LENDING LIBRARY

It is extremely useful to develop techniques that make more efficient use of the time and effort spent in programming. We discuss one such method — creating libraries of routines that can be merged into programs — and list the sort of details that must be taken into account when programmers share the task of coding.

Following the structured design methods that we have already described in this course may seem like a long-winded approach — but it does, in fact, save time (not only in the coding but especially in the debugging of a program). This is because programs that are created at the keyboard tend to have unnecessarily complicated structures and algorithms, which means that they take longer to write, are more prone to error and, because they are more difficult to follow, take much more effort to test and debug. Planning the program in advance simplifies the structure and the algorithms and thus leads to fewer coding errors and easier testing and debugging.

Most importantly, designing ahead saves the programmer from writing a control or file structure that is later found to be inadequate (perhaps not enough space in a field in the file has been allowed for). Problems like this, which are fundamental to the way the program works, can lead to major portions of it needing to be rewritten.

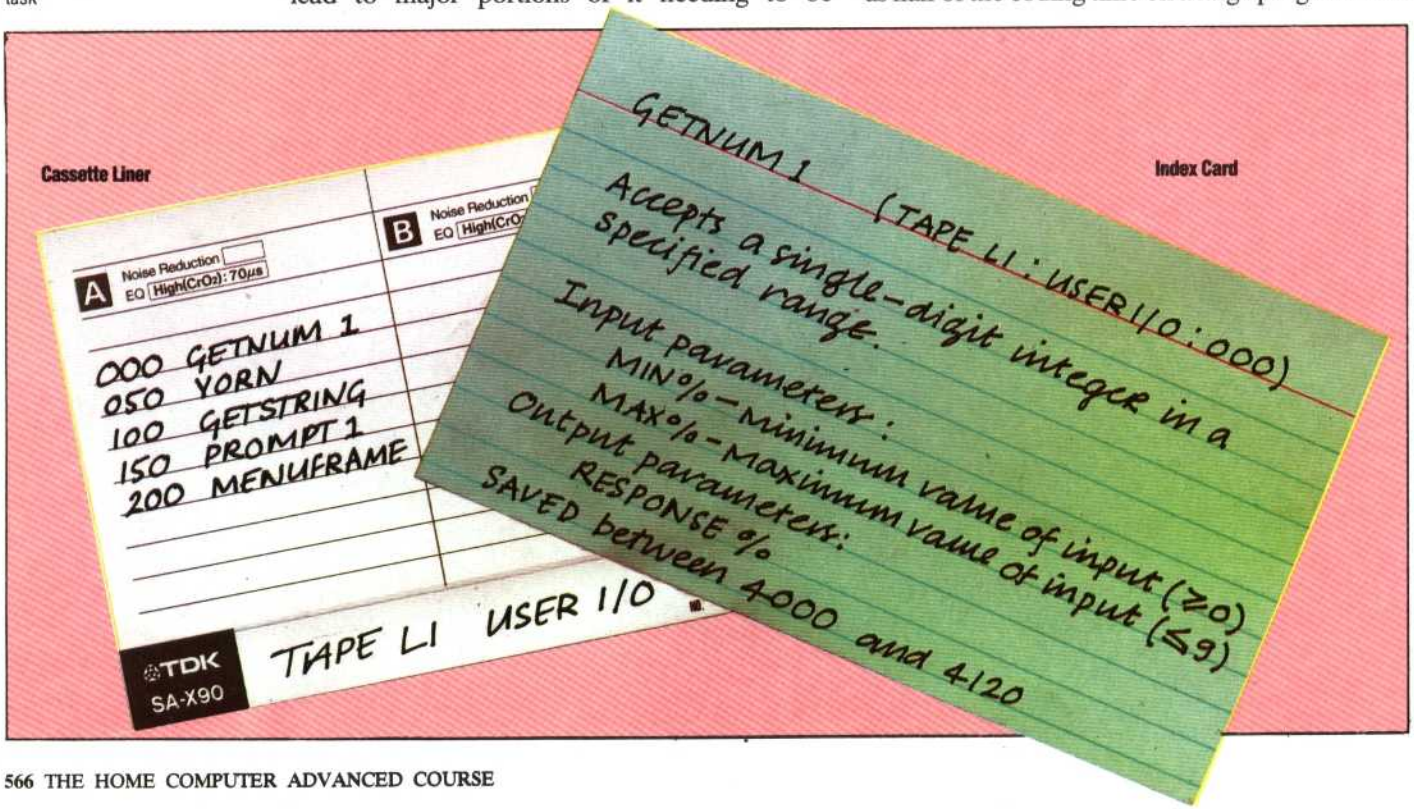
Those with a 'proper' typewriter-style keyboard may like to invest some time in learning to touch-type. Apart from this, though, there is little that may be done to increase the speed at which program lines are entered at the keyboard. However, the process of coding programs may be greatly speeded up in several ways. The first is the simplest: invent, adopt and use a number of 'conventions' when coding. Such measures include: using particular types of name for local variables to differentiate them from main program variables; beginning each subroutine at lines ending in 000; ending each subroutine with RETURN on a line of its own; starting each type of subroutine in a particular block of lines (file-handling routines between 9000 and 9999, utilities at 50000 onwards, and so on).

The benefits of using these conventions are numerous: you don't have to hunt for the menu routines because you know that they are always in the same place; you don't have to worry about whether you have used the same variable name in the main program and in a subroutine — because its name will indicate it is a local variable.

PROGRAM LIBRARIES

Such coding techniques are also useful when libraries of programs are created. A well-organised library of subroutines can save as much as half of the coding time on a large program. The

Uniform System
Libraries of subroutines are useless without a uniform documentation system accompanying them. This is especially true for cassette users — inspecting the contents of an undocumented cassette by loading and listing each program is a thankless task



MIKE CLOWES