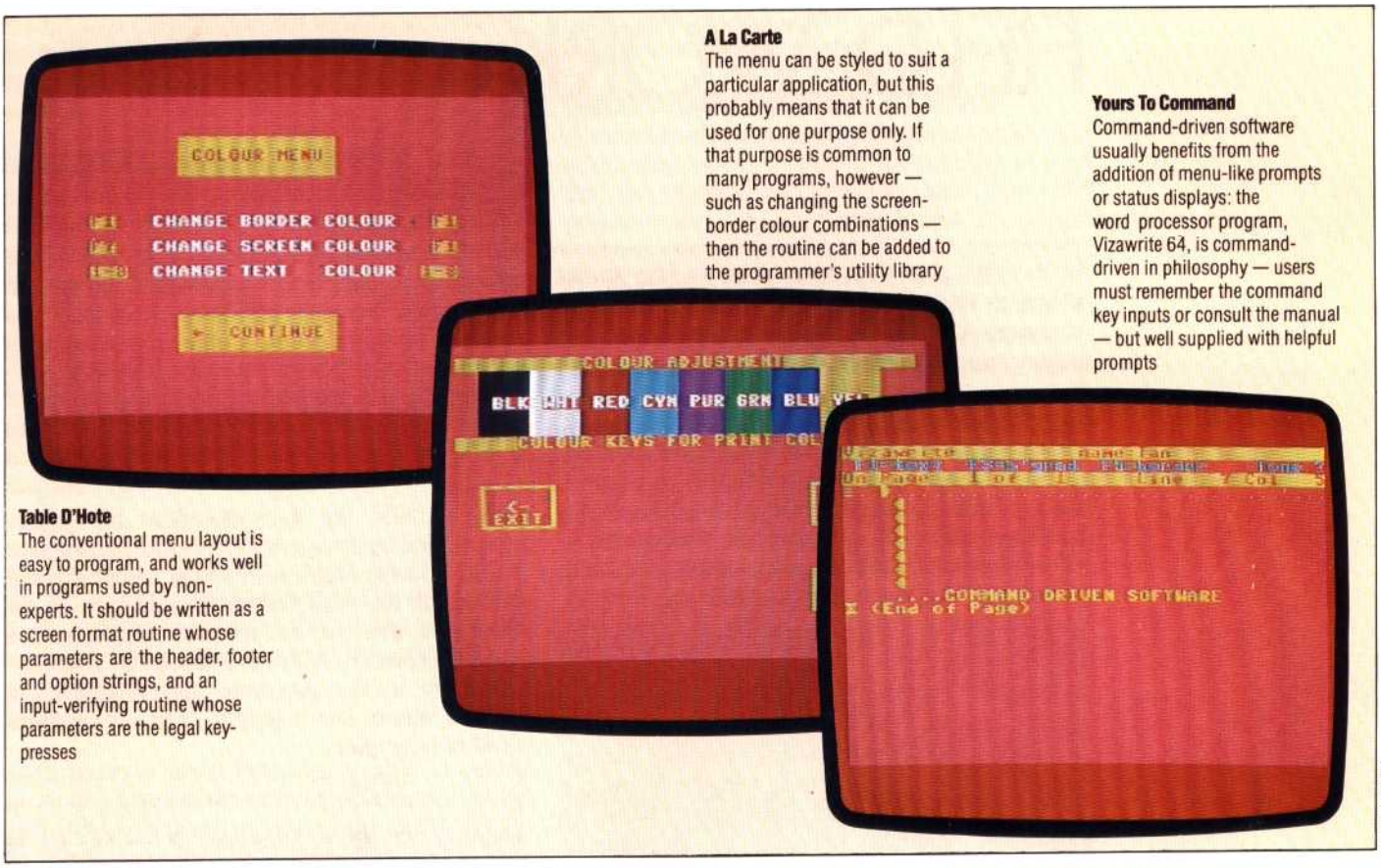

A La Carte
The menu can be styled to suit a particular application, but this probably means that it can be used for one purpose only. If that purpose is common to many programs, however — such as changing the screen-border colour combinations — then the routine can be added to the programmer's utility library

Yours To Command
Command-driven software usually benefits from the addition of menu-like prompts or status displays: the word processor program, Vizawrite 64, is command-driven in philosophy — users must remember the command key inputs or consult the manual — but well supplied with helpful prompts

Table D'Hote
The conventional menu layout is easy to program, and works well in programs used by non-experts. It should be written as a screen format routine whose parameters are the header, footer and option strings, and an input-verifying routine whose parameters are the legal key-presses

IAN McKINNELL

to wait for an input. The second is to 'parse' this input — the interpreter must separate the input line into its functional units. The third task is to interpret the command by preparing the appropriate subroutine call. (What is the routine's address? Are there parameters to be passed?) Finally, it must actually call the routine to be executed. When control returns, the interpreter goes back to its first job — waiting!

The format of a command may be extremely elaborate, and some command languages are similar to a simplified form of English. An example of a command language is the Unix shell, where the typical command format is:

Command + optional parameter list
e.g. L
or L -1

Here, the Unix command L lists a file directory, while L -1 (where -1 is an optional parameter) lists a file directory in 'long' format.

The parser must be able to recognise the various parts of the command line. Unix keeps things simple (in most cases) by taking the first word as the command and recognising parameters by a preceding minus sign. Command language parameters are not for the use of the command interpreter itself, but are required by the subroutines that the interpreter calls. Routines used in the command system should ideally adopt a standard format for input parameters. If this is done, the command interpreter can pass the parameters in the form in which they were entered (as strings, perhaps).

It is obviously much easier to create a command interpreter than to write a menu system. Experienced users tend to prefer command systems, as these are faster and more flexible than menu-driven programs. Most operating systems are command driven, which is unfortunate for novice users as such systems provide no signposting facilities and the on-line help routines (if there are any) require some knowledge of the system. In addition, the sheer number of commands and optional parameters in a typical command system means that even those reasonably familiar with the system will require help facilities or need to consult the operating manual frequently.

Beginners hate commands and experts hate menus. This problem is virtually insoluble, although some hybrid systems exist that can be quite effective. For example, the Wordstar word processing program is basically a command-driven system, but it can appear to the user to be a menu system. The commands are control codes (some with parameters) and the user runs the system entirely with these. The menus that appear on the screen use these commands as mnemonics for selecting options so that, as the novice uses the menus to run the program, the commands are learnt at the same time. The help level may be set to permit the menus to be dispensed with once the user is sufficiently proficient to find them a nuisance. The Wordstar menu system is only two levels deep, however; this approach would be more difficult to implement with more complex systems.

Two Into Three
There are usually many ways of solving the same problem — these displays show different ways of allowing the user to change the screen, border and text colours on a Commodore 64