CHAPTER 15. TEXT EDITOR

The source code of the editors are in the file UTILITY.BLK, screens 12 to 27.

The Editor is the most often used utility in an operating system to support programming activity. The friendliness of an operating system depends heavily on the editor it provides to the user. Since the source code in Forth is organized around blocks of 1024 bytes and the editor has to deal only with blocks of fixed size, the editor is simpler than the editors in other systems which have to be able to handle large text files, usually of variable length.

The line editor in F83 system is compatible with the editor described in the popular book "Starting FORTH". For details on the various commands in this editor, see the book by Leo Brodie. There are a few extensions, most notably the word NEW which allows you to enter multiple lines of text.

A screen editor is also provided in F83 so that the user always has a full screen showing on his terminal. The description of the screen editor will be in the Chapter 16.

15.1. STRING UTILITY

The string manipulation primitives include string comparison and searching. The string search command is used in the editor to find the desired string. The only unusual feature about this string package is the presence of a variable called CAPS, which determines whether or not to ignore the case of the subject and pattern strings. If case is ignored then A-Z=a-z. The default is to ignore case.

Many string primitives are defined in the kernel, like string compare, lower-to-upper case conversion, etc. Many of them are defined in machine code form to increase execution speed. Here their high level definitions are shown for completeness and for reference. You should consult the sections in the kernel for the code definitions actually used in the system, in the file KERNEL86.BLK, screens 41-43.

```
VARIABLE CAPS                   If true, lower case characters are to be converted to upper case.

: UPC          ( char --- char' )   Convert a character to upper case.
        DUP                         Copy char for comparison.
        ASCII a ASCII z BETWEEN  Is it between a and z?
        IF BL - THEN                If so, convert to upper case by subtracting 32.
        ;

: ?CHAR        ( char --- char' )   Convert a character to upper case if CAPS flag is set.
        CAPS @                      Is CAPS true?
        IF UPC THEN          If so, convert; otherwise skip.
        ;

: COMPARE   ( addr1 addr2 count --- n )   Compare two strings at addr1 and addr2 of equal
                                          length. Case may be significant depending on CAPS.  0 is
                                          returned if the strings are equal.  1 is returned if string at
```

```
                                addr1 is greater than that at addr2.  -1 is returned if the string
                                at addr1 is less than that at addr2.
        >R                      Save the count.
        0                       The initial value of n to be returned.
        -ROT                    Put it under addr1.
        R>                      Retrieve the count.
        0 ?DO                   Scan through the strings.
                OVER I C@       Get a character from string 1.
                ?UPCHAR         Convert it to upper case if needed.
                OVER I C@       Get the corresponding character from string 2.
                        ?UPCHAR Convert.
                - DUP           Are the characters the same?
                IF              No.  The characters are not equal.
                        >R      Save the comparison result.
                        ROT DROP Discard the initial n.
                        R>      Retrieve comparison result.
                        0< IF -1 If it is less than zero, return -1 because string 2 is larger.
                        ELSE 1 THEN     If the comparison is positive, return 1 to indicate that
                                string 1 is greater than string 2.
                        -ROT    Put the result below addr1, replacing the initial n.
                        LEAVE   Quit the do-loop immediately.
                ELSE DROP       Characters are equal.  Discard the result of comparison.
                THEN
        LOOP
        2DROP                   Discard the addresses.
        ;

: INSERT        ( sa sl ba bl --- )     Insert a string at sa into the buffer at ba.  The length of string
                                inserted is the smaller of sl and bl.
        ROT OVER MIN >R         Save the smaller of sl and bl on the return stack.
        R@ -                    bl-sl or 0 if sl>bl.
        OVER DUP                Buffer address ba.
        R@ +                    Address of the remainder of the buffer.
        ROT CMOVE>              Shift the string in the buffer forward, making room for the
                                string to be inserted.
        R>                      Restore the count of insert string.
        CMOVE                   Copy string to buffer.
        ;

: REPLACE       ( sa sl ba bl --- )     Copy a string from sa to ba. Characters copied is the smaller
                                of sl and bl.
        ROT MIN                 Smaller of sl and bl.
        CMOVE                   Copy from sl to ba.
        ;

: DELETE                ( ba bl sl --- )  Delete sl characters from the start of buffer, ba.  Fill the end
                                of buffer with blanks.
        OVER MIN >R             Save the smaller of bl and sl.
```

```
        R@ -                    The remainder of the buffer.
        DUP 0>                  Any character to be move forward in the buffer?
        IF                      Yes.  Copy the remainder of buffer forward to the start of buffer.
                2DUP            Duplicate ba and remainder of buffer.
                SWAP DUP        Buffer address ba.
                R@ +            Address of remainder of buffer.
                -ROT SWAP CMOVE         Copy the remainder of buffer to the beginning of the buffer.
        THEN
        +                       Address of remainder of buffer.
        R> BLANK                Fill the remainder of buffer with blanks.
        ;

VARIABLE FOUND                  A local variable to be used by SEARCH as a flag.

: SEARCH    ( sa sl ba bl --- n f )     Search for the string at sa inside the string at ba.  If found,
                                        return the offset of the found string as n and a true flag.  If
                                        not found, f is false and n is meaningless.
        FOUND OFF               Initialize FOUND to be false.
        OVER >R                 Save buffer address ba.
        ROT TUCK -              bl-sl.
        1+ 0 ?DO                Scan the buffer for the string. Stack is now ( sa ba sl --- )
                3DUP COMPARE    Is the string found at this position?
                0= IF           If found,
                FOUND ON        turn on the FOUND flag,
                LEAVE           and quit the do loop immediately.
                THEN
                SWAP 1+ SWAP    Increment the buffer address ba to do the next comparison.
        LOOP
        DROP NIP                Discard sl and sa.
        R> -                    Offset from the beginning of the buffer to the starting point
                                of the found string.
        FOUND @                 Get the found flag.
        ;
```

These string commands are the basic words used in implementing the string editing commands which are needed in both the line editor and the screen editor in this F83 system.

15.2. TERMINAL DEPENDENT DEFERRED WORDS

Several words which will be used to control screen display in the screen editor are defined here as deferred words so that words defined in the line editor can be used also in the screen editor by re-vectoring these deferred words.

```
DEFER AT    \( col row --- )        Position the cursor at the given location specified by the
                                    stack numbers.
        DOES>                       A vectored word.
        -ROT 2DUP #LINE !           Store col in #LINE.
```

```
        #OUT !                  Store row in #OUT.
        ROT PERFORM             Execute the word vectored to by AT.
        ;
AT                              Execute AT vectors to itself.

DEFER BLOT      ( col --- )     Delete the rest of the current line.

DEFER -LINE     ( --- )         Delete the current line and scroll the rest of screen up by one
                                line.

: DARK          ( --- )         Clear the screen and home the cursor.
        DOES>                   Dark is a deferred word and can be re-vectored.
        PERFORM                 First execute the routine whose execution address was put
                                into the parameter field.
        #LINE OFF               Reset line count.
        #OUT OFF                Reset character count.
        ;
DARK                            Vector DARK to itself.

VOCABULARY EDITOR               Create a new EDITOR vocabulary.
EDITOR ALSO DEFINITIONS         Make it the current vocabulary so that following definitions
                                will be included in it.

DEFER .SCREEN   ( --- )         Display the entire screen.

:  (AT)         ( col row ---)  Do a carriage return in line editor mode.
        2DROP CR    ;

: (BLOT)        ( col --- )     Fill the rest of line with spaces.
        C/L SWAP -              Characters in the rest of this line.
        SPACES                  Output spaces.
        ;

: (DARK)        ( --- )         Clear the screen with line feeds.
        24 0 DO CR LOOP         Send 24 carriage returns.
        ;

' (AT) IS AT                    Initialize AT,
' (BLOT) IS BLOT                and the rest of the deferred words to support the dumbest
                                possible terminal.
' (DARK) IS DARK
' NOOP IS -LINE
' CR IS .SCREEN
```

15.3. THE CURSOR COMMANDS

The cursor in the editor is a pointer pointing to the character position where the next editing actions will occur. The cursor position is stored in a variable R#, as the offset from the beginning of the screen buffer to the address of the current character. All cursor commands use or modify this variable. Often the cursor is represented by a caret '^' on CRT display.

```
VARIABLE R#                     Defined in the nucleus.

: TOP               ( --- )     Go to the top of the screen.
        R# OFF                  Initialize R# to zero.
        ;

: C                 ( n --- )   Move the cursor by n characters, right or left.
        R# @                    Current cursor position.
        C/SCR 1-                1023, a 10 bit mask.
        AND                     Ensure the cursor is within the screen.
        R# !                    Replace it.
        ;
: T                 ( n --- )   Go to the beginning of line n.
        TOP                     Reset R#.
        C/L *                   Beginning of the nth line.
        C                       Set the cursor.  Always within screen.
        ;

: CURSOR            ( --- n )   Return the current cursor position.
        R# @  ;

: LINE#             ( --- n )   Return the current line number.
        CURSOR C/L /            Divide the cursor position by characters per line.
        ;

: COL#              ( --- n )   Return the current column number.
        CURSOR C/L MOD          The modulo of cursor position.
        ;

: +T                ( n --- )   Increment the current line by n.
        LINE# +                 Get the new line number.
        T                       Select it as the current line.
        ;

: 'START            ( --- addr ) The buffer address of the start of the screen.
        SCR @                   The current screen number.
        BLOCK                   The address of the buffer.
        ;

: 'CURSOR           ( --- addr ) The actual address of the current character in the buffer
                                 pointed to by the cursor.
```

```
        'START                  Beginning of the screen.
        CURSOR +                Add cursor offset to get the address in the buffer.
        ;

: 'LINE         ( --- n )       The address of the beginning of the current line.
        'CURSOR                 Address of the cursor.
        COL# -                  Subtract the column number to get back to the beginning.    ;

: #AFTER        ( --- n )       Return the number of characters after the cursor on the current line.
        C/L                     Characters per line.
        COL# -                  Characters after cursor.
        ;

: #REMAINING    ( --- n )       Return the number of characters after the cursor on screen.
        B/BUF                   Characters per screen.
        CURSOR -                Characters after cursor on screen.
        ;

: #END          ( --- n )       Number of characters between the beginning of current line
                                to the end of screen.
        #REMAINING              Characters from cursor to end of screen.
        COL#                    Characters from start of line to cursor on current line.
        +       ;
```

15.4. EDITING BUFFERS

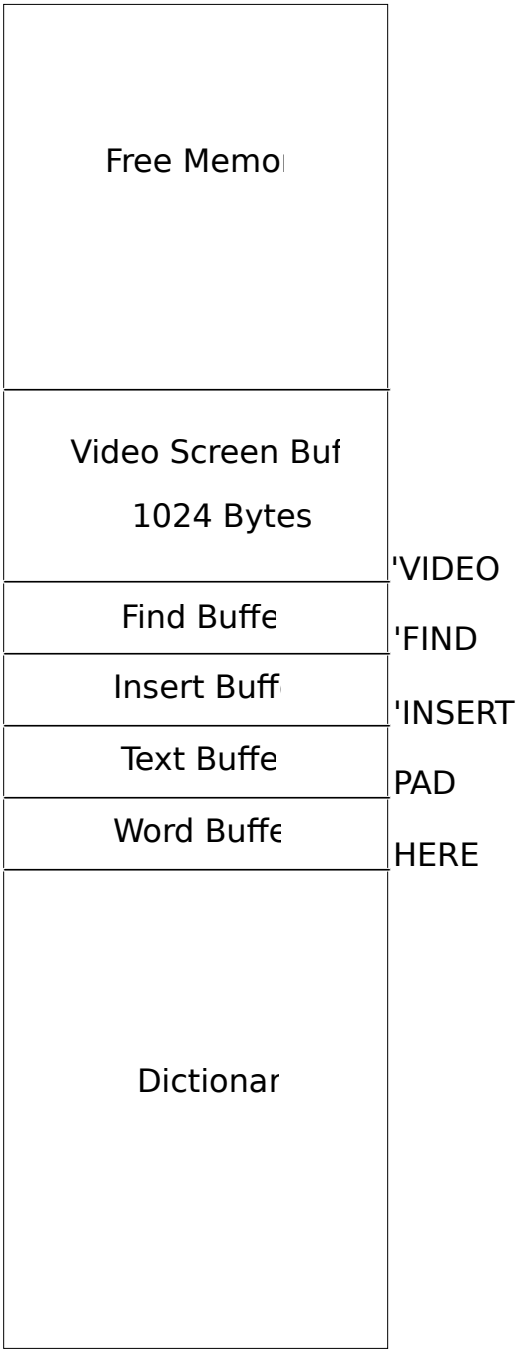
PAD returns the address of a text buffer used by FORTH system for string output and temporary storage. The Starting FORTH editor requires two additional text buffers: an insert buffer and a find buffer. The insert buffer stores a text string which will be inserted into the screen during editing, and the find buffer stores a string to be used in string search. These two buffers are assigned immediately above the PAD buffer and they all float some distance above the top of the dictionary. Since they are using the free memory space between the data stack and the dictionary, they do not require fixed allocation in the RAM memory.

Some of other editing utility words are also defined here.

```
VARIABLE CHANGED        A variable indicating that the current screen has been edited
                        so that date stamp can be applied automatically.

: MODIFIED      ( --- )         Mark the screen as updated and also set the CHANGED flag.
        CHANGED ON              Set CHANGED flag.
        UPDATE                  Set the UPDATE flag.
        ;

ASCII ^ CONSTANT EOS    EOS is the character to denote the end of a string on input. It
                         allows multiple editing commands on one line.
```

```
: ?TEXT         ( addr --- addr+1 n )   Accept a string to addr.  For a null string, do not disturb the
                                        string already at addr.
        >R                              Save addr on return stack.
        EOS PARSE                       Scan the input stream for ^ or end of line.
        DUP                             Get the character count of the input string.
        IF                              If character count is not 0, do the string copying.
                R@                      Retrieve addr.
                C/L 1+ BLANK            Clear one line at addr.
                HERE COUNT              The string is in the word buffer.
                R@ PLACE                Copy the string to addr.
        ELSE 2DROP                      Clean the stack after PARSE.
        THEN
        R>                              Get the addr back.
        COUNT                           Replace it by addr+1 and count.
        ;

10 CONSTANT ID-LEN                      Length of the id stamp buffer.
CREATE ID ID-LEN ALLOT                  Id stamp buffer containing the user name and date stamp.
ID ID-LEN BLANK                         Initialize the id stamp buffer.
84 CONSTANT C/PAD                       All text buffers are to be 84 characters long.

: 'INSERT               ( --- addr )    Return the address of insert buffer.
        PAD C/PAD  +                    84 bytes above PAD.
        ;
```

Figure 15.1 The editing buffers.



```
: 'FIND                 ( --- addr )    Return the address of find buffer.
        'INSERT C/PAD +                 84 bytes above the insert buffer.
        ;

: 'VIDEO                ( --- addr )    Return the screen editor buffer.
        'FIND C/PAD +                   4 bytes above the find buffer.
        ;

: .FRAMED               ( addr --- )    Print a string at addr framed with single quotes.
        ." '"                           Print preceeding quote.
        COUNT TYPE                      Print the string.
        ." '"                           Print following quote.
        ;

: .BUFS                 ( --- )         Display the contents of the insert and find buffers.
        CR ." I "                       Header for insert string.
        'INSERT .FRAMED                 Print insert buffer.
        CR ." F "                       Header of find string.
        'FIND .FRAMED                   Print find buffer.
        ;

: ?MISSING    ( n f --- n, or abort )   If flag is false, print the find buffer and abort.  Otherwise,
                                        return only n.
        0= IF                           If flag is false, do the following.
                DROP                    Discard n.
                'FIND .FRAMED           Print contents of find buffer.
                ."  not found "         This is used when a string cannot be found in the screen.
                QUIT                    Give up and return to the text interpreter.
        THEN ;

: KEEP                  ( --- )         Copy the current line into the insert buffer.
        'LINE                           Address of current line.
        C/L 'INSERT PLACE               Copy the line to insert buffer.
        ;

: K                     ( --- )         Exchange the contents of the insert and find buffers.
        'FIND PAD C/PAD CMOVE           Copy find buffer into PAD buffer.
        'INSERT 'FIND C/PAD CMOVE       Copy insert buffer to find buffer.
        PAD 'INSERT C/PAD CMOVE         Copy old find string to insert buffer.
        ;

: 'F+                   ( n1 --- n2 )   Add the length of the found string to n1.
        'FIND                           Address of the find buffer.
        C@                              Length of find string.
        +       ;

: W                     ( --- )         Abbreviation of SAVE-BUFFERS.
        SAVE-BUFFERS      ;
```

```
: 'C#A  ( --- addr count )      Return the address of the cursor and the characters after
                                cursor on the current line.
        'CURSOR                 Address of the cursor.
        #AFTER                  Characters after cursor.
        MODIFIED                Update flags.
        ;

: (I)   ( --- len 'insert len 'cursor #after)   Get input string into the insert buffer and
                                leave addresses and lengths necessary to do the insertion.
        'INSERT ?TEXT           Get input text and copy it into the insert buffer.
        TUCK                    Tuck a copy of len under address of insert buffer.
        'C#A                    Push cursor address and character count on stack.
        ;
```

15.5. LINE EDITING COMMANDS

Line editing commands modify the contents of the current line in the current screen. Many of these commands expect a string immediately following the command in the same input line. The string may be a null string, i.e., the command is followed immediately by a carriage return. In this case, the contents of the appropriate buffer are used in place of input string. The text string is shown as <text>, which is a sequence of ASCII characters. Blanks or spaces can be included in the string. The string is terminated either by a carriage return or by the carat character '^'.

```
: I             ( --- )     I <text>  inserts text string on the current line at the cursor.
        (I)                 Get the insert string.
        INSERT              Insert it on the current line.
        C                   Move the cursor to the end of the inserted string.
        ;

: O             ( --- )     O <text>  overwrites text string on the current line at the cursor.
        (I)                 Get the insert string.
        REPLACE             Write over the current line with the insert string.
        C                   Move cursor to end of inserted string.
        ;

: P             ( --- )     P <text>  replaces the current line with <text> and blank fill
                            the rest  of the line.
        'INSERT ?TEXT       Get insert string.
        DROP                Discard character count.
        'LINE C/L CMOVE     Copy the entire line.
        MODIFIED            Update flags.
        ;

: U             ( --- )     U <text>  inserts a line under the current line.  Subsequent
                            line in the screen are pushed down by one line.  The last line
                            is lost.
```

```
        C/L C                       Move the cursor to next line.
        'LINE C/L OVER #END INSERT  Insert a dummy line at the next line and push all
                                    subsequent line down.
        P                           Put the insert string at the next line.
        ;

: X             ( --- )             Delete the current line and save it in the insert buffer.
        KEEP                        Save the current line in the insert buffer.
        'LINE #END C/L DELETE       Delete the current line.
        MODIFIED                    Update flags.
        ;

: SPLIT         ( --- )             Break the current line in two at the cursor.
        PAD C/L 2DUP BLANK          Clear the PAD buffer.
        'CURSOR #REMAINING          Address of cursor and characters to end of screen.
        INSERT                      Insert a line of blanks at the cursor.
        MODIFIED    ;

: JOIN          ( --- )             Put a copy of next line after the cursor.
        'LINE C/L +                 Beginning of the next line.
        C/L                         Copy one line,
        'C#A INSERT                 to the cursor.
        MODIFIED    ;

: WIPE          ( --- )             Clear the screen to blanks.
        'START B/BUF BLANK          Fill the screen with blanks.
        MODIFIED    ;

: M             ( m n --- )         M copies the current line to nth line in the mth screen. M is
                                    neutralized because the editor should not affect other screens
                                    and M moves the current line to another screen.
        TRUE ABORT" Use G!"         Always abort.
        ;

: G             ( screen line --- ) Get a line from another screen and insert it in front of the
                                    current line.
        C/L *                       character offset to the line.
        SWAP IN-BLOCK +             Get the source block from the in-file. Add offset to the
                                    source line.
        C/L 'INSERT PLACE First put the source line in the insert buffer.
        C/L NEGATE C                Move the cursor to the line above.
        U                           Insert the source line.
        C/L C                       Move the cursor back.
        ;

: BRING         ( screen first last --- )  Get a range of lines from another screen.
        1+ SWAP DO                  Scan the range of lines.
                DUP                 Source screen number.
```

```
            [ FORTH ] I         Select the loop index in FORTH, not the I defined above in
                                the EDITOR.
            G                   Get one line.
        LOOP DROP               Discard the screen number.
        ;
```

15.6. STRING EDITOR COMMANDS

The main task of the string editor is to locate a string inside the current screen and place the cursor at the end of the found string. This allows the user to modify strings in a screen quickly to make local modifications as he debugs his source code. The string pattern to be searched is put in the find buffer.

```
: FIND?             ( --- n f )     Get the find string from the input stream and search for a
                                    matching string in the screen starting at the current cursor
                                    position. Return the character offset of the found string as n
                                    and a true flag if the string is found. Return a false flag if not
                                    found, and n is meaningless in this case.
        'FIND                       Address of the find buffer.
        ?TEXT                       Get the input text into find buffer.
        'CURSOR #REMAINING SEARCH   Search the screen from the cursor down to
                                    find a match.
        ;

: F                 ( --- )         F <text>  finds the text and leaves the cursor just pass it.
        FIND?                       Get the find string and do the searching.
        ?MISSING                    Quit with an error message "?" if string is not found.
        'F+ C                       Move cursor to the end of the found string.
        ;

: E                 ( --- )         E <text>  erases the string just found.
        'FIND C@                    The character count of the found string.
        DUP NEGATE C                Move the cursor to the beginning of the found string.
        'C#A ROT DELETE             Delete the found string.
        ;

: D                 ( --- )         D <text>  finds and deletes a string.
        F                           Find.
        E                           Erase.
        ;

: R                 ( --- )         R <text>  replaces the text string just found with the string in
                                    the insert buffer.
        E                           Erase the found string.
        I                           Insert the insert string.
        ;
```

```
: S                    ( n --- )    n S <text> searches for the text through all screens from the
                                    current up to screen n.  Each time a match is found, n
                                    remains on the stack until screen n is reached.
        1 ?ENOUGH                   Abort if the data stack does not have at least one item on it.
        FIND?                       Search the current screen.
        IF 'F+ C EXIT THEN          Found in current screen.  Move cursor and quit.
        DROP                        Discard dummy number on stack when string is not found.
        FALSE                       Put a false flag on stack for do loop scanning.
        OVER SCR @ DO               Scan a range of screens.
                N                   Next screen.
                TOP                 Beginning of next screen.
                'FIND COUNT         Find string.
                'CURSOR #REMAINING  Buffer address and count in the next screen buffer.
                SEARCH              Search the text string.
                IF                  Found the string.
                'F+ C               Move cursor to end of found string.
                DROP TRUE           Replace false flag with true.
                LEAVE               Exit the do loop.
                ELSE DROP           Discard the character offset if string is not found.
                THEN
        KEY? ABORT" Break!"         If any key is received on the keyboard, quit the searching.
        LOOP                        Otherwise, continuing the search to the next screen.
        ?MISSING                    n should be on the stack.
        ;

: (TILL)               ( --- )      Search in the current line for the text string.
        'FIND ?TEXT                 Get the text string and put it in the find buffer.
        'C#A SEARCH                 Search current line from the cursor for the find string.
        ?MISSING                    If string can"t be found, abort.
        ;

: TILL                 ( --- )      TILL <text>  deletes all text on the current line from cursor
                                    to the end of the find string.
        'C#A (TILL)                 Search the find string.
        'F+ DELETE                  Delete from cursor to end of found string.
        ;


:  JUST                ( --- )      Justify.  Delete up to but not including the text string.
        'C&A (TILL)                 Find the text string.
        DELETE                      Delete all characters between the cursor and the starting
                                    character of the found string.
        ;

:  KT          ( --- )              Keep-Till.  Copy all the characters between the cursor and
                                    the end of the text string into the insert buffer.
        'CURSOR (TILL)              Find the text string.
        'F+                         The end of string.
```

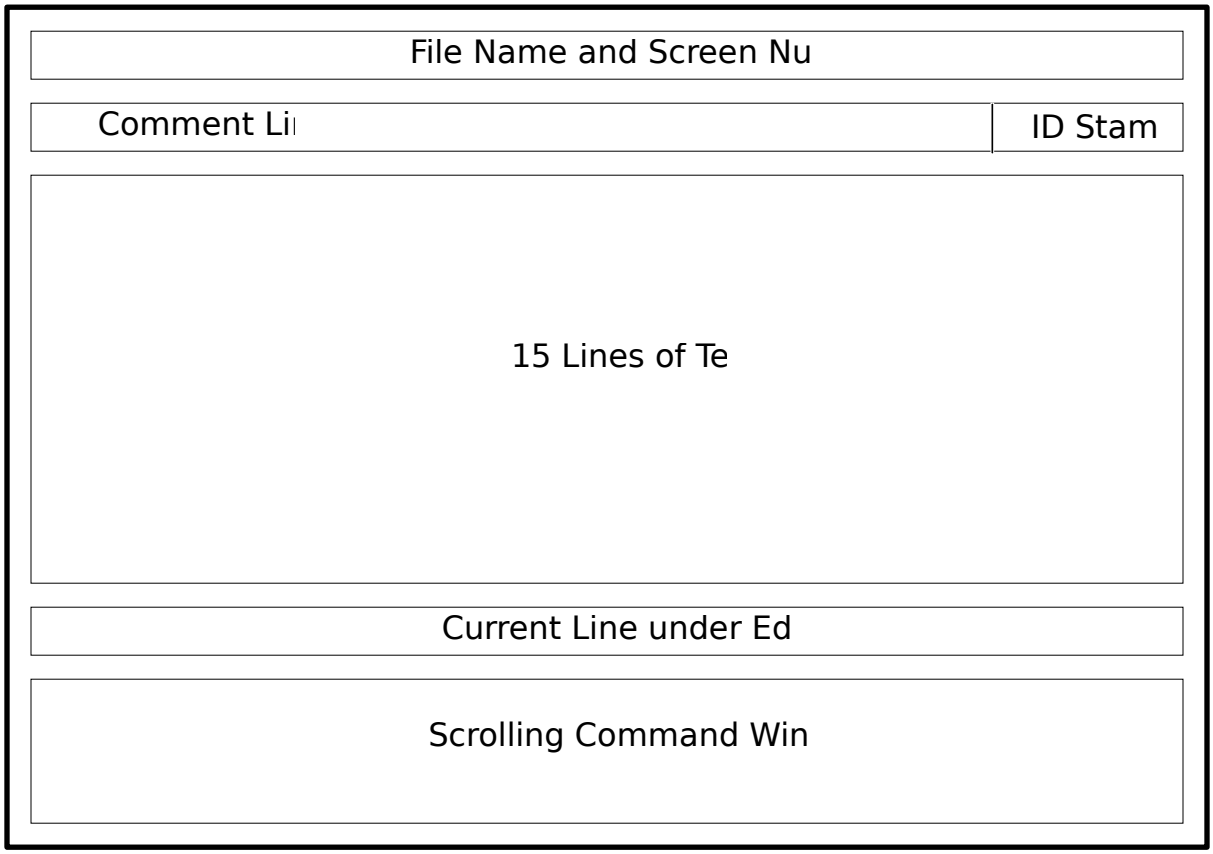
```
'INSERT PLACE          Copy the characters into insert buffer.
;
```

15.7. SCREEN EDITOR

The line editor works on the source screen one line at a time. It serves well all the editing functions. The only problem is that the text screen displayed on the terminal scrolls up with the entering of new lines at the bottom of terminal screen. After some editing, compiling and testing, the text screen starts to disappear over the top of the terminal and the user must type L command to re-display the text again. As the terminals getting smarter and smarter, it is nice if we can use some of the extra functions in the terminal to keep the text screen at the top of the terminal all the time. This is basically what a screen editor does. Any time the text screen is modified, the modification will be written on the displayed text screen immediately.

If all the terminals were built the same way, it would be a simple exercise to write a screen editor. However, terminals are built with different screen control commands. The screen editor must be tailored to the specific terminal to use its specific cursor and display control commands. The screen editor in F83 uses all the words developed in the line editor for editing functions. The terminal-specific functions are concentrated in four words: AT, DARK, BLOT, and -LINE, which manage a continuous display of the text screen being edited. The display is updated automatically as each command line is executed.

Figure 15.2 Screen editor display



15.8. THE SCREEN DISPLAY COMMANDS

The display on the terminal is assumed to have a 24 by 80 character format. The screen editor displays the screen number on the top line or line 0 on the screen display. Lines 1 to 16 are used to display 16 lines of text in the current screen. Line 17 display the current line. Lines 18 to 23 are used as a scrolling screen for command input and character output. The text screen stays at the top of display and always shows the updated text of the current screen under editing.

```
3 CONSTANT DX                   Column offset for screen text. Allow room for line numbers.
1 CONSTANT DY                   Row offset for screen text.  Allow room for screen number.

: .LINE             ( --- )     Display the current line, with the cursor shown as an up-
                                arrow or caret.
        LINE# 2 .R SPACE        Display the current line number.
        'LINE COL# >TYPE        Display the text before cursor.
        ASCII ^ EMIT            Display ^, current position of the cursor.
        'CURSOR #AFTER >TYPE    Display the text after the cursor.
        ;

: REDISPLAY( line# --- )         Update the image of line n on the terminal.
        0 OVER DY +              Column and row of line n on the display.
        AT                       Move the display cursor to the display coordinates.
        DUP 2 .R SPACE           Print the line number.
        DUP C/L *                Character offset of line n.
        'START +                 Address of line n in screen buffer.
        C/L TYPE                 Display the entire line.
        SPACE .                  Display the line number.
        #OUT @ BLOT              Erase the rest of the line.
        ;

: CHANGED?( line# --- f )        Return a true flag if the line has changed since last display.
                                 It is sensitive to case changes.
        C/L *                    Character offset to the line.
        DUP 'START +             Address of the line in the buffer.
        SWAP 'VIDEO +            Address of same line in the video buffer.
        C/L COMP                 Compare the lines in text screen and video buffer.  Return
        ;                        true if two line are different.

: .ALL          ( --- )          Redisplay all lines which have changed, the screen number,
                                 the cursor line, and scroll the command region.
        DISK-ERROR @ 0=          If no disk error, display the screen.
        IF
                DX 0 AT .SCR     Display the screen number.
                #OUT @ BLOT      Erase the rest of the line.
                [ FORTH ]        Switch context to FORTH because
                ?STAMP           Stamp the screen if not done.
                L/SCR 0 DO       Scan all the lines in the screen.
                        I CHANGED?       Has this line been changed?
```

```
                IF I REDISPLAY      If changed, redisplay the new line.
                THEN
            LOOP
            'START 'VIDEO B/BUF CMOVE     Update the video buffer.
            0 18 AT .LINE           Display the current line under the displayed screen.
            0 19 AT -LINE           Delete line 18 on display and scroll up the rest of screen display.
            0 23 AT                 Position the display cursor at the bottom of display,
                                    assuming 24 line display.
            #OUT OFF                Clear output character count.
        THEN
        ;

: EDIT-AT           ( --- )         Move the diplay cursor to show the position of the editor
                                    cursor for editing functions.
        CURSOR C/L /MOD             Convert cursor offset to screen coordinates.
        SWAP DX +                   Column number.
        SWAP DY +                   Row number.
        AT                          Move the display cursor.
        ;

: NEW               ( n --- )       Move the display cursor to the beginning of line n and accept
                                    text for following lines until a null line (a line begins with
                                    a carriage return) is entered, i.e., 2 CR's gets you out of NEW.
        L/SCR SWAP DO               Scan from line n down.
            [ FORTH ] I             Loop index.
            [ EDITOR ] T            Position editor's cursor.
            EDIT-AT                 Position the display cursor.
            >IN OFF                 Reset the input character offset.
            QUERY                   Wait for a line of input text.
            SPAN @                  Number of characters in the input text.
            IF                      If not a null line,
                P                   Put the text in the current line.
            ELSE                    For a null line,
                [ FORTH ] I         Get the current line number,
                REDISPLAY           Put back the old line.
                LEAVE               Quit here.
            THEN
            .SCREEN                 Refresh the display.
        LOOP
        .SCREEN
        ;

: GET-ID            ( --- )         Check the ID stamp field.  If it is empty, prompt for user's
                                    id and date.
        ID ID-LENGTH                Get the ID string address and length.
        -TRAILING NIP 0=            Is the length 0?
        IF                          Yes.  Prompt the user.
            CR ." Enter you ID: "
```

```
            ID-LEN 0 DO          Display a string of dots.
                ASCII . EMIT
            LOOP
            ID-LEN BACKSPACES    Backspace over the dots.
            ID ID-LEN EXPECT     Input the id stamp to the id stamp buffer.
        THEN
        ;

: STAMP             ( --- )      Put the stamp at the end of line 0 in the current screen.
        ID                       Id buffer address.
        'START C/L +             End of line 0.
        ID-LEN 1- -              Backup the length of stamp text.
        ID-LEN 1-
        CMOVE                    Copy the stamp to screen.
        ;

: ?STAMP            ( --- )      Update the ID if the screen has changed and clear the change
                                 flag.
        CHANGED @ IF             Changed?
            STAMP                Stamp the screen.
            CHANGED OFF          Reset changed flag.
        THEN  ;

2VARIABLE AUTO                   Addresses of CR and STATUS to patch vectors for CRT
                                 and TTY, respectively.
VARIABLE EDITING?                Set during editing.

VARIABLE CHANGED                 Set if the edited screen has been changed.

: INSTALL           ( --- )      Initialize the screen editor.
        EDITING? @ NOT IF        If not in the editing mode, initialize screen editor.
            ['] .SCREEN          Address of the screen refresher.
            AUTO @ !             Vector it through AUTO.
            EDITING? ON          Turn on editing flag.
            CHANGED OFF          Turn off changed flag.
        THEN
        DISK-ERROR OFF           Turn off the disk error flag always.
        ;
```

15.9. THE SCREEN EDITOR COMMANDS

```
FORTH DEFINITIONS           The following screen editing commands should be made
                            accessible from the common FORTH vocabulary.

: DONE              ( --- ) Normal exit from the screen editor. Update the id stamp, tell
                            you if the screen was modified, flush the screen to disk file,
                            and remove automatic screen refresh.
        [ EDITOR ]
        EDITING? @ IF       If still in the editing mode,
                PREVIOUS
                EDITING? OFF    turn off editing flag,
                CR  SCR ?       type the screen number,
                >UPDATE @ 0< NOT IF     and look at the update field.
                ." Un" THEN     Not updated.  Print prefix.
                ." modified"    Complete the message.
                ?STAMP          Update the ID stamp.
                W               Save contents to disk file.
        THEN
        DISK-ERROR OFF      Turn of disk error flag.
        AUTO 2@ !           Re-vector CR to normal scrolling mode.
        ;

: ED            ( --- )     Re-enter the screen editor.  Clear and initialize the display
                            and begin automatic screen refresh.
        [ EDITOR ] GET-ID   Get id stamp.
        INSTALL             Initialize the screen editor.
        EDITOR              Make EDITOR the context vocabulary.
        'VIDEO B/BUF ERASE  Clear the video buffer.
        DARK                Home the cursor and clear CRT.
        .ALL                Print the screen on CRT.
        ;

: EDIT          ( n --- )   Set n as the current editing screen and call ED to do screen editing.
        1 ?ENOUGH           Abort if the stack has less than one item.
        SCR !               Store n in SCR, making screen n the current editing screen.
        [ EDITOR ] TOP      Move editor cursor to top of screen.
        ED                  Do screen editing.
        ;

: (WHERE)   ( pos scr --- ) When an error occurred during compilation, the position and
                            screen number where error occurred are saved on the data
                            stack. (WHERE) uses these two numbers to invoke the
                            screen editor and show the screen to be edited.
        DISK-ERROR @ 0=     Make sure the error is not caused by disk reading.
        IF
                EDIT            Do screen editing.
                [ EDITOR ] 1- C Position the display cursor before the error.
                'WORD COUNT 'FIND PLACE
```

```
                                Put the word in trouble into the find buffer.
    THEN  ;
```

15.10. CONFIGURING THE TERMINAL

Each terminal manufacturer has its own way of controlling the display on the terminal screen. To use the full power of a screen editor, the screen editor must know how to position the display cursor at any position on the display screen and a few other things like initialize the display, erase one line and delete one line. In F83 system, terminal configuration commands are provided for a number of popular terminals. If your terminal is not in this list, you will have to define the proper commands. You can follow the pattern as provided. It is not a very difficult task.

An example is shown here. You should consult the F83 listing for other terminals.

```
: SMART             ( --- )     Initialize vectored routines for the IBM PC.
      ['] CRLF                  Execution address of carriage return.
      ['] CR >BODY              Vector address in the deferred word CR.
      AUTO 2!                   Store them in the AUTO area.
      ['] .ALL IS .SCREEN       Vector .SCREEN to .ALL.
      ;

CODE IBM-AT       ( col row --- )  Position cursor at the specified location on CRT screen.
      AX POP DX POP AL DH MOV      Copy col and row into DX.
      BH BH XOR                   Clear BH register.
      2 # AH MOV                  Cursor positioning code.
      16 INT                      Call BIOS.
      NEXT
      END-CODE

CODE IBM-DARK   ( --- )       Clear screen and home the cursor.
      2 # AX MOV              Home code.
      16 INT                  Call BIOS.
      NEXT
      END-CODE

CODE IBM-BLOT   ( col --- )   Clear from cursor to end of line.
      80 SWAP -               Remaining characters on the line.
      SPACES                  Output that many spaces.
      ;

CODE IBM--LINE  ( --- )       Delete the current line and scroll the rest of the screen.
      BP PUSH                 Save BP register.
      BH BH XOR               Clear BH.
      3 # AH MOV  16 INT      Call BIOS for cursor position.
      DH CH MOV               Line number moved to CH.
      CL CL XOR               Column number cleared to zero.
      24 256 * 79 +           Bottom line and right-most column
```

```
        # DX MOV                  copied to DX register.
        7 # BH MOV
        6 256 * 1 + # AX MOV      Code stored in AX.
        16 INT                    Call BIOS.
        BP POP                    Restore BP.
        NEXT
        END-CODE

: IBM                ( --- )      Initialize the screen editor for IBM PC.
        SMART                     Vector CR and .SCREEN.
        ['] IBM-AT IS AT          Vector the rest of terminal specific commands.
        ['] IBM-DARK IS DARK
        ['] IBM--LINE IS -LINE
        ['] IBM-BLOT IS BLOT
        ;
```