# CHOICE OPTIONS

**In any complex program there will be points at which the user will need to branch to one of a number of available options. The menu system lays out the list of choices at certain specified stages for the user to select from; a command system allows the user to choose from a range of options at any time. We consider both techniques.**

Menus may be simple lists of numbered items or they may be screens full of elaborate icons, but the principle behind their use is the same. A menu is used when the program reaches a multi-way branch in its logic; the user is asked to choose which route to take from a list of available options displayed on the screen. Menu-based programs tend to be tree structures: the user enters the tree at the 'root' and is guided by the menu options towards one of the 'leaves', where the information or function is to be found.

The major advantage of this approach is that the user needs little or no knowledge of the program structure because the route is well 'signposted' all the way. However, more experienced users find the job of navigating their way through a chain of menus tedious for often-repeated tasks. Novices, too, may have difficulty with the tree structure; correcting a wrong decision involves working back through all the menus to the point at which the mistake was made, re-entering the correct option, and then continuing from there. Prestel is a particularly 'deep' structure of this type, and users frequently encounter this problem. Menus need not form a tree at all — they may be organised into a network by using loops. However, this tends to increase the risk of getting lost within the program structure and so is not suitable for novice users.

Designing a menu system can be difficult, although the actual programming is relatively easy. The main problem is that the entire program must be clearly specified before any code is written. (This is good practice anyway, but is not always straightforward.) Adding new functions at a late stage can involve changing several menus earlier in the program, and this may require major restructuring. When the program is designed, all menu logic should be included in a single routine that calls the routines at the 'leaves' when these are reached. The menu routine can thus be seen as a more complex form of the normal control routine, with all internal branching controlled by the user. This keeps the design tidy, and serves to separate the control logic from the functional parts of the program, allowing each to be developed and debugged independently.

Program flow will now follow a set pattern. For each menu along the route, the menu logic routine passes a set of user prompts to a routine that puts them into 'slots' in a menu 'frame'. The slots will probably contain a screen header message that displays a title and any other necessary information about the menu, a 'footer' that explains how to make the choice (with sufficient space for the user's response), and the menu options themselves. In general, the most effective menu layout has up to eight options displayed in a column, with the response code (number, letter, mnemonic, etc.) at the left of each item.

The menu routine calls an input routine, perhaps passing to it the conditions that must be specified for a legal input, and accepts in return the user's response. It then interprets this response (typically a single keypress) and either passes control to the next menu or calls the appropriate application routine if it is the last menu in the chain. Once the routine has been executed, the menu from which it was called might be redisplayed, or control could pass to some other part of the program (the root menu, perhaps).

Menus require a lot of text for headers, footers and prompts, but much of this will be repeated for each menu 'frame'. The explanation of how to choose a menu option (the 'help' command), an option offering an exit to the root menu, and other recurring choices may all be required by several different menus. If this is the case, space may be saved and the logic made clearer if all prompts are held in a string array (or on a random access disk file), from which they may be called by their index number. Design the menu display routine to accept references to this array and to display the appropriate headers, footers, prompts, etc.

A command-driven system is one that has a range of commands available to the user at any stage in the program. Each command goes straight to a subroutine that performs the required function. This system must be designed to inspect all input to ascertain whether it is data or a program command. The difference is usually signalled by the user pressing a particular key before each command input. The Control key is often used for this purpose. A word processor, for example, might accept the word 'save' as just one more word of text, but interpret it as a storage command if the Control key was pressed before the word was typed.

In a command-driven system, the 'tree' is very shallow and broad, and a single routine, acting as the control program, is used to direct the user to the required subroutine. This 'command interpreter' has four main tasks. The first is simply