subroutines called by the control program. Our listings show how two of these routines might look. The first (beginning at line 4000) simply prompts the user for a number between 1 and 20 (the word length). It uses a general-purpose subroutine that is assumed to exist at line 51000, which will take a string specified in PROMPT$, print it and then accept a number input by the user. If this number is not an integer that falls between the limits set by MIN% and MAX%, an error message will be given and the user will be asked to input a new number. This subroutine may easily be used in other programs, and a library of such general-purpose modules may be built up for use in later projects.

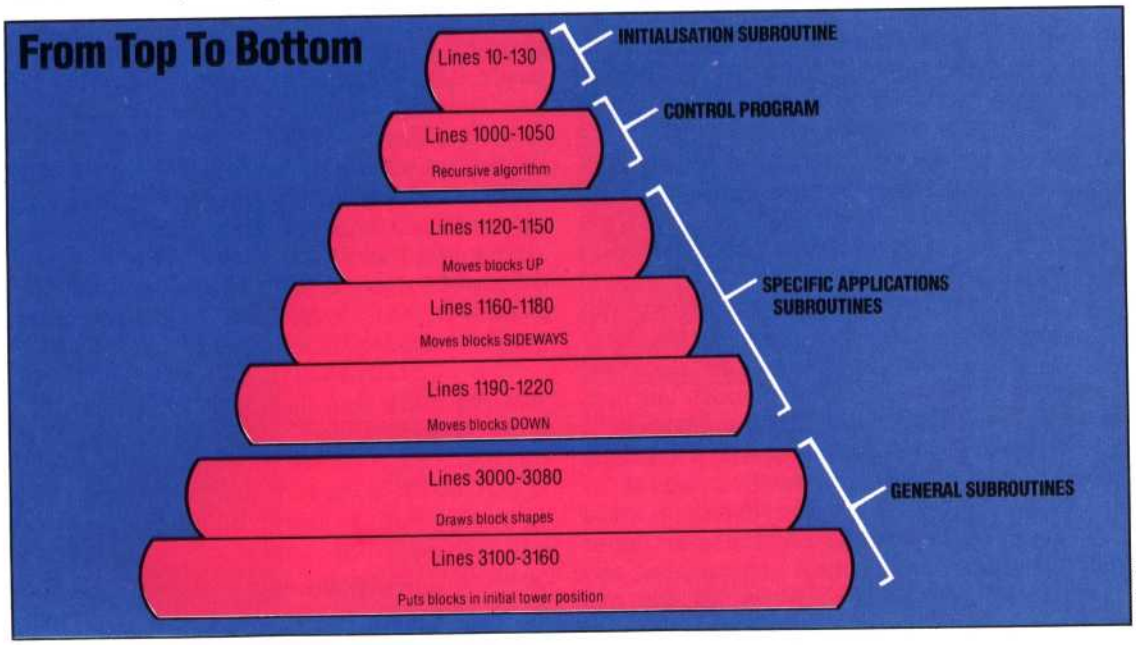
```
4000 REM Discover word length from
player
4010 REM
4020 PROMPT$="How many letters are
there in your word ?"
4030 MIN%=1
4040 MAX%=20
4050 GOSUB 51000:REM input an integer
between MIN% & MAX%
4060 WORDLEN%=RESP%:REM RESP% is used by
the subroutine at 51000 to pass back
the response
4070 RETURN

8000 REM select data set and load it
8010 REM
8020 IF WORDLEN%>7 THEN FILE_L%=8
                   ELSE FILE_L%=WORDLEN%
8030 FILENO_L$=STR$(FILE_L%)
8040 FILENAME$="TABLE"+FILENO_L$
8050 GOSUB 9000:REM OPEN, READ & CLOSE
the file with the likelihood data for
the appropriate word length.
8060 RETURN
```

The other routine (beginning at line 8000) uses local variables (FILE L% and FILENO L$). We have assumed that the data needed to guess a letter is in eight sets of tables that give the likelihood of finding any particular letter next to any other. As we want only one set of data in RAM at any time, we must build up a string in FILENAME$ to hold the name of the data file, and then call the subroutine at line 9000 to read the file.

In many cases, we will find that our program will move directly from one routine to another. However, we will usually want to create an extra routine that calls each of the other two in turn. This may seem like an unnecessary complication, but it allows us to keep a tight control over the program's 'flow' and it has the added bonus of keeping program modules separate so that they may be easily added to other programs.

This use of subroutines that are transportable from one program to another does involve extra work, and care must be taken when designing the routines so that they are suitable for use in a wide variety of circumstances. This may often be achieved simply by replacing constants with variables. It is important that all subroutines should be well documented. The documentation should specify the exact purposes of the routine, giving details of the variables used, the values expected as input and output, and any side-effects (moving the cursor position, changing the memory map, closing files, and so on).

A standard layout is also very helpful; you should make sure that all line numbers have a fixed interval, the titles and comments are restricted to a set number of lines at the beginning of the routine, and that RETURN is always on the last line. Be sure to note the first and last line number of each routine. When a library routine is required, make sure that the program has an appropriate gap in its line numbers and then MERGE the subroutine into the program. If your micro has no MERGE command, it may be possible to use a text editor to combine programs that have been SAVEd in ASCII format rather than the usual 'tokenised' form. If this is not possible, your library subroutines will need to be typed in each time they are used. However, the fact that they will not need to be redesigned should make the extra work worthwhile.



Top-down Programming
This diagram illustrates the principle of top-down programming. We have used the Towers of Hanoi program that appears on page 475. The line numbers in the diagram refer to the BBC listing.

The first layer of the structure represents the initialisation program, which must be completed before the rest of the program can be executed. The CONTROL PROGRAM in our diagram represents the recursive algorithm, which performs the calculations and calls the other subroutines as necessary. The SPECIFIC APPLICATIONS SUBROUTINES (lines 1120 to 1220), are used to move the block shapes from pile to pile in the display. The final two sections of the diagram, GENERAL SUBROUTINES, represent the last two sections of the program that are used to format the initial display and create the design for the blocks. Compare this structure with the listing, and you will see that the program is constructed in exactly this sequence