COMPUCOLOR TEXT EDITOR

The text editor is a RAM program designed to facilitate the generation and correction of source programs and other text files. The program is supplied on disk and occupies approximately 4.2K bytes of the user RAM space when loaded. The remainder of the RAM is used by the editor as a buffer in which editing or file creation takes place. Because of the capabilities of FCS, file sizes are limited only by the disk space available and not by the buffer size. The buffer size only limits the amount of the file which is available for editing at any given time.

The program operates by reading the file into the buffer where it can be edited and then outputting the corrected text to a new file. This is done using the APPEND and WRITE commands. APPEND reads data from the input file onto the end of the data in the buffer and WRITE writes the data from the front of the buffer onto the output file. APPEND and WRITE read and write a number of lines from 1 to 255.

When using the APPEND and WRITE commands, all text enters and leaves the buffer sequentially. The flow of text is always from the old file to the new file via the buffers. Text residing in the buffer can be edited in any order using the various commands available. However, once it is written from the buffer it cannot be edited again on this pass.

How to Load The Editor

Place the Editor disk in the computer, enter FCS by typing ESC D, and type 'RUN EDT', as shown below:

```
FCS>RUN EDT
```

The computer will respond with a '>'. Once loaded, the EDT disk is no longer required and may be removed if desired. At this point, either of the following commands may be entered:

```
>CRE<file specifier>      For example: >CRE TEST
or
>EDIT <file specifier> TO <file specifier>
```

The first command is used in order to CREate a new file, and the second is used to EDIT an existing file. In the event that the edited file will be on the same disk with the same name then the "TO" and second file specifier may be omitted. If a file type is not specified, then SRC (source) is assumed. Unless otherwise specified, the editor will edit the latest version.

The user must not change the disk, or delete any file on the disk during the use of the editor. This restriction is necessary since the editor keeps pointers to disk location for use of the APPEND and WRITE commands.

Upon entry with either the CREATE or EDIT command, the editor will print:

```
COMPUCOLOR II FILE EDITOR V7.78
EDT>
```

and await commands.

COMMANDS

Editor commands are entered by their first letter only. The escape key is used as string delimiter (defines the end of a string) and echoes as "$" on the screen. Normal escape sequences have no effect while in the Editor. Command lines are terminated by a carriage return. Until a command line is terminated, it may be cancelled or edited.

Two differect color prompts '>' are used by the editor. A yellow prompt indicates that the editor is in the command mode and a purple prompt indicates that the editor is in the insertion mode. This is done so that commands will not accidentally be entered into the text.

A parameter in the range 0-255 may be applied to some of the commands (larger numbers will be interpreted as the low order 8 bits of their value.) A negative sign may be used in some cases to indicate a reverse direction. The parameter will always preceed the command upon which it operates.

Multiple commands can be entered on a single line and then executed. The editor executes this stream one character at a time. Commands may also be executed repetitiously by using a repeat factor followed by the command enclosed in brackets ('[', ']') (see examples). This is called a command loop. Command loops may be nested inside another loop -- up to 16 levels deep. If a closing bracket is omitted, the loop will not repeat. If an extra closing bracket is entered for which there is no corresponding open bracket, execution will terminate with an 'ILLEGAL COMMAND!' message and await a new command line. A negative sign in front of a repeat factor is ignored.

An error or the exection of an escape will always break out of any loop and cause the rest of the line to be ignored.

Input Editing

Input to the Editor is buffered until a 'return' is entered. Until the line is entered, either the backspace or delete key will remove the last character in the line. Erase line (control K) will clear the line, and erase page will clear the screen without altering the line contents.

Color Codes and foreground/background selection codes are accepted as normal characters and placed in the buffer, as are A7 ON (control N), TAB (control I), BLINK/A7 OFF (control O).

For the convenience of users with cursor control clusters cursor up may be used as escape, cursor right as tab, home as carriage return, cursor down as line feed and cursor left as delete.

The use of carriage return and line feed keys differs. Carriage return inserts a carriage return/line feed sequence and terminates the

line. Linefeed also inserts a carriage return linefeed but does not terminate the line. In this way carriage return/linefeed sequences may be included in string parameters to be inserted or searched for. When used after an 'I' command the results would be identical.

The line is automatically terminated when the limit of 82 characters is reached. No carriage return/linefeed sequence is inserted in the buffer.

The user can exit the editor by either a CPU reset or the 'X' command. It can be re-entered by typing 'ESCAPE A'.

Before continuing, the user should be familiar with the use of two terms described below:

LINE -- a line is a string of text terminated by a carriage return line feed (CRLF). The CRLF ocuupies two character positions, but is not visible on the screen. These two characters must be considered when repositioning the pointer or the pointer may be positioned differently than expected. A line is limited to 82 characters and will be automatically terminated when the limit is reached.

POINTER--The pointer represents the location in the buffer at which an edit is to take place. Several commands are available which enable the user to position the pointer at any location in the buffered text.

The following is an alphabetic list of the commands recognized by the editor. In these examples the charater '$' is always the escape character. The '$' key will not produce the desired result.

APPEND

Reads the specified number of lines from the input file and appends them to the end of the buffer. If the buffer approaches within 256 bytes of being full, append termiates with BUFFER FULL!'; thus leaving some room for insertion. A negative number is treated as positive and '0A' has no effect. If the end of the input file has been read or there never was an input file (as in a CREATE command) append has no effect. Append always leaves the pointer at the end of the buffer.

EXAMPLE: '255AB255T(return)' reads in 255 lines, backs the pointer to the beginning of the buffer and prints 255 lines.

BEGIN

Moves the pointer to the beginning of the buffer. All data (if any) is placed in front of the pointer and subsequent 'T' commands will print beginning with the first line in the buffer. All parameters are ignored. No errors are possible.

Example: See above.

CHARACTER

Moves the pointer the number of characters specified down in the buffer or up if the parameter is negative. 'OC' and '-OC' have no effect. Carriage return/linefeed sequences (end of line) count as two characters and must be considered when using 'C' to move the pointer from one line to another.

Example: If the commands 'B255T(return)' reveals this in the buffer:

```
EDT>B255T(return)
TEST LINE 1
TEST LINE 2
TEST LINE 3
EDT>
```

the subsequent commands '15CT(return) and '-3CT(return)' have the following effect:

```
EDT>16CT(return)
T LINE 2
EDT> -3CT(return)
TEST LINE 2
EDT>
```

DELETE

Deletes the specified number of characters starting at the pointer down into the buffer or up if the parameter is negative. 'OD' or '-OD' has no effect. See CHARACTER concerning end of line sequences. Negative deletions do not delete the character at the pointer as do forward deletions. Rather they delete beginning with the character just before the pointer.

Example: With the buffer contents as described in the CHARACTER example, the commands '15C3DB255T(return)' would have the following effect:

```
EDT> 13C5DB255T(return)
TEST LINE 1
LINE 2
TEST LINE 3
EDT>
```

EXIT

Writes all data in the buffer to the ouput file, copies the remainder of the input file (if one exists) to the output file and sets up a directory entry for the output file. APPEND, WRITE and EXIT commands are all subject to FCS system errors. When the Editor receives an error from FCS, it returns with the message 'SYSTEM FILE ERROR!' in order to allow the user to correct the problem. To avoid this situation the user should observe the following rules:

Do not disturb the file device (disk drive unit) while editing excepting the input device after all data has been read from the file.

Do not use FCS commands and return to the Editor.

Make sure that there is enough space for the output file before you enter the Editor. You can not delete files to make room after the Editor has started.

If these rules are followed there should be no errors except those actually generated by the disk drive unit.

FIND

Searches the buffer for the specified string from the pointer forward until a match or the end of the buffer is found. If a match is found the pointer is moved to the first character following it, else the message 'SEARCH STRING NOT FOUND!' is displayed and the pointer is not moved. All parameters are ignored. The search string is the set of characters after the 'F' command through but not including the string delimiter ESCAPE '$'.

Example: Assume the following is the buffer:

```
EDT>B255T(return)
THIS IS A TEST OF THE FIND COMMAND.
EDT>
```

Then the commands 'FTEST$T(return)' and 'FTEXT$T(return)' have the following effect:

```
EDT>FTEST$T(return)
OF THE FIND COMMAND
EDT>FTEXT$T(return)
SEARCH STRING NOT FOUND!
EDT>
```

The second command's result is an error because the string 'TEXT' was not after the pointer. A subsequent 'T' command would reveal that the pointer is unchanged.

```
EDT>T(return)
OF THE FIND COMMAND
EDT>
```

INSERT

Inserts the specified string in the buffer after the pointer. The pointer will not move and will be at the same character it was before the command. Characters following the pointer are not changed. If the buffer becomes full the message 'BUFFER FULL!' will be displayed. Some part of the inserted string may have actually been inserted. Insertion mode does not

terminate until an escape '$' is entered. If a carriage return is entered the current line will be processed and a new line started. When the prompt is printed the user will still be in the insertion mode. When in insertion mode the prompt will print out in purple rather than yellow so that commands will not accidentally be entered into the text.

Example: Assuming the buffer contents as for the FIND example, the command 'BFTEST$I OF THE INSERT AND$B255T(return)' will result in:

```
EDT>BFTEXT$I OF THE INSERT AND$B255T(return)
THIS TEXT IS A TEST OF THE INSERT AND OF THE FIND COMMAND
EDT>
```

KILLS

Kills (deletes) the specified number of lines from the pointer forward or from just before the pointer backward if the parameter is negative. 'OK' deletes from the pointer to the beginning of current line. A line is defined as a set of characters terminated by a linefeed. If the pointer is in the middle of a line, 'K' will delete only that part of the line from the pointer forward. A 'K' command will delete what a 'T' with the same parameter would display.

Example: Assume the buffer contents to be:

```
EDT>B255T(return)
     LINE 1
     LINE 2
     LINE 3
     LINE 4
     LINE 5
     EDT>
```

The command '2L2KB255T(return)' will jump forward two lines, delete two lines, then return to the beginning of the buffer and print it.

```
     EDT>2L2KB255T(return)
     LINE 1
     LINE 2
     LINE 5
     EDT>
```

Now, the command '1L3CTKB255T(return)' will produce:

```
     EDT>1L3CTKB255T(return)
     E 2
     LINE 1
     LINLINE 5
     EDT>
```

Note that the 'E 2' has been deleted and that the printout from the two 'T' commands has been concatenated. Notice also that the KILL removes the CRLF from LINE 2, which resulted in LINE 5 being added to it.

LINES

Moves the pointer forward or backward the specified number of lines. The command 'OL'moves the pointer back to the beginning of the current line.

Example: Assuming that the buffer contains:

```
EDT>B255T(return)
LINE 1
LINE 2
LINE 3
EDT>
```

The command '1L3CT(return)' will move the pointer to the fourth character of second line and the command 'OLT(return)' will move it back to the beginning of that line.

```
EDT>1L3CT(return)
E 2
EDT>OLT(return)
LINE 2
EDT>
```

SUBSTITUTE

Searches for a specified string from the pointer foreward unitl the first occurrence of the string. The string is then replaced by a second specified string. Exactly equivalent to a 'find' string, negative delete for the length of the first string and an insert of the second string. All parameters are ignored. If the insertion string is null the frist string will be found and deleted only.

Example: Assume the buffer contains:

```
EDT>B255T(return)
THIS IS A TEST LINE Q6%4%44Q FOR HTE SUBSTITUTE COMMAND
EDT>
```

The command 'SQ6%4%44Q$$' will delete the garbage from the line and the command 'SHTE$STHE$(return)' will correct the spelling error. In the first case a null string is substituted for the garbage.

```
EDT>SQ6%4%44Q$$(return)
EDT>SHTE$THE$B255T(return)
THIS IS A TEST LINE FOR THE SUBSTITIUTE COMMAND.
EDT>
```

TYPE

Types the specified number of lines beginning at the pointer going forward or backward if negative. 'OT' types from the beginning of the current line to the pointer.

Example: See above examples. Assume that the buffer is the result of the above (as for SUBSTITUTE)

```
EDT>FLINE$OT(return)
THIS IS A TEST LINE
EDT>
```

Note that the find places the pointer just after the search string.

WRITE

Writes out the specified number of lines to the output file, starting from the beginning of the buffer. All data written is deleted from the buffer. Once data is written it cannot be returned to the buffer. A negative sign is ignored. 'OW' has no effect. See possible errors under EXIT.

Example: The command '255A255W' will read 255 lines from the input file and transfer them to the output file. This is a much faster way to copy the remnant of the input file than the 'E' command which copies one byte at a time. The copy function of exit is limited to prevent accidental loss of data.

X

Temporary exit from the Editor to FCS. A subsequrnt escape '^' sequence will return to the Editor if buffer and pointers are not altered. See cautions in EXIT.

ZETA

Move the pointer to the end of the data in the buffer. All parameters are ignored.

Example: The following illustrates the use of the 'Z' command:

```
EDT>B255T(return)
FIRST LINE
MIDDLE LINE

EDT>ZILAST LINE
EDT>$B255T(return)
FIRST LINE
MIDDLE LINE
LAST LINE
```

OTHER EXAMPLES

EDT>LT move pointer one line and display line

EDT>-LT move pointer back one line and display line

EDT>B255[STRS-80$COMPUCOLOR$] move the pointer to the beginning of the buffer. Removes "TRS-80" from the text and replace it with "COMPUCOLOR". Do this 25 times. If a string is replaced more times than it exists, "SEARCH STRING NOT FOUND!" will be displayed when the search reaches the end of the buffer.

EDT>3[255A255W] read in 255 lines and then write 255 lines to output file. Do this 3 times.