out
  25 REM  the offending line, and print the position of
  30 REM  the ttw BASIC pointers, pointing out the error.
  35 REM
  40 REM This program assembles into user keylcharacter
  42 REM  area at &0B00 ornrards.
  44 REM
  46 REM Before using with BASIC 1, the EQUS shoutd be
  48 REM  replaced with their equivatents:
  50 REM  "EQUB X"  => "]?P%=X:P%=P%+1:[OPTopt%"
  52 REM  "EQUW X"  => "]!P%=X:P%=P%+2:[OPTopt%"
  54 REM  "EQUS A$" => "]$P%=A$:P%=P%+LEN$P%:[OPTopt%"
  56 REM
  99
 100 PROCsetup :REM Set up correct R0M entry points
 490
 550 BRKV = &0202
 799
 900 start% = &0B00 :REM User key/char space
 905
 910 FOR opt% = 0 TO 3 STEP 3
 920 P% = start%
 950 [OPT opt%
1000 .init
1005     LDA &8015           \Test that the correct
1010     CMP #baschr         \ version of BASIC is
1015     BEQ basok           \ in the ROM.
1016
1020     BRK                 \If it isn't, print an
1025     EQUB 60             \ error message.
1030     EQUS "Not BASIC "   \ (baschr set by PROCsetup)
1035     EQUB baschr
1040     EQUB 0
1041
1045 .basok
1050     LDA BRKV              \Load the current BRK vector
1055     LDX BRKV+1            \ into A and X.
1056
1060     CMP #newbrk MOD &100  \If this routine is already
1065     BNE ntsavd            \ set up, don't change BRKV.
1070     CPX #newbrk DIV &100
1075     BEQ saved
1076
1078 .ntsavd
```

```
1080     STA svbrkv         \It has not been set up
1085     STX svbrkv+1       \ atready, so save old
1090     LDA #newbrk MOD &100\ BRKV, and set up the new
1095     STA BRKV           \ one.
1100     LDA #newbrk DIV &100
1105     STA BRKV+1
1106
1110 .saved
1115     RTS
1480
1490 \ *** Enter here on BRK ***
1500 .newbrk
1502     PHA                \Save A,Y,X on 6502 stack
1504     TYA
1506     PHA
1508     TXA
1510     PHA
1511
1515     JSR pnewl          \Start a new line
1516
1520     LDA #&FF           \Set up immediate area
1525     STA &3D            \ as defautt for error area.
1530     LDA #&06           \ (83D) is used to point to the
1540     STA &3E            \ start of the line in error
1545
1550     LDA &C             \If error occurred in immed mode,
1560     CMP #7             \ don't look for a line
1570     BEQ immed
1575
2010     JSR setERL         \Get ERL, and
2020     LDA &8             \ copy it into the
2030     STA &2A            \ integer accumulator
2040     LDA &9             \ ready for "schlin"
2050     STA &2B
2055
2060     JSR schlin         \Point (&3D) at start of line
2070     BCS noline         \Exit if line not found
2072
2075     JSR pnewl          \Start a new line, followed by
2080     JSR plnum5         \ the line number
2082
2085 .immed
2090     LDA #0             \Reset counters for
2100     STA countA         \ the position of the pointers
2110     STA countB         \ on the line
2115
2120     LDA &A             \Save PTRA in temp area
2130     STA ptrtmp
2140     LDA &B
2150     STA ptrtmp+1
2160     LDA &C
2170     STA ptrtmp+2
```

```
2175
2180     LDA &3D            \Set PTRA to point to start
2190     STA &B             \ of line in error.
2200     LDA &3E            \ (PTRA is used by the line number
2210     STA &C             \ decoding routine)
2220     LDY #1
2230     STY &A
2235
2240     JSR prtlne         \Print out line, setting counters
2245
2250     LDX countA         \Print posn of PTRA
2260     JSR prtptr
2262     JSR pnewl
2265
2270     LDX countB         \Print posn of PTRB
2280     JSR prtptr
2285
2290     LDA ptrtmp         \Restore PTRA from temp area
2300     STA &A
2310     LDA ptrtmp+1
2320     STA &B
2330     LDA ptrtmp+2
2340     STA &C
2342
2345 .noline
2350     PLA                \Restore X,Y,A from 6502 stack
2355     TAX
2360     PLA
2365     TAY
2370     PLA
2371
2375     JMP (svbrkv)       \Continue with defautt BRK routine
2376
2900 .exit
2910     JMP pnewl          \Print CRLF at end of line
2920
2990 \ *** Print out line at PTRA, setting counters    ***
2991 \ ***  countA and countB to the screen positions  ***
2992 \ ***  of the saved PTRA and PTRB                 ***
3000 .prtlne
3010     LDY &A             \Get next character, and
3020     INC &A             \ increment PTRA
3030     LDA (&B),Y
3035
3040     CMP #&0D           \If end of line,
3050     BEQ exit           \ print CRLF and exit.
3055
3060     CMP #&8D           \If a line number,
3070     BEQ lineno         \ print it
3075
3080     JSR ptoken         \Print char or token in A
3090     JMP counts         \ and skip line number section
```

```
3095
3100 .lineno
3110     JSR getlno         \Get line number after token
3120     JSR plnum0         \ and print it
3130 .counts
3140     CLC                \Move PTRA (position of next
3150     LDA &A             \ char to be printed) into
3160     ADC &B             \ integer accumulator
3170     STA &2A            \ at &2A and &2B
3180     LDA &C
3190     ADC #0
3200     STA &2B
3205
3210     LDA ptrtmp         \Get old PTRA from temp area
3220     ADC ptrtmp+1       \ into X (LSB)
3230     TAX                \
3240     LDA ptrtmp+2       \ and  A (MSB)
3250     ADC #0
3255
3260     CPX &2A            \If char at old PTRA has not
3270     SBC &2B            \ been printed yet,
3280     BCC nocntA         \
3290     LDA &1E            \ set countA to COUNT
3300     STA countA         \ (COUNT held in &1E)
3305
3310 .nocntA
3320     CLC                \Get PTRB
3330     LDA &1B            \
3340     ADC &19            \ into X (LSB)
3350     TAX                \
3360     LDA &1A            \ and  A (MSB)
3370     ADC #0
3375
3380     CPX &2A            \If char at PTRB has not been
3390     SBC &2B            \ printed yet,
3400     BCC nocntB         \
3410     LDA &1E            \ set countB to COUNT
3420     STA countB
3425
3430 .nocntB
3440     JMP prtlne         \Go back for another char
4990
4991
4992 \ *** Print a "^" in the Xth cotumn ***
4993 \ ***  (entry point is "prtptr")    ***
5006 . loop
5010     LDA #ASC(" ")      \Print a space
5020     JSR pchar
5022
5025 .prtptr
5030     CPX &1E            \If not at the right cot,
5040     BNE loop           \ print another space.
```

```
5045
5050     LDA #ASC("^")      \Print a "^"
5060     JSR pchar
5065
5080     RTS                \Exit
7790
7792                        \ *** Routine variabtes area ***
7800 .svbrkv EQUW !BRKV     \Space to save BRK vector
7801
7810 .countA EQUB 0         \Screen posn of PTRA
7815 .countB EQUB 0         \Screen posn of PTRB
7816
7820 .ptrtmp EQUW 0         \Temp for PTRA
7825     EQUB 0
8000 ]
8010 NEXT
8015 @%=0
8020 PRINT'"Code length =&"~P%-start%
8190
8200 PRINT'''''"** WARNING: Once assembled, the code"
8210 PRINT"generated by this program is not"
8220 PRINT"transferable between different BASICs"
8230 PRINT
8300 PRINT"Execute ""CALL &"~init""" to initialise."'
8310 END
8990
8992 REM *** Set up ROM entry points, allowing for ***
8993 REM ***  BASIC 1 and BASIC 2.                 ***
9000 DEFPROCsetup
9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
9030 IF $&8009=basic1$ THEN PR0Cset1 :ENDPROC
9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
9050 PRINT "NOT BASIC 1 OR 2"
9060 END
9290
9292 REM *** Set up BASIC 1 entry points ***
9300 DEFPROCset1
9305 baschr = ASC"1":REM Used by init routine
9310 setERL = &B3F6 :REM Get no of line in error into &8,9
9315 schlin = &9942 :REM Find start of line given line no
9320 plnum5 = &98F5 :REM Print &2A,2B in decimal (field 5)
9325 plnum0 = &98F1 :REM Print &2A,2B in decimal (field 0)
9330 ptoken = &B53A :REM Print char, or token if A > &7F
9335 pchar  = &B571 :REM Print char in A, and incr COUNT
9340 pnewl  = &BC42 :REM Print CRLF, and zero COUNT
9345 getlno = &97BA :REM Get tokenised line no at PTRA
9350 ENDPROC
9490
9492 REM *** Set up BASIC 2 entry points ***
9500 DEFPROCset2
9505 baschr = ASC"2":REM Used by init routine
```

```
9510 setERL = &B3C5 :REM Get no of line in error into &8,9
9515 schlin = &9970 :REM Find start of line given line no
9520 plnum5 = &9923 :REM Print &2A,2B in decimal (field 5)
9525 plnum0 = &991F :REM Print &2A,2B in decimal (field 0)
9530 ptoken = &B50E :REM Print char, or token if A > &7F
9535 pchar  = &B558 :REM Print char in A, and incr COUNT
9540 pnewl  = &BC25 :REM Print CRLF, and zero COUNT
9545 getlno = &97EB :REM Get tokenised line no at PTRA
9550 ENDPROC
```

The general operation of the routine is as follows:

1 The pointer at &3D ,&3E is set up to point to the start of the line in error, by searching through the program if necessary.

2 The line is printed out, updating counters which mark the screen position of PTRA and PTRB. Tokens are expanded by the ROM routine ‘ptoken’, but this does not handle line number tokens. These have to be dealt with separately.

3 The markers which point to the positions of PTRA and PTRB are printed out, using the counters set while the error line was being printed.

4 Finally, a JMP is made to the default BRK handler to print out the error message.

The programs in the last few chapters are not really meant to show everything that can be done: they are really just an indication of the way that the BBC BASIC can be enhanced by overlaying procedures, or adding new commands and utilities.

Chapters 10 and 11 detail the routines inside the ROM, and the the other errors generated by BASIC, and these may give ideas for experimenting with more new command and functions, like graphics commands or statistical functions.