Branching Out

As a long program is developed, its structure takes on the appearance of a tree, with more branches at each successive stage of refinement

In the last instalment of the Basic Programming course, we took a look at some of the problems involved in searching through a list to find a specific item — assuming that the list had already been sorted into order. This is a topic to which we will return in more detail when the time comes to start writing search routines. In the meantime, however, we will develop the theme of top-down programming to produce code for the second two parts of the main program. This contains four calls to subroutines or procedures:

```
MAIN PROGRAM
BEGIN
   INITIALISE (procedure)
   GREET (procedure)
   CHOOSE (procedure)
   EXECUTE (procedure
END
```

The first procedure, *INITIALISE*, will involve numerous fairly complex activities — setting up arrays, reading data into them, performing various checks and so on — and we will leave the details of this procedure until later. The next two parts of the main program comprise the GREET and CHOOSE procedures. In developing these procedures, we will suggest a methodology that helps prevent the many layers involved in top-down program development from becoming disorganised and confusing.

The problem with the top-down refinement approach to program development is that the number of steps needed before we are ready to start coding into a high level language is indeterminate. Two or three steps may be enough for simple procedures, but more difficult procedures may require many steps before the problem has been sufficiently analysed to allow 'source code' (as the high level language program is called) to be written. This means that writing a program using this method is similar to drawing a tree lying on its side. As the 'branches' proliferate (that is, as the refinements become more detailed) they take up more space on the page. Eventually, it becomes impossible to fit everything onto a single sheet, and that is the point where it becomes easy to lose track of what's going on.

One very effective way to organise the documentation of the program is to number the stages of its development systematically. We have used Roman numerals to indicate the level of refinement and Arabic numerals to indicate the subsection of the program. A separate sheet of loose-leaf paper is then used for each level of refinement and the pages for each program block or module can be easily kept together. Here is the numbering system for our program:

```
I MAIN PROGRAM
BEGIN
   1. INITIALISE
   2. GREET
   3. CHOOSE
   4. EXECUTE
END
```

As mentioned above, we are leaving the development of INITIALISE for the moment, and concentrating on developing the GREET and CHOOSE procedures.

```
II 2 (GREET)
BEGIN
   1. Display greeting message
   2. LOOP (until space bar is pressed)
      ENDLOOP
   3. Call *CHOOSE*
END
```

```
III 2 (GREET) 1 (display message)
BEGIN
   1. Clear screen
   2. PRINT greeting message
END
```

```
III 2 (GREET) 2 (LOOP wait for space bar)
BEGIN
   1. LOOP (until space bar is pressed)
         IF space bar is pressed
         THEN
      ENDLOOP
END
```

```
III 2 (GREET) 3 (call *CHOOSE*)
BEGIN
   1. GOSUB *CHOOSE*
END
```

At this point it should be clear that **III-2-1** and **III-2-3** are ready to be coded directly into BASIC, but that **III-2-2** needs another stage of refinement:

```
IV 2 (GREET) 2 (LOOP)
BEGIN
   1. LOOP (until space bar is pressed)
         IF INKEYS is not space THEN continue
      ENDLOOP
END
```

We are now at the point where all the coding into