CHAPTER 4
CONVERSION NOTES

These notes for the conversion of the adventures in this book so that they may work on microcomputers other than the BBC Micro, consist of two main parts: the first part relates to differences which correspond directly to commands on other micro, and the second concerns a major structural disadvantage for those computers which have no “READ” and “DATA” statements and/or the inability to support multi-statement lines, like the ZX81, for example. A reasonable knowledge of programming your micro is desired, but if this does not apply to you, and you are uncertain of what is and is not relevant to your micro, then you should be able to gain information concerning this from your computer manual. It is assumed that you have a reasonable amount of memory — 16 Kilobytes of RAM is considered reasonable. The conversion notes are about those commands used chronologically in the adventure “Captive”; these are the commands which may not be supported by other micros.

I. 1) *LET* — In line 10, the LET statement has been left out. It is optional on the BBC Micro, but it is necessary on some other micros like the ZX8l. i.e. “X=0” would become “LET X=0”.

2) *RESTORE/READ* — If your computer does not have these commands then see section II of this chapter.

3) *’(apostrophe)* — This is a control for the position of the cursor on the screen — it moves it down to the beginning of the next line.

4) *CHR$* (*number*) *before inverted commas* — The colour that the text is to be printed in is controlled by the number. If your computer does not have colour then put an extra space in after the first set of inverted commas. If, on the other hand, your computer does have colour, then look at the manual for your computer if you are unsure about how to set the text colour in a program. Also note that the BBC Micro has the option of leaving out the brackets with CHR$.

5) *CLS* — This clears the screen and homes the cursor to the top left hand corner. This is standard on most machines, but on the PET, for example, this is obtained by using an inverse heart shape inside inverted commas in a PRINT statement — the cursor movements on this computer are achieved by using other similar shapes which, like the heart shape, are obtainable from the keyboard.

6) *GET$* — An alternative for this is INKEY$, which has various syntax for different computers, so look at your manual if you are in doubt

about what to use. If your computer has neither of these, then a simple INPUT statement would be put in place of "Z$=GET$" — i.e. "INPUT Z$". Remember that the carriage return key must be pressed after the "INPUT" statement while the program is running.

7) *THEN* (*line number*) — Some computers may require to have "GOTO" between "THEN" and the line number to make it become "THEN GOTO (line number)".

8) *ELSE* — This is a feature of structured BASIC, so if your computer does not have it, then do not worry. To avoid using it in line 40, for example, a new line would be taken after "IF Z$ = "N" THEN 150":

```
40      IF Z$<> "Y" THEN 40
```

9) *TIME/REPEAT/UNTIL* — "TIME" is the variable on the BBC Micro that holds the value of the computer's internal clock. "REPEAT/UNTIL" is similar to the "FOR/NEXT" loop which can replace it: for example, 60 TIME=0:REPEAT UNTIL TIME>400 can be replaced with: 60 FOR B=1 to 5000:NEXT B. The number underlined should be changed according to the time delay that you require, so that you can read the information on the screen.

10) *ENVELOPE* — This allows the programmer to change the tone produced by the "SOUND" statement. Leave this out if your computer does not have sound, or it does not have the ability to change the tone if it does have sound. If, on the other hand, you have a computer which does this, then include here the programming so that a wavering tone is produced whenever you use sound. The variable "W" controls the volume and frequency of this tone.

11) *SOUND* — Sounds are produced with this command — "W" controls the frequency at which the note starts, and the "ENVELOPE" makes it in the form of a wavering tone. If the sound on your micro cannot take this sort of format then it might be best to leave it out completely, since it may not sound very well — "ENVELOPE" can make the sound ring out continuously while the program is running, whereas "SOUND" has a set duration, which would stop at some point when the computer is waiting for a command.

12) *IF W=120 PRINT* — The BBC Micro allows the word "THEN" to be omitted between the "120" and "PRINT". Another example of this could be: IF E(4)=7 E(4)=22. If you are in doubt whether "THEN" should be put in or not, then try applying then syntax "IF (condition) THEN (expression)", and if it applies, then put it in. By omitting it on the BBC Micro, memory is saved.

13) VDU 31,x,y — The purpose of this command is to move the cursor to a location "x" columns along, and "y" rows down. The next thing to be printed will be printed at this location. A possible alternative for this on

other micros is the "PRINT AT" statement which also moves the cursor to a selected position on the screen. If, on your computer, the layout looks better without leaving rows of space between the categories, then adjust this until you achieve a display which you like.

14) *VDU 31,0,13,134* — The third number after "VDU 31" is to do with the colour that the text in the "PRINT" statement following it is to be printed in —"VDU" is similar to "PRINT".

15) *LINES 420/430* — The input line is cordoned off by lines above and below it, so the characters in these lines, when printed out, should do this.

16) I*NPUT "Command?" B$* — If your computer does not accept this then it can be changed to: 440 PRINT "Command?";

```
440     INPUT B$
```

Some computers print out a question mark when asking for an input, so if this applies to your computer, then miss out the question mark after "Command".

17) LEFT$(B$,3) — This is standard on most computers, but on the ZX's, there is what I consider to be a slightly better syntax: instead of "LEFT$", "RIGHT$", and "MID$", it just has the one format. For example:

```
LEFT$(B$,3)becomes B$(TO 3)
RIGHT$(B$,3)becomes B$(3 TO)
MID$(B$,2,4)becomes B$(2 TO 6)
```

"LEFT$" and "RIGHT$ should be easy enough to understand, since as their names suggest, they deal with the left and right parts of strings respectively. "MID$" deals with the middle of a string — the first of the number after the string name, is the number of characters from the left of the string which is being dealt with. The second number is the number of characters which will be in the substring.

18) ON M GOTO — Many computers have this but others do not. An alternative for this can be achieved by arranging the command routines in steps of line numbers which are of equal distance apart. If the step is fifty, then line 700 could become: 700 GOTO 750+M*50. 'If your's does not accept this either then you will have to do it the long way. i.e.

```
700     IF M = 1 THEN 800
701     IF M = 2 THEN 850
702     . . . . .
```

On the other hand, you could try storing the line numbers in a "DATA" statement and take the values out according to the value in "M": 700 RESTORE (line number where the data is stored):FOR B=1 TO M:READ Z:NEXT B:GOTO Z.

19) LINE 710 — This is the line that requests that the game be started

again, and has the appropriate message flashing on and off — VDU 23;11,0;0;0;0 turns off the flashing cursor, and VDU 23;11,255;0;0;0 turns it on again. If you are in doubt how to use this on your computer then you can apply the following:

```
710     INPUT "Press carriage return to start again" Z$
712     IFZ$<>""THEN 710
714     GOTO 30
```

The computer will only respond if you press return and do not press anything else.

20) *LINE 1310* — With "DATA" statements, many computers need to have words enclosed in inverted commas, whereas the BBC Micro does not.

II. I will now deal with structural problems concerning other computers. To begin with, if your computer does not have lower case text, then you will have no option, but to type things in completely in upper case.

If you are unable to apply the "READ" and "DATA" statements on your micro, then take note: instead of using "DATA" statements you must store the information in strings. To begin with I will deal with the definition of the variables in "E". The method of doing this takes up more memory, but it is simpler to apply, and will take the form of: E(l)=4:E(2)=8:E(3)=22:E(4)=16:E(5)E . . . :E(22)=22.

The information for what rooms the adventurer moves to on particular requested directions may be stored in the strings P$, Q$, R$, and S$. A suitable character will be chosen to represent zero (say CHR$(50) on the ZX81) and the characters in the strings correspond to values above, below, or equal to zero which are added to fifty and changed into characters corresponding to these values. As an example for the movements north in "Captive", I will make the values correspond directly for what they would be on the ZX81.

P$ would equal "NMNMOMMOMOJMMMIMNMMNM" from the DATA of 1,0,1,0,2,0,0,2,0,2,-3,0,0,0,-4,0,1,0,0,1,0.

On the ZX81, the routine to check and print out whether there is or is not a direction to the north is as follows:

```
320     PRINT
321     PRINT
322     PRINT "EXITS:-";
323     IF P$(A)< > "M" THEN PRINT "NORTH";
```

Also, concerned with the actual movement of the player, the routine for movement north 1ould become:

```
550     IF B$(1)<>"N" THEN GOTO 560
551     IF P$(A)="M" THEN GOTO 600
552     LET A=A+CODE(P$(A))-50
553     GOTO 160
600     PRINT "NO EXIT."
601     GOTO 160
```

The routines for the other movements would be carried out in a similar manner —instead of dealing with P$, Q$, R$, or S$ would be dealt with.

You can store the room names in A$, and have a full stop separating each name — this is CHR$(27) on the ZX8 I. If the line in which A$ is stored takes more than the screen size then you should not worry, since the ZX81 will allow this. However, to make editing easier, you may wish to have half of the room names in one string, and the other half in another string so that you will have two strings —A$, and X$, for instance:

```
LET A$="PRISON CELL.BELL TOWER. WINDING
STAIRCASE.GUNPOWDER CHAMBER.PLACE WITH A
ROCKY FLOOR.WALL WITH SCRATCHES ON IT.SIGNAL
TRANSMITTER ROOM.ROOM OF CHAINS.PADDED
CELL.AREA WITH A HOLE IN THE CEILING."
LET X$="MUDDY AREA.ALT AR.PLACE BESIDE A
MONOLITH,DIMLY LIT
PASSAGE.LOCKSMITHS,FROZEN ROOM.BRIGHTLY
COLOURED ROOM.OBSERVATION POINT.REPAIRS
ROOM.AIR LOCK.OUTSIDE OF SHIP."
```

The program lines to print out the appropriate room names are as follows:

```
300     LET BB=6
301     IF A<10 THEN GOTO 311
302     LET AA=1
303     IF AA=A THEN GOTO 307
304     LET BB=BB+1
305     IF CODE(A$(BB))=27 THEN LET AA=AA+1
306     GOTO 303
307     LET BB=BB+1
308     IF CODE(A$(BB))=27 THEN GOTO 320
309     PRINT A$(BB);
310     GOTO 307
311     LET AA=1
312     IF AA=A THEN GOTO 316
313     LET BB=BB+1
314     IF CODE(X$(BB))=27 THEN LET AA=AA+1
315     GOTO 312
```

```
316     LET BB=BB+1
317     IF CODE(X$(BB))=27 THEN GOTO 320
318     PRINT X$(BB);
319     GOTO 316
```

From the above programming information you will see how inefficient it is to use strings compared with "DATA" statements, because more program lines are required and the strings take up storage memory as well as the memory taken up by these extra lines of program. The full stop divides each piece of data, and the computer counts through this data by incrementing the variable "AA" every time it meets a full stop, and stopping when the value in "AA" corresponds with that in "A". Each character of the string at that point will then be printed on the string until another full stop is met. When the value in "A" is changed then the different room name corresponding to that value will be printed out.

The next item of data to be sorted out concerns the names of the objects used in an adventure. These could be stored in J$ as follows, or if there are a large number of objects in your adventure, you could store surplus names in K$, but it is easier if you try to have them all together:

```
LET J$= "GRENADE.ROUGH-METAL.SHINY-KEY.ICE-
BLOCK.GLOVES.SABRE.AERIAL.TORCH.HEADPHO
NES.MAGNIFIER.LOCKED-DOOR.DOOR.BELL.
SCRATCHES.KEY-CUTTER.HOLE.TRANSMITTER.
WINDOW.MUD-MAN.WIRE.INSCRIPTION.BOULDERS.
SWARCK."
```

The first thing concerning the objects is printing them out under the categories of "Objects" and "Inventory". Since the ZX81 has only thirty-two columns in the screen, it is easier for these two categories to be printed in two columns — one for "Objects" and the other for "Inventory" — side by side on the screen, for otherwise, the words would tend to overlap onto the next line. To prevent this when printing across the screen, I limit the number of letters for each object to about twelve — this can be increased to fifteen letters with two columns on the ZX81. The program lines to do this are as follows, and the values in "E" determine the presence or absence of objects:

```
360     PRINT
361     PRINT
362     PRINT AT 5,0; "OBJECTS"; AT 5,16; "INVENTORY"
363     PRINT "(7 GRAPHICS 7)"; AT 6,16;"(9 GRAPHICS 7)"
364     LET H=6
365     LET HH=0
366     FOR G=1 TO 22
367     LET HH=HH+1
368     IF E(G)=A THEN GOTO 372
```

```
369     IF CODE(J$(HH)<>27 THEN GOTO 367
370     NEXT G
371     GOTO 378
372     LET H=H+1
373     PRINT AT H+6,0;
374     PRINT J$(HH);
375     LET HH=HH+1
376     IF CODE(J$(HH))<>27 THEN GOTO 374
377     NEXT G
378     LET F=0
379     LET FF=0
380     FOR G=1 TO 22
381     LET FF=FF+1
382     IF E(G)=0 THEN GOTO 386
383     IF CODE(J$(FF))<>27 THEN GOTO 381
384     NEXT G
385     GOTO 420
386     LET F=F+1
387     PRINT AT F+6,16;
388     PRINT J$(FF);
389     LET FF=FF+1
390     IF CODE(J$(FF))<>27 THEN GOTO 388
391     NEXT G
```

(i) *LINES 369−363* print up the titles for the categories.

(ii) *LINES 364−377* print out the objects.

(iii) *LINES 378−391* deal with the inventory.

The above program lines for each category are based around the FOR/NEXT loop of "G" which counts through all the objects, checking to see if they are in the room that the adventurer is occupying, in the first case, and in the second case there is a check to see if the objects are being carried. If an appropriate object has been found, then it is printed out from "J$", the character numbers being in the variables "HH" and "FF" — like the routine for the room names, the object names corresponding to the numbers are found through counting the full stops.

The next routines are the last routines which use strings instead of "DATA" statements; the inputted commands are interpretted by the strings, and the appropriate values are stored in the variables "M" and "O". The program lines follow the string, I$, in which the first three letters of each command are stored:

```
LET I$="GETDROWEAKICRINREACUTEX
AKILLIGOPETHRSAYTAKUNLLOO"
```

```
440     PRINT AT 21,0; "COMMAND?"
441     INPUT B$
610     LET M=0
611     LET N=0
612     LET O=0
613     LET C$=I$
614     LET D$=J$
615     IF LEN B$<6 THEN GOTO 440
620     FOR I=1 TO 16
621     IF B$(TO 3)=C$(TO 3) THEN LET M=I
622     LET C$=C$(4 TO)
623     NEXT I
630     IF M<>9 THEN GOTO 650
640     PRINT AT 16,; "I DO NOT UNDERSTAND YOU."
641     GOTO 160
650     FOR J=1 TO 23
651     LET E$=B$(3 TO)
660     FOR K=1 TO 7
661     IF LEN E$=4 THEN GOTO 681
662     IF E$(1)<>"" AND D$(TO 3)=E$(2 TO 4) THEN LET N=1
670     IF D$(TO 3)<>E$(2 TO 4) THEN GOTO 680
671     LET O=J
672     LET K=7
673     LET J=23
680     LET E$=E$(2 TO)
681     NEXT K
682     LET D$=D$(2 TO)
683     IF D$(1)<> "." AND LEN D$>3 THEN GOTO 682
684     LET D$=D$(2 TO)
685     NEXT J
686     IF O<> THEN GOTO 690
687     PRINT AT 16,J; "PARDON?"
688     GOTO 160
690     IF N=1 THEN PRINT AT 16,; "LEARN TO TYPE."
```

Although the above version of the routine looks a great deal longer than the version for the BBC Micro, it is not so much longer as you think since without the ability for several statements to appear on the one line, the number of line numbers increases dramatically. However, the amount of memory taken up does not change greatly.

The string, I$, which, like the other strings, should be defined early on in the program, in the first few line numbers. There is no need in the variable I$ for full stops to be inserted between each of the groups of three letters since the intervals between the beginnings of each of the groups is constant.

(i) *LINES 440−441* —This is the input routine.

(ii) *LINES 610—614* This is the definition of the variables for the routines. The variables C$ and D$ are set equal to I$ and J$ respectively. This is because the beginnings of the strings are deleted in gaining the next bit of data, and so it would not be beneficial to delete pieces of strings which the computer needs to check against every time a command and an object are entered.

(iii) *LINE 615* — This checks for the entered command and object being less than the minimum of six letters. With the ZX81, an error will result if the computer tries to check for more characters in a string than there are in it.

(iv) *LINES 620−641* — A check is made for the first three letters of each command equalling the first three letters of the entered command. The command number is stored in "M", as applies for the version for the BBC Micro. If the command is not recognised then the "PRINT" statement in line 648 comes into action.

(v) *LINES 650−685* — A search is made for the object name entered, and if it is found, then its number is stored in the variable "O" — if not, the word "PARDON?" is printed out in line 687. The method used is very similar to that used for the BBC Micro, the main difference being that the relative information is obtained from a long string with each item separated by a full stop, instead of from a list of data. The ZX8l syntax make it look different.

(vi) *LINES 686−690* — These lines check to see if the object has been recognised and that a space has been inserted between the command and the object, printing out the relative messages depending on the relative conditions.

The second structural problem inherent in the ZX8l is the inability to use more than one statement on a line. The best way to deal with this is to give examples, and from these you can apply the necessary changes to other lines in the programs:

EXAMPLE 1:- Change line 260 from:

```
260   IF E(4)=7 E(4)=22:T=1:E(7)=19:E(20 =22:PRINT
      CHR$130"The transmitter has cooled down,"CHR$130"but it
      does not have an aerial."
```

To:

```
260   IF E(4)<> 7 THEN GOTO 270
261   LET E(4)=22
262   LET T=1
263   LET E(7)=19
264   LET E(2)=22
265   PRINT "THE TRANSMITTER HAS COOLED DOWN, BUT
      IT DOES NOT HAVE AN AERIAL."
```

You will see from the above example, the advantage of numbering line numbers in steps of ten, since other lines can be slotted in without much trouble. This is particularly necessary on the ZX81, where several lines may be taken up, compared with only the one line on other computers. Also, from this example, you will see the conversion notes coming into use: in line 260, "THEN GOTO" is used; the command "LET" is required in lines 261, 262, 263, and 264; and the "PRINT" statement in line 265 is in block capitals without colour, and it fits into the thirty-two characters per line, therefore corresponding to the ZX81 screen format. You may have noticed that where the assimilation of a condition spans over several lines, instead of checking for the condition being met, the computer checks for the condition not being met — this is particularly noticeable where the condition consists of two or more parameters, for otherwise memory would be taken up checking it in each line. This can be seen in the next example.

EXAMPLE 2:- Change line 820 from:

```
820   IF O=4 AND E(5)<>−1 PRINT "It is too cold to carry.":
      GOTO160
```

to either:

```
820   IF O=4 AND E(5)<>−1 THEN PRINT "IT IS TOO COLD TO
      CARRY."
821   IF O=4 AND E(5)<>−1 THEN GOTO 160
```

or:

```
820   IF O<>4 OR E(5)=−1 THEN GOTO 830
821   PRINT "IT IS TOO COLD TO CARRY."
822   GOTO 160
```

Although the first version of line 820 for the ZX8l is contained in two lines, it is more clumsy than the second version, which requires three lines. There are several rules for the changing of a condition being met to the condition not being met, which are as follows: "AND" should be changed to "OR"; "OR" should be changed to "AND"; something equalling something else should be changed to the first thing not equalling the second; and where there is an inequality to begin with, an equality will take its place.

EXAMPLE 3:- Change line 270 from:

```
270   IF A=11 AND E(6)<>0 AND E(19)=A PRINT CHR$130 "A
      mud-man has just killed you":GOTO 710
```

to:

```
270   IF A<>11 OR E(6)=0 OR E(19)<>A THEN GOTO 280
271   PRINT "A MUD-MAN HAS JUST KILLED YOU."
272   GOTO 710
```

This example is mainly just to concrete the points nade in the two previous examples. You will soon find that it is not very difficult to convert multistatement lines to single statement lines.

A way of gaining some skill in converting programs, which are originally written for other computers, to work on your own computer, is to look for interesting programs in computer magazines, to make the necessary alterations so that the syntax is correct, and then to type them in. By trying this, you will help to develop your own skills as a programmer, since a greater effort will be required, than simply typing in programs which are written for the micro you use. Instead of keeping the programs just as they are in the magazines, you will find that there are bits in them which you think can be improved upon, and so you will have a greater impulse to modify them, since you will already be changing parts of them to suit your own micro. If your computer happens to have more features than the computer that the program was intended for, then you may wish to try and modify it to include your own computer's features — for example, if the program does not include colour, and you would be able to include this feature into into, then you would do so, provided that the resulting screen format turns out to be pleasing.

By following out such techniques you will gain a more general knowledge of programming in BASIC which you will be able to apply to virtually any computer that can be programmed in this language. If you decide to buy a better computer in the future, then you will be able to apply your old programming techniques almost from the word "GO", and then expand your knowledge to include the features provided by that computer.

I trust that if you do not have a BBC Microcomputer, then the conversion notes in this chapter are sufficient for you to be able to use the programs with your own computer. The first section, which includes various smaller syntax details, and the second section, which includes structural details, deal with the possible ways in which your micro may differ from the BBC Micro. I have found that the BBC Micro was a good computer to base this book around, since it has many features, like colour and sound, which are becoming a standard in the more recent computers, and it also has a version of BASIC which is well structured and easy to use — the ZX81 is suitable for basing the structural details around, because it is at the lower end of the market, and lacks the features which the more expensive computers have: two major difficulties about it which I have tackled are the absence of "READ" and "DATA", along with the inability for multi-statement lines. In short, most other computers are upwards compatible to it, and so if there are conversion details which are specifically concerned with this computer, then conversion to other computers will not be much of a problem.