```
================================================================================
00/09/14 16:47:51 FILE: VEINDEX    TEXT   4096 09/14/00 16:47
================================================================================
```

```
00      00    0000000000                  0000000000      00
00      00    0000000000                  0000000000     000
00      00    00                                  00    0000
00      00    00                                 000   00000
00      00    00                                000       00
000     00    00                               000        00
 000   000    00000000      0000000000        000         00
  00  00      00000000      0000000000       000          00
  00  00      00                            000           00
  00  00      00                            00            00
   0000       00                            00            00
    00        00                            00            00
    00        0000000000                    00       00000000
    00        0000000000                    00       00000000
```

```
             0   0   000   00000  0000     0     000
             0   0  0   0  0      0   0    0    0   0
             0   0  0      0      0   0    0    0
             0   0   000   0000   0000           000
             0   0      0  0      0 0               0
             0   0  0   0  0      0  0          0   0
              000    000   00000  0   0          000
```

```
             0   0   000   0   0  0   0   000   0
             00 00  0   0  0   0  0   0  0   0  0
             0 0 0  0   0  00  0  0   0  0   0  0
             0 0 0  00000  0 0 0  0   0  00000  0
             0   0  0   0  0  00  0   0  0   0  0
             0   0  0   0  0   0  0   0  0   0  0
             0   0  0   0  0   0   000   0   0  00000
```

I N D E X

SECTION PAGE

V E - The Visual Editor for the HP71

VE is a software package for the HP71 portable computer designed by members of CHHU-ITaly.

The package comprises the following files:

- VE71	BASIC main program.
- VEDIT	BASIC bootstrap program.
- VELEX	LEX run-time support file.
- VEDB	DATA base file, (run-time screen dependant information).
- VEFOLD	BASIC (optional) run-time preprocessor program.
- MKVEDB	BASIC (optional) support program, (MaKe VEDB).
- MKVEKEYS	BASIC (optional) support program, (MaKe VE KEYS).
- KEYDATA	BASIC sample key definitions for use with MKVEKEYS.

Distribution Notice

We [the Authors] hereby donate all material included on the distribution disc (the above mentioned files and all relative documentation files) to the CHHU HP71 ROM Project Committee for inclusion in the ROM and/or in its User's Manual. However, we ask the CHHU ROM Project Committee not to distribute this material outside the ROM project. If the Committee were not interested in the present package, we ask it be returned as is.

CHHU Chapters and CHHU affiliated clubs are hereby given permission to publish the present document only.

Hardware and Software Requirements

HP71B with ca. 17 Kbytes of RAM for programs and run time variable allocation + any additional RAM required by user's text file; HPIL module; HPIL video interface and monitor; following LEX files: STRINGLX (HP), CUSTUTIL (HP), EDLEX (HP) and FKEYLEX (JPC-SIG).

Notes:

The above mentioned RAM requirement is calculated with EDLEX residing in MAIN RAM. EDLEX can be found in some ROMS made by HP (e.g the FORTH-ASSEMLER ROM). If you have any of these ROMS, EDLEX need not be loaded into RAM, and thus you can save 2557 bytes.

FKEYLEX is a pubblic domain LEX file for non-commercial use only by CHHU, CHHU Chapters and CHHU affiliated clubs. If you do not have FKEYLEX, you can request a copy of it from the French CHHU Chapter:

    Saturn Iterest Group
c/o Pierre David
    33 Bd. St. Martin
    75003 PARIS
    France

or from the coordinator of CHHU's international SWAP programme:

Michael Markov
P.O. Box 17
Lockwood
NY 14859, U.S.A.

Features

VE is a full screen text editor designed to be operatable by any HPIL video interface compatible with the escape sequences of the HP82163 32-column video interface device. This means that VE can be used by almost all video interfaces currently available on the market, for example: the HP92198, the PACSCREEN, the GRABAU and the MICRONIX.

VE can be used in two modes: as a real full screen editor that takes advantage of the full screen size available whatever this might be, and as a screen oriented line editor capable of handling a "virtual" screen wider than the real screen. In this second mode, for example, it is possible to edit 80 column lines on a 32 column video interface, with all the convenience and powerful features of VE still available.

VE is easy to operate. All standard editing keys on the HP71 keyboard have their usual meaning, so it is possible to start using VE immediately, without having to learn strange key sequences. Commands are issued by pressing the [f]-prefix key followed by a single alphabetical key. In this way all comands are identified by a single easy-to-remember alphabetical character.

VE responds to the following commands: AUTOMATIC, BACK, COPY, DELETE, ERASE, FIND, GOTO, HIGHLIGHT, INSERT, JOIN, LINE, MOVE, NEXT, OPEN, POSITION, QUERY, REPLACE, SELECT, TAB, VIEW, WORD and YANK, which are all described in detail in this document.

VE gives the user extensive control over cursor movements: left, right, up, down, start/end of file, start/end of current line; next/previous word on current line; next/previous user specified character on current line. Furthermore it is possible to POSITION the cursor to an absolute numbered line; to a line given by a user specified expression relative to the current line, the start/end of file or the start/end of a marked block of lines.

VE allows the user to define tabstops at his convenience. Tabstops combined with an automatic indent mode are invaluable for writing program source files in FORTH or ASSEMBLER. There is also an AUTOMATIC wrap around mode, which is very useful when writing ordinary documents.

VE allows the user to OPEN a new line in a text file, as well as to JOIN two lines into a single line. It is possible to delete a single LINE or a single WORD.

VE offers complete cut-and-paste functions. It is possibile to mark or SELECT a block of text lines, MOVE it around in the text file, make a COPY of it, DELETE it, save (YANK) it in an external buffer/file (that can even reside on a mass memory device), and retrive (INSERT) it elsewhere or in another file from the external buffer/file.

VE offers complete FIND, REPLACE and QUERY functions, with a complete set of "metacharacters" (wild cards).

During a VE editing session it is possible to invoke the BASIC environment, and thus to perform a calculation or call a program, (provided the program does not trash VE's environment or change the DISPLAY and PRINTER assignments, or the settings of flags 0, 1, 2, 3, 4, and -1), and then return to VE.

VE allows for USER defined key definitions for typing aid, just like in the BASIC environment. Furthermore, these key definitions can contain VE commands, with or without parameters and/or user prompts. In this way VE offers a MACRO ability, this means that VE is a PROGRAMMABLE text editor. For example, a macro definition can: copy the work file to mass memory and end editing session; delete a block of a given number of lines starting from the current line; build a standard template or header for letters and documents; clear the whole work file; find a predifined pattern and merge an external file into the work file.

VE is line-oriented, in the sense that a line is the basic unit of information. There are not such concepts as "paragraphs" or "sections" of text. VE is a screen editor, not a word processor, albeit it is probably the most powerful text handling tool available to HP71 users.

Getting Started

Before you start using VE, there are a few things you should know and do. As mentioned in the preceding section, VE is a full screen text editor operatable by any HPIL video interface compatible with the escape sequences of the HP82163 32 column video interface device. "Compatible" refers to escape sequences only, not to screen size. This implies that VE has to be told the exact screen size, as well as few other parameters.

Since this information is seldom changed, most parameters are stored as constant values in a DATA file named VEDB (Visual Editor Data Base). The VEDB file must reside in HP71 memory when VE is called. The VEDB file that comes on the package disc contains default values suited for the HP82163 32 column video interface. You should be able to use the given VEDB file even on larger (80*24) screen size video interfaces without any modification. The VEDB file contains all escape sequences used by VE to drive the video interface. If your video interface does not respond to the escape sequences used by the HP82163 video interface, then you must modify the VEDB DATA file by running a set up program called MKVEDB (Make VE Data Base). As the name implies, this programs allows you to define the contents of the VEDB file. The use of MKVEDB is explained in Appendix A. (Furthermore, the VEDB file contains a special string used to store "invisible" characters: these are explained under the ERASE command).

To invoke the VE program you must type a command line with the following syntax:

```
CALL VE(<#S.ROWS>,<#S.COLS>,<#W.ROWS>,<#W.COLS>,<DEVICE>,<TABS>)
```

where the parameters have the following meaning:

#S.ROWS	real number of rows on screen,
#S.COLS	real number of columns on screen,
#W.ROWS	virtual number of rows, (i.e. rows in window),
#W.COLS	virtual number of columns, (i.e. columns in window),
DEVICE	string specifying the display device to be used,
TABS	string specifying the position of tab stops.

This command line is not easy to remember, so you will find it convenient to assign it to a user defined key or to use a little program like the VEDIT program that comes on the distribution disc, and then invoke VE with CALL VEDIT or RUN VEDIT. If you examine VEDIT, you will notice it contains the following program line:

```
100 CALL VE(32,16,32,16,"%48","4") IN VE71
```

which contains default values suited for the HP82163 32-column video interface. You should be able to use the given VEDIT file even on larger (80*24) screen size video interfaces without any modification, although you will only use a portion (32*16) of the full screen size available. However if you wish to exploit full screen size offered by these interfaces, you only need to modify the parameters passed over to VE. When you modify these parameters you must keep in mind the following:

The parameters #S.ROWS and #S.COLS refer to the exact screen dimensions, whilst #W.ROWS and #W.COLS refer to the dimension of the "virtual" screen window you wish to use. The relation #W.ROWS <= #S.ROWS must always be true. If #W.COLS <= #S.COLS then #W.ROWS and #W.COLS simply define a portion, i.e. a window, of the real screen that will be used by the text editor. However, if #W.COLS > #S.COLS then VE will automatically switch to it's second mode of operation, acting as a screen oriented line editor. In all cases the window used for text output to the screen will be divided into two parts: the work area and the message line. The message line is always the last line of the window, and the work area is the whole window less the message line.

The parameter DEVICE is a string indicating which HPIL device on the loop will be used as a display.

The parameter TABS is a string indicating the current settings of tab stop positions. There are two ways to specify this parameter. The TABS string can contain a numeric expression, and in this case the value of the expression will give the number columns between one tab stop and the next one, starting from the first tab stop which is always positioned in the first available screen column. For example, if TABS is the string "2+3", then VE will set a tab stop every fifth column.

If the TABS string does not contain a numeric expression, then successive tab stops after the first one will be set at the columns given by the ASCII range plus one determined by the first character in TABS and the following characters. For example, if TABS is set to "AHOX". This means that the first tab stop will be set (by default) at column 1; the second tab stop will be set at column 1+NUM("H")-NUM("A")=8; the third tab stop will be set at column 1+NUM("O")-NUM("A")=15 and the fourth tab stop will be set at column 1+NUM("X")-NUM("A")=24. (Note that the tab settings given in this example are suited for writing ASSEMBLER source files).

There is yet another setup feature worth to mention in this section, although it is explaind in more detail in Appendix B. VE has the capability of invoking automatically a preprocessor program called VEFOLD. This program was designed in order to "fold" all the lines in a text file that happen to be longer than the current width of the screen window. It is essential that every text file you edit with VE does not contain lines that are longer than the current screen window width. If you use VE for the first time, you will not be concerned by this problem, (so you will not have to load the VEFOLD file into the HP71). You will never have to bother about this problem as long as you do not decrease the screen window width to edit a file which was created with a larger screen window width. Just remember that the VEFOLD program is available if you need it.

Using VE

Provided that you have all necessary hardware, you can start using VE after you have loaded the following files into MAIN RAM: VE71, VEDIT, VELEX, VEDB, FKEYLEX, STRINGLX, CUSTUTIL and EDLEX. (Remember that EDLEX can also be found in a ROM. Check this out, and if it is the case, then you do not need to load EDLEX).

When VE is called, for example by running the VEDIT program, it will ask for the name of the text file you wish to edit. (If you wish to abort the editing session at this point, simply type an illegal file name like an asterisk followed by [ENDLINE] and you will be back to the BASIC environment). Type the name, press [ENDLINE] and VE will be ready to let you edit the named file. If the file does not exist, it will be created.

From this point on, using VE is very straigth forward. You may use the alpha-numeric keyboard to write any text you wish, and the standard HP71 editing keys (BACK, -CHAR, I/R, LC, -LINE) to perform the most common editing functions.

VE is based on a WYSIWYG - What You See Is What You Get - philosophy, (although "invisible" characters make it slightly difficult to adhere to such a philosophy completely, but more about this under the ERASE command). This means that you can use the four cursor keys to move around anywhere on the screen, and any changes or additions you make at the cursor position will also be reflected in the text file. The four cursor keys, however, will not let you go beyond the left or right margin of the screen window, as well as the start or end of file. If the cursor is at any of these positions and you try to surpass them, VE will emit an alerting beep. This will give you a precise feedback of where you are in the file. Try it!

The cursor keys preceded by the g-prefix key have their usual meaning. That is: [g] [LEFT arrow] and [g] [RIGHT arrow] will position the cursor, respectively, at the begining and at the end of the current line. [g] [UP arrow] and [g] [DOWN arrow] will position the cursor, respectively, at the start and at the end of the text file you are working with. You can learn more about cursor positioning under the POSITION command.

The key sequences [f] [left parenthesis] and [f] [right parenthesis] may be used to position the cursor, respectively, on the next and on the previous word of the current line. Note that these two functions have a "circular" behavior; i.e. if you try to go to the previous (next) word, but the cursor is already on the first (last) word of the line, then the cursor will jump around to the word at the other end of the current line. This is very convenient, as it allows you to chose the shortest path to get to a particular word.

If your text file contains more text than can be displayed with a single screen shot, then you may page through the text. The [f] [ENDLINE] and [g] [ENDLINE] key sequences will display, respectively, the previous and the next screen of text.

You can obviously use both upper case as well as lower case characters, and you can toggle between the two cases by using the standard HP71 editing key sequences: the [g]-prefix key to toggle the case of the character typed next, or [f] [L/C] to toggle the current case mode. You can tell the current state of Lower Case mode by looking at flag annunciator 4 of the HP71's LCD display. This annunciator will be visible whenever Lower Case mode is enabled (on).

If you wish to change the current work file without getting out of the editor, you can "restart" the editor by pressing [g] [ON]. When you have finished, press the [ATTN] key to terminate the editing session and return to the BASIC environment.

V E   C O M M A N D   D I C T I O N A R Y

The following "Command Dictionary" provides a complete description of each command implemented in the VE full screen editor. The Command Dictionary is organized so that you can use it both as an instruction book and as a reference tool. All commands are listed in alphabetical order, and all information relevant to a command, including examples and details, is given under that command's entry in the dictionary.

For any command you wish to issue to VE, remember the general rule that all commands are issued by pressing the [f]-prefix key followed by an alphabetical key, this key being the first letter in the command's name. For example, if you wish to issue the AUTOMATIC command, you just have to press the [f] key followed by the [A] key. Some commands may require you to type some additional information too. For example, if you issue the FIND command, you will obviuosly need to provide for a search pattern. The command stack is active most of the time you need to type any additional information; this can be handy if you want to repeat the same response several times or if you just have to give a slightly different response. Any additional information you need to type to issue a given command is described under that command's dictionary entry, along with any special rules required to do that.

*****   A   -   A U T O M A T I C   *****

The AUTOMATIC command refers to the automatic wrap around mode wich is initially enabled (on) when you start the VE program. The AUTOMATIC command will alternatively toggle the automatic wrap around mode off and on. You can tell the current state of the automatic wrap around mode by looking at flag annunciator 0 of the LCD display. This annunciator will be visible whenever the automatic wrap around mode is disabled (off).

Note that the behaviour of the wrap around mode is affected also by the state of the insert or replace mode. You may toggle between insert and replace mode, and viceversa, be pressing the standard HP71 editing key sequence [f] [I/R]. Replace mode will be enabled automatically every time VE starts. You can tell the current state of the insert/replace mode by looking at flag annunciator 1 of the LCD display: this flag will be visible whenever insert mode is enabled. The insert/replace mode is also reflected by the two different types of cursor that you will see on the screen, although this is not always true, depending on the particular video interface at hand. (The PACSREEN, for example, has a bug that will mess up the two different cursors: this explains why we have decided to show the real state of the insert/replace mode with flag annunciator 1 too).

Wrap around mode and insert/replace mode will cause the following four different actions depending on the respective states:

1) Wrap around mode disabled and replace mode enabled: replace cursor

with next alphanumeric key, and if not at right margin then advance cursor else emit a warning chirp.

2) Wrap around mode enabled and replace mode enabled: just like previous case, but if at right margin then do wrap around.

3) Wrap around mode disabled and insert mode enabled: if there is room left on the current line, then insert next alphanumeric key under cursor, else emit a warning chirp.

4) Wrap around mode enabled and insert mode enabled: just like previous case, but if at right margin then do wrap around in inset mode.

Note that even the backspace command, (whose standard HP71 editing key sequence is [f] [BACK]), has two different actions in insert and in replace mode, although this is nothing new to HP71 users, as the two different actions take place even with the HP71's standard LCD editor.

Another feature to mention is that wrap-around mode will preserve the indentation of the last typed line. The indentation will also be preserved when you end a line with [ENDLINE] and this line is the last line of the file. This is very useful when writing program source files or, in general, when writing out tabular information.

***** B - B A C K *****

The BACK command, (not to be confused with the backspace command mentioned in the preceding section), refers to a particular cursor movement that can take place on the current line. As soon as you have issued the BACK command by pressing [f] [B], the HP71 will wait for you to press another key. If this key is an alphanumeric key and the corresponding character occurs to the left of the current cursor position, then the cursor will be positioned at the first occurence of the specified character that happens to the left of the current position. If the given character does not occur before the current cursor position, then no action takes place. If another command is issued instead of the pressing of the expected character key, then that command will be executed.

***** C - C O P Y *****

The COPY command allows you to copy a block of text from one place to another within the work file. Before you issue the COPY command, you must mark the block of text you wish to copy by issuing the SELECT command (see the SELECT command's dictionary entry for more details). If you inadvertedly issue the COPY command and no block of text has been marked, then VE will display the error message "Missing mark(s)". Once you have marked a block of text, all you have to do is to position the cursor on the line above which you whish to copy the marked block of text and then press [f] [C]. Note that the destination position may not reside within the marked block of text. In this case the COPY command will take no action but display the error message "Inside block".

***** D - D E L E T E *****

The DELETE command allows you to delete a block of text. Before you issue the DELETE command, you must mark the block of text you wish to delete by issuing the SELECT command (see the SELECT command's dictionary entry for more details). If you inadvertedly issue the DELETE command and no block of text has been marked, then VE will display the error message "Missing mark(s)". Once you have marked a block of text, all you have to do is to press [f] [D]. VE will then ask you to confirm your action with the prompt "Delete? Y/N/Q". If you press [N] (for No) or [Q] (for Quit), then the DELETE command will be aborted. If you press [Y] (for Yes), then the deletion will take place, and a message will be displayed to let you know how many lines are being deleted. Note that when this prompt appears, no other keys except [Y], [N] and [Q] are active. Be careful when using the DELETE command: once you have deleted a block of text there is no way to get it back again. This is why the DELETE command asks you for confirmation, even though you may find this unecessary or annoying.

***** E - E R A S E *****

The ERASE command may seem a bit tricky, and it is - so if you are a first time user you may ignore this command altogether. The purpose of this command is to erase "invisible" characters and trailing blanks from the work file.

"Invisible" characters are invisible in the sense that the video interface at hand is not able to display them on the screen. Usually - with normal use of VE - you will never encounter invisible characters, although, on some occasions, it might happen you have to deal with these characters, (for example, when writing particular program source files). If this is the case you will notice that things get "messed up" on the screen; for example, the cursor won't be positioned correctly. This happens because VE "knows" that a certain number of characters exists in a given line of text; if some of these characters happen to be invisible, the real number of charcters in the text line will not correspond to the number of (visible) charcters that will be displayed on the screen, and thus the "real" cursor position VE knows about will be "wrong" on the screen. (Do not mind if all this seems a lot of nonsense!).

If you are knowledgable about what charcaters you are using, then you may overcome this situation by carefully thinking of which keys you need to press. But if you lose control, then the ERASE command will come to rescue. In fact when you issue the ERASE command, all invisible characters will disappear, and the correspondence between what you see and what you get will be restored.

As you might have infered from the preceding lines, what characters are invisible depend upon the video interface you are using. The ERASE command can be instructed to recognize different sets of invisible characters, as all invisible characters are listed in the VEDB file. The VEDB file that comes on the distribution disc contains a list of invisible characters that is suited for the HP82163 32-column video interface. These characters are all those in the ASCII ranges: 0-31, 127-159, and 255-255. If this set is not good for you, you can make your own by running the MKVEDB program (see Appendix A).

You can take advantage of the ERASE command even if you are not concerned by the unaesthetic effects of invisible characters. For example, you may have the habit of using certain seldom used characters as markers in your text files. This will allow you, for example, to position yourself at different parts of the work file by using the FIND command. Now, if you define the list of "invisible" characters to be the list of characters you are using as markers, the ERASE command will give you a quick means of "cleaning-up" the work file from all markers. The ERASE command is a (programmable) tool for selectively extirpating a set of characters from the work file, although this set has to be defined before you use VE.

Another beneficial effect of the ERASE command is that it will remove unecessary blank spaces at the end of each line of text. In this way you can possibly reduce the amount of memory required to store your text file.

When the ERASE command has executed, a brief message will tell you how many characters have been erased from the work file.

*****   F   -   F I N D   *****

The FIND command will allow you to locate a certain string of characters (or "pattern") in the workfile. You can issue the FIND command by pressing [f] [F], after which VE will wait for you to specify the pattern to be located. You may then type the wanted pattern and press [ENDLINE]. Note that when you type the wanted pattern, then you must also provide a starting (and optionally an ending) delimiter. For example, if you want to locate the string "CHHU is great", you should press [f] [F], type "*CHHU is great*" and then press [ENDLINE]. In this case the asterisk "*" acts as a delimiter, although it could be just any character you please, even a blank space. It is mandatory to specify the first delimiter, the last one being optional. The first character of the typed response will always be assumed to be the delimiter.

When you have typed your pattern and pressed [ENDLINE], the cursor will be positioned on the first occurence of the search pattern that happens after the current cursor position. If the search pattern does not occur between the current cursor position and the end of file, then the search will be repeated starting from the start of file. If the pattern exists, then the cursor will be positioned on its first occurence in the work file, (which, in this case, will obviously be before the current cursor position). If the pattern does not exist in the whole file, then the following error message will be displayed: "Pattern not found".

If you issue the FIND command accidentally, you can nullify its action by not specifying any search pattern, i.e. you only need to press [ENDLINE] after [f] [F].

One powerful feature of the FIND command is that you are allowed to use the so-called "metacharacters". Metacharacters are a special kind of characters that, under certain circumstances, have special meanings. The four meta-characters applicable to the FIND command are: ".", "@", "^" and "$". (Note: these meta-characters are the same as those used in EDTEXT, the text editor in the FORTH/ASSEMBLER ROM or in the TEXT EDITOR ROM.). These four characters have the following special meanings:

The period (".") represents any single character.

The commercial "at" ("@") represents any number of any characters. You can read this metacharacter as "some characters".

The up-arrow ("^") represents the beginning of a line.

The dollar sign ("$") represents the end of a line.

To switch these characters to their special meanings, you have to place a backslash ("\") in the search pattern. Unfortunately, the backslash character is not available on the standard HP71 keyboard, thus, if you whish to use the metacharacters, you will have to assign this character as a typing aid to a user defined key, for example the [/] key. You can do this by typing "DEF KEY '/',CHR$(92);" in the BASIC environment.

To return the metacharacters to their normal meaning, place a second backslash in the string. If you need the backslash character itself in the search pattern, you can use two sequential backslashes. The program will interpret two sequencial backslashes as a single backslash character and not as a switch.

The following paragraphs show some examples of the use of the FIND command with or without metacharacters. In these examples, all keys you have to press are enclosed by square brackets, and all strings you have to type are encolsed by double quotation marks.

[f] [F] "*CHHU IS GREAT*" [ENDLINE] will locate the string "CHHU IS GREAT". Note that the FIND command is case sensitive. This means that you have to type a different response string if you need to locate the string "CHHU is great", and yet another string if you need to locate "CHHU is GREAT".

[f] [F] "*.OG*" [ENDLINE] will locate the string ".OG". Note that in this case the period is taken literally, and it is not interpreted as a metacharacter.

[f] [F] "*\.OG*" [ENDLINE] will locate any three character string which ends with the two letters "OG", for example, "FOG", "DOG" and the same string ".OG" of the preceding example. In this case the backslash character switches the meaning of the period to its special meaning of "any character"

[f] [F] "*\I@T" [ENDLINE] will locate any string starting with an "I" and ending with a "T". For example "IT", "INCANDESCENT" or "IS GREAT". In this case the "@" character means "some characters".

[f] [F] "*\^The" [ENDLINE] will locate the string "The" only if it is at the beginning of a line. Note that when using the up-arrow ("^") special character any other up-arrow characters after the first one are ignored; i.e. eceeding special up-arrow characters are treated as literal up-arrow characters.

[f] [F] "*\^$" [ENDLINE] will locate any line that starts and ends immediately, i.e. any empty line. Note that "empty line" refers here to any line that contains no characters whatever; thus a line of blank spaces is not an empty line.

You can find more information about metacharacters under the REPLACE comand's dictionary entry.

***** G - G O T O *****

The GOTO command is similar to the BACK command, with the only difference that the cursor will be positioned to the right of the current cursor position. (See the BACK command's dictionary entry for details).

***** H - H I G H L I G H T *****

The HIGHLIGHT command allows you to take advantage of a video attribute available on HP and HP-compatible video interfaces, namely the possibility of displaying highlighted, (i.e. in reverse video), characters. The HIGHLIGHT command acts as a toggle: it will alternatively enable and disable highlight mode. To switch highlight mode on and off, just press [f] [H]. Any character(s) you type after you have enabled highlight mode will be displayed in reverse video.

You can tell the current state of the highlight mode by looking at flag annunciator 2 of the LCD display. This annunciator will be visible whenever highlight mode is enabled (on).

In general, you can use the HIGHLIGHT command for two purposes. Firstly, the HIGHLIGHT command can obviously be used simply to emphasize a particular word or portion of text on the screen. Secondly, the HIGHLIGHT command can be used to introduce in a program source file characters whose ASCII value have the most significant bit (the 8-th bit) set. Although this second possibility is a rather technical issue, it is worthwile to know about it. Indeed, the only action that takes place when highlight mode is enabled, is the setting of the 8-th bit of the ASCII value of all characters that are typed. This is because all HP and HP-compatible video interfaces will display in reverse video all characters whose 8-th bit is set.

Another fact you should remember when using highlight mode, is that the highlighted space character is not equivalent to the "normal" space character. This has an influence on the way the WORD [f] [W], position to previous word [f] [(] and position to next word [f] [)] commands operate. The three commands will distinguish between words only if they are separated by a true, i.e. "normal", space character; so they will not operate as you may expect on a line of highlighted text. This may not seem coherent, although it is if you think of highlighted words separated by highlighted spaces as a single entity, (which is normally the case). If you need to write highlighted words that actually need to be distinguished as different words, separate them with "normal" space characters.

*****   I   -   I N S E R T   *****

The INSERT command is usually used in combination with the YANK command, (see the YANK command's dictionary entry), although it can be used by itself. The INSERT command will allow you to merge an external file into the current work file, i.e. make a copy of the external file's contents into the current work file. To do this, just position the cursor on the line above which you whish to insert a copy of the external file's contents and then press [f] [I]. You will then be requested to type the name of the external file. The prompt will propose by default the file name that was used with the last issued INSERT or YANK command. At this point you have the opportunity of aborting the command simply by not specifying any file name, i.e. simply press [f] [-LINE] and [ENDLINE] after [f] [I]. Otherwise, type the file name and end the input with [ENDLINE].

If the file name you typed will cause any problem, an appropriate error message will be displayed (for example "Invalid Filespec" or "File Not Found"), and the prompt will be repeated. Again, if you whish, you can abort the command, or give another file name.

If the file name is valid, that file's contents will be copied into the current work file, above the current line.

*****   J   -   J O I N   *****

The JOIN command allows you to join to two short lines into one longer line. Each time you issue this command, [f] [J], the programm will attempt to join the current line with the following line. If this is possible, any trailing blanks will be removed from the current line, and then the current line will be concatenated with the following line. If this is not possible, then a warning message will be displayed ("Line too long"), and the cursor will be positioned on the following offending line, in order to allow for following JOIN comands to continue joining, or to make it easier to split the offending line.

***** L - L I N E *****

The LINE command will delete the current line from the work file. If you wish to delete a particular line, just position the cursor on that line and then press [f] [L].

***** M - M O V E *****

The MOVE command allows you to move a block of text from one place to another within the work file. Before you issue the MOVE command, you must mark the block of text you wish to move by issuing the SELECT command (see the SELECT command's dictionary entry for more details). If you inadvertedly issue the MOVE command and no block of text has been marked, then VE will display the error message "Missing mark(s)". Once you have marked a block of text, all you have to do is to position the cursor on the line above which you whish to move the marked block of text and then press [f] [M]. Note that the destination position may not reside within the marked block of text. In this case the MOVE command will take no action but display the error message "Inside block".

***** N - N E X T *****

The NEXT command will allow you to locate the next occurence of the search pattern specified in the last issued FIND, REPLACE or QUERY command. This command is used to page through a file in order to locate a particular occurence of a search pattern, which can even contain metacharacters. The search is performed exactly as with the FIND command. See the FIND command's dictionary entry for more details.

***** O - O P E N *****

The OPEN command will open an empty line above the current line. If you whish to open an empty line above a particular line, just position the cursor on that line and then press [f] [O].

***** P - P O S I T I O N *****

The position command allows you to position the cursor on a particular line in the workfile, or to discover wich is the current line. When you press [f] [P], the prompt "Line #:" will be displayed followed by the line number of the current line. Note that the line count starts from Ø. If you are happy to learn the current line number, simply press [ENDLINE], and no other action will take place. On the other hand, if you wish to change the current line, type an expression giving the line number of the line to which you want to go to. For example, if you want to get to the 86th line of the work file, type the response "86-1" and then press [ENDLINE]. (In this example the "minus one" is necessary because the line count starts from zero). If the expression evaluates to a line number which is negative or greater than the total number of lines currently present in the work file, then the cursor will be positioned, respectively, on the first and on the last line of the file.

When you type the response expression, you can even use the following symbolic references, (which really are variables used by VE):

Symbol Meaning

Y1	Current line.
A	Start of marked block, (position of 1st mark).
A1	End of marked block, (position of 2nd mark).
U	Last line of file.

Note that A and A1 are 0 if the respective mark has not been set. The following examples will clarify the use of these symbolic references.

Expression	Position
"U/2"	Middle of file.
"U/3"	One third of the file.
"U*3/4"	Three fourths of the file.
"A+(A1-A)/2"	Middle of marked block.
"Y1+20"	20 lines past current line.
"Y1-15"	15 lines before current line.

The POSITION command is particularly handy when you are correcting assembler source files, as the listing files produced by the FORTH/ASSEMBLER ROM locate errors by source file line number references.

*****   Q   -   Q U E R Y   *****

*****   R   -   R E P L A C E   *****

The QUERY and the REPLACE commands are actually the same command. They both allow you to locate a given text pattern and to replace it with another given text pattern. The difference between the two commands is that the QUERY command allows you to confirm or deny any single replacement, and even to interrupt the command at any point, whilst the REPLACE command performs its action globally and unconditionally. Flag annunciator 3 will be visible in the LCD display whenever the QUERY command is active. When you issue any of these commands, with [f] [Q] or [f] [R], VE will wait for you to specify the search and the replacement patterns. You must specify the search pattern exactly as you do with the FIND command (see the FIND command's dictionary entry for more details). The only difference is now that even the second delimiter is mandatory. You may then specify the replacement pattern, and provide a third optional delimiter. At last, press [ENDLINE] to start the replacement process. For example if you want to replace the string "chew it" with "CHHU IT", you should type the following: "*chew it*CHHU IT*" and then press [ENDLINE]. In this case the asterisk "*" acts as a delimiter, although it could be just any character you please, even a blank space. It is mandatory to specify the first and the second delimiter, the last one being optional. The first character of the typed response will always be assumed to be the delimiter.

The scope of the QUERY and of the REPLACE commands can be controlled. Normally these commands act on the whole file, but if the first mark has been set (with the SELECT command), the replacements will take place only starting from the position of the first mark to the end of the work file. Further more, if even the second mark is set, then the replacements will take place only within the marked block of text. (See the SELECT command's dictionary entry for more information on marks and blocks of text).

When the QUERY is chosen, the program will ask you to confirm every single replacement. When an occurence of the search pattern is located, the QUERY command will display the prompt "Y/N/Q ? :<search pattern>: to :<replacment pattern>:", meaning that you should press [Y] to confirm this particular replacement, [N] to leave this occurence of the search pattern intact, or [Q] to quit the replacement search and make the last line where the search pattern occured the current line.

When all replacements have been made, the commands' action will end and a message will tell you how many replacements were actually made.

During the execution of these commands, it can happen that a replacement will produce a line longer than the current screen window width. If this happens, a warning message ("Replacement too long") will notify that this situation arises, and that particular replacement will not be performed. Instead the programs proceeds with the following replacement(s), if any.

If you issue the QUERY or the REPLACE command accidentally, you can nullify their actions by not specifying any pattern at all, i.e. you only need to press [ENDLINE] after [f] [Q] or [f] [R].

The QUERY and the REPLACE command allow you to use the same metacharacters recognized by the FIND command for the search pattern, (see the FIND command's dictionary entry). Furthermore, you may use yet another metacharacter in the replacement pattern. This new matacharacter is the ampersand ("&"), which any text that matched the search pattern. To switch the ampersand to its special meaning, you have to place a backslash in the replacement pattern, just like the backslash is used to give special meaning to the other metacharacters that may be used in the search pattern.

The following paragraphs show some examples of the use of the QUERY and of the REPALCE commands with or without metacharacters. In these examples, all keys you have to press are enclosed by square brackets, and all strings you have to type are encolsed by double quotation marks.

[f] [R] "*chew it*CHHU IT" [ENDLINE] will replace the string "chew it" with the string "CHHU IT". Note in this first example that the third optional delimiter is not present.

[f] [R] "*HAT*\T&*" [ENDLINE] will replace the string "HAT" with the string "THAT". In this example the ampersand in the replacement string is used with its special meaning.

[f] [R] "*\^Humpty*Dumpty*" [ENDLINE] will replace the string "Humpty" with the string "Dumpty" only if the search string appears at the beginning of a line.

[f] [R] "D\@+@D\(&)*(&)D" [ENDLINE] will "square sums"; for example, "1+1" will be changed into "(1+1)*(1+1). Note in this example the use of the metacharacters, and the use of the letter "D" as a the delimiter.

[f] [R] "*\.\\\.*&\&*" [ENDLINE] will replace a string made up of any single character followed by a backslash and a period with an ampersand and the search string itself. For example "A\." will be changed into "&A\.". This example is tricky: study it carefully, it will give you a good insight on how the metacharacters work.

[f] [R] "*\^$**" [ENDLINE] will replace empty lines with nothing, i.e. this command will delete empty lines. With the QUERY command this particular search pattern is assumed to be a special case: as the empty lines are being replaced with nothing, during the command's execution the deleted lines will be displayed with ">line deleted<", to give the user some feedback and context on the screen. This will obviusly not be reflected in the work file, where the deleted lines will actually disappear.

[f] [R] "*\^ **" [ENDLINE] will delete any starting blanks from a line. This command can be useful to remove indentations.

[f] [R] "*\^.**" [ENDLINE] will remove any starting character from a line. If you have selected a block of text, this command can be used to clear the lines in the block, opposed to deleting them, although this will take a long time, as the replacement takes place a character at a time.

[f] [R] "*\^@**" [ENDLINE] is similar to the previous example, but works faster, as this time the replacement takes place a line at a time. Note that this command will make two replacements for every non-empty line. You can however make it work even faster (one replacement per line) if you specify the optional "one time per line" flag, explained in the

following example.

[f] [R] "*\^....**L" [ENDLINE] will remove ONLY the first four characters in each line. This command is useful if you need to edit text files created with an HP75: it will delete the four digit line number that the HP75 puts at the beginning of every line. Note in this example the final "L" after the last delimiter. This is an optional flag wich means "only one time per line". (This flag will be set by any character following the last (the third) delimiter, not only by an "L"; the "L" was chosen in this example as a mnemonic abbreviation for "line".).

***** S - S E L E C T *****

The SELECT command is to be used in combinaition with the FIND, DELETE, COPY, MOVE, REPLACE, QUERY and YANK commands. The purpose of the SELECT command is to select a portion of text on which the other commands will act. The SELECT command acts as a three-state toggle. The first time you issue the SELECT command the current line will be marked with the first of the two available markers. You will know this has happend when the message "Mark 1" is displayed. The second time you issue the SELECT command, the current line will be marked with the second marker. Again, you will know this has happend when you see the "Mark 2" message. The third time you issue the SELECT command, the two marks will be cleared. You will know this from the "Marks cleared" message.

Usually, the commands that operate in conjunction with the SELECT command will perform their action on the block of text comprised between the first and the second mark. Some commands need only one mark, (and optionally both marks); in this case the block of text is comprised between the assigned mark and the end of the work file.

***** T - T A B *****

The TAB command will position the cursor to the next tab stop. See the description of the TABS parameter in the section "Getting started".

***** V - V I E W *****

The VIEW command will show you how much memory is still available. This command is present because VE does not perform any extensive error checking on memory usage, so the responsability of this is put onto the user. If you do not have much memory left, you should periodically check how much is left. If no more than 500 bytes are left, you should stop your editing session, quitting VE, and get more memory. You can either get more RAM memory modules or make room by purging any unnecessary files from RAM, or by claiming free-ported ports.

If, unfortunately, you run out of memory during a VE editing session and the program crashes, (remember it is your responsability not to get to this point), you should be able to recover most of your own BASIC environment by typing the "CONT 900" to the BASIC interpreter. In this way you will quit VE in a normal way, so that system, loop and timeout status will be restored.

***** W - W O R D *****

The WORD command will delete all characters between the current cursor position and the beginning of the next word on the current line. Thus if the cursor is positioned on the first character of a given word, that word will be deleted simply by pressing [f] [W].

***** Y - Y A N K *****

The YANK command will allow you to copy the whole work file or a block of text to an external file. Usually you will use this command in combination with the SELECT and INSERT commands (see these commands' dictionary entries). If no marks have been set with the SELECT command, then the YANK command will act on the whole workfile. If only one mark has been set, then the YANK command will act on the block of text comprised between that mark and the end of the work file. If both marks have been set, then the YANK command will act on the block of text comprised between the first and the second mark.

When you issue the YANK command by pressing [f] [Y], you will be requested to type the name of the external file. The prompt will propose by default the file name that was used with the last issued INSERT or YANK command. At this point you have the opportunity of aborting the command simply by not specifying any file name, i.e. simply press [f] [-LINE] and [ENDLINE] after [f] [Y]. Otherwise, type the file name and end the input with [ENDLINE].

If the file name you typed will cause any problem, an appropriate error message will be displayed (for example "Invalid Filespec"), and the prompt will be repeated. Again, if you whish, you can abort the command, or give another file name.

If the file name is valid, the YANK command will copy the selected block of text to the external file.

When you issue the YANK (or the INSERT) command for the very first time during an editing session, the default file name proposed will be "BUF". You can think of this file as a temporary buffer which can exist only during an editing session, but that will certainly be purged when you quit the VE program. If you use any other file name, that file will be created permanently, (i.e. it will survive your quitting the editor). The "BUF" file is particularly handy when you need to move quickly a block of text from one file to another. In fact, during any VE editing session, you can "restart" the editor by pressing the key sequence [g] [ON]. In this case, you can change the current work file without destroying the contents of the "BUF" file. At the same time you are sure that when you will end the editing session, (probably after INSERTing the "BUF" file in a new work file), the "BUF" file will be purged, so that you will not find any garbage files in RAM.

There is yet another feature. If you issue the YANK command without having selected any block of text, (i.e. when you are acting on the whole work file), you can even specify a HPIL mass-memory device in the filename. This will allow you to make back-up copies of your work files on discs and/or tapes without getting out of the program. Note that if you do this without any mass-memory device connected to the HPIL loop, or without any medium in the mass-memory device, you will get the "Invalid Filespec" error.

MACRO Programming

The following section is for advanced users. You may wish to practice a little using VE before you go on to the following, as you will get the most out of the following only if you have a good understanding of how VE works and how to use its commands. You should also have a little knowledge of programming in BASIC, although all instruction will be described thoroughly.

A unique feature of VE which definitively puts it above most text editors, even of larger computers, is the possibility of using user defined keys that may contain even any command to which VE can respond, with or without user parameters. In other words, VE is MACRO programmable text editor.

When you create a user defined key that contains a VE MACRO definition, you will have to respect a rather complicated format, (which is NOT described in this document). For this reason the distribution disc contains an additional support program called MKVEKEYS (MaKe VE KEY definitionS), which will ease the difficulty of defining VE MACROs. As long as you use MKVEKEYS according to the following instructions, you will be able to create user defined keys in the format that VE pleases. Beware! this program is "quick and dirty" and it lacks sophisticated user protection, so you must know what you are up to when using it.

As a first WARNING: DO NOT RENUMBER MKVEKEYS! (This is not really necessary, but it will allow you to use the following description as a reference, even at a later time. The following paragraphs presume that MKVEKEYS is in the form it was released on the distribution disc, i.e. the real program starts at line 1000).

When you want to create a VE MACRO, in general, you will have to carry out the following five step procedure:

(1) EDIT a BASIC program file that will contain the key definition. This program will contain only DATA statements, and it must not contain lines with line numbers greater than 999. In fact MKVEKEYS expects as input DATA string items in lines 1, 999, and it will create as output a single user definition. The first DATA item must be the name of the key to be defined, (written as an ordinary HP71 keycode), all the following items being the definition itself. For every key press you wish to put into the definition, you will have to write a single DATA string item, this being an ordinary HP71 keycode. When you write out a command key to a key definition, (i.e. any key VE recognizes as a command), the command itself may require one or two parameters, (an input string or an input string and a list of key presses). It is possible to specify inputs simply by keying them in as the item following the name of the command key. It is also possible to make VE stop at inputs and wait for a user supplied response, by keying in the item "STOP" (capital letters), or accept default values supplied at inputs, by keying in the item "DEFAULT".

(2) EDIT the MKVEKEYS program file.

(3) DELETE any existing lines between 1 and 999 from the MKVEKEYS program file, in order to be sure no preceding key definition will

interfere with the new one.

(4) MERGE the BASIC program file you created in step 1 into MKVEKEYS.

(5) RUN MKVEKEYS. (Note: when you run MKVEKEYS you must have the VELEX lex file in memory). MKVEKEYS will read the key definition and create a user defined key according to this definition and with respect to the fromat expected by VE's key interpreter. When you subsequently run VE, you can turn your user defined keyboard on and off with the normal HP71 key sequence [f] [USER].

The following paragraphs will give you some examples. In these examples, all keys you have to press are enclosed by square brackets, and all strings you have to type are encolsed by double quotation marks.

EXAMPLE 1: Create a typing aid that will write "CHHU is great" at the current cursor position; assign this definition to [f] [1].

```
"EDIT EX1" [ENDLINE]
"10 DATA F1" [ENDLINE]
"20 DATA C,H,H,U,' ',i,s,' ',g,r,e,a,t" [ENDLINE]
"EDIT MKVEKEYS" [ENDLINE]
"DELETE 1,999" [ENDLINE]
"MERGE EX1" [ENDLINE]
[RUN]
```

Now if you edit any file with VE, enable USER mode and press [f] [1] the string "CHHU is great" will be written to the screen and to the work file. This example is exactly equivalent to an ordinary "typing aid" key definition that you have in the ordinary HP71 BASIC environment. The first data item indicates which key is to be defined, all the following data items spell out the typing aid a character at a time. The following examples will show you how to use VE commands in your key definitions.

EXAMPLE 2: Create a MACRO that will write out your address at the beginning of the work file; assign this key definition to [f] [2]. (In the example we will use CHHU's address).

```
"EDIT EX2" [ENDLINE]
"10 DATA F2" [ENDLINE]
"20 DATA #162,FO,FO,FO,FO,FO" [ENDLINE]
"30 DATA C,H,H,U,#159,#51" [ENDLINE]
"40 DATA P,.,O,.,' ',B,O,X,' ',1,0,7,5,8,#159,#51" [ENDLINE]
"50 DATA S,A,N,T,A,' ',A,N,A,#159,#51" [ENDLINE]
"60 DATA C,A,' '9,2,7,1,1,' ',U,.,S,.,A,." [ENDLINE]
"EDIT MKVEKEYS" [ENDLINE]
"DELETE 1,999" [ENDLINE]
"MERGE EX2" [ENDLINE]
[RUN]
```

This MACRO does the following: positions the cursor on the first line of the work file. The "#162" is the equivalent of pressing the [g] [UP arrow] key. Makes space for the address by issuing five OPEN line commands. Writes out the first line. Moves the cursor to the beginning of the line: the "#159" is the keycode for [g] [LEFT arrow]. Moves the down one line: this is done by the "#51" which is the keycode for [DOWN arrow]. Repeating this scheme, writes out the rest of the address. Note that in this MACRO the key sequence [g] [LEFT arrow] [DOWN arrow] is used in place of [ENDLINE], as the MACRO cannot know what the current state of the Insert/Replace mode is.

For the same reason this MACRO has to create space with the five OPEN line commands.

This example shows how cursor control keys can be programmed into a MACRO as well as a command (the OPEN command) that takes no parameters.

EXAMPLE 3: Create a MACRO that will momentarily display the line number of the current line. Assign this key definition to [f] [3]

```
"EDIT EX3" [ENDLINE]
"10 DATA F3" [ENDLINE]
"20 DATA FP,DEFAULT" [ENDLINE]
"EDIT MKVEKEYS" [ENDLINE]
"DELETE 1,999" [ENDLINE]
"MERGE EX3" [ENDLINE]
[RUN]
```

This MACRO will simply execute the POSITION command, and take the DEFAULT answer, i.e. the current line. During the process the current line number is momentarily displayed.

EXAMPLE 4: Create a MACRO that deletes a block of lines spanning from the current line to the first line containing an occurence of a user specified pattern. Assign this key definition to [f] [4].

```
"EDIT EX4" [ENDLINE]
"10 DATA F4" [ENDLINE]
"20 DATA FO,FL,FS,FF,STOP,FS,FD,STOP" [ENDLINE]
"EDIT MKVEKEYS" [ENDLINE]
"DELETE 1,999" [ENDLINE]
"MERGE EX4" [ENDLINE]
[RUN]
```

In this example the sequential use of the OPEN line and LINE delete commands will unconditionally clear all marks. The first SELECT command will set the first mark at the current line. Then the FIND command will STOP to wait for user input. After the execution of the FIND command the second SELECT command will set the second mark at the first line containing a match to the user specified search pattern. Note that this position may even before the original current line if the match didn't occure between the original current line and the end of file. If no match at all was found, then this position will simply be the original current line. (This is exactly how the FIND command moves the cursor normally). At last the DELETE command will STOP to let the user confirm or not the deletion.

EXAMPLE 5: Create a MACRO that will FIND a predifined pattern, for example, a string made up of five spaces. Assign this key definition to [f] [5].

```
"EDIT EX5" [ENDLINE]
"10 DATA F5" [ENDLINE]
"20 DATA FF,'*     *'" [ENDLINE]
"EDIT MKVEKEYS" [ENDLINE]
"DELETE 1,999" [ENDLINE]
"MERGE EX4" [ENDLINE]
[RUN]
```

This MACRO shows how you can program a predefined response to an input request. This MACRO can be useful if you are writing a document in which each new paragraph has its first line indented by five spaces: it will let you move the cursor to the beginning of the next paragraph, no matter where the cursor is currently positioned.

Virtual Windows

As explained in the "Getting Started" section, VE allows you to define a window of the real screen that will be used by the editor to display text. When doing this you are even allowed to define a window that is wider than the available screen width, up to a 150 characters width as a maximum.

NOTE: the program will not check the "Window width" parameter. It is your responsability not to exceed the 150 characters limit.

If you define the window to be wider than the available screen width, VE will automatically switch to it's second mode of operation, and it will work as a screen oriented line editor. (This mode is particulary useful if you have the HP82163 32-column video interface.). In this mode only one line at a time will be displayed on the screen, although all commands and all editing keys will work exactly as they do in the normal full screen mode. So there is nothing new you will have to learn to use virtual windows: you just have to get used to working with one line at a time. The only other difference you will notice, is the different position of the message line: its position will be calculated by the program, and it will appear immediately after the (only) line of displayed text.

History and Credits

VE was initially conceived of and developed for his own needs by Stefano Piccardi [CHHU 487] in November 1985 on the HP75 portable computer and the HP82163 32-column video interface. (The HP75 version - VE75 - is now available on the CHHU SWAP DISC #4). The general outline and the first working version were designed and coded in only three days.

More or less at the same time the September 1985 issue of the CHHU Chronicle was published, and therein appeared an article by Michael Markov [CHHU 3]: "Whishlist for the CHHU HP71 ROM Project", where one whish was for "a good (fast) full screen editor" for the HP71. Stefano Piccardi recognized immediately that this could be VE.

Stefano Piccardi thus involved Stefano Tendon [CHHU 635], and asked him to search for and/or eventually develop any LEX files that would be necessary to implement VE on the HP71. Michael Markov also helped at this stage (December 1985-February 1986) by providing all useful LEX files he had access to as the coordinator of CHHU's SWAP DISC project.

In May 1986 the porting of the editor from the HP75 to the HP71 was accomplished by Stefano Tendon. The first prototype version for the HP71 worked, slowly, using an HP150 as a terminal. At the same time it was decided it was necessary to have a lex file. By July 1986 Stefano Tendon coded a working version of the VELIST statement, and the editor's speed performance became more acceptable. At this stage of development, VE71 was adapted to work with a PACSCREEN video interfaced borrowed from Angelo Maggio [CHHU 840].

During the following months many new ideas were investigated and implemented jointly by Stefano Piccardi and Stefano Tendon; VE71 underwent continuous improvements and enhancements, making it into quite a different program from the original HP75 version. Screen handling was improved, new commands were added (INSERT, POSITION, QUERY and YANK), the execution of existing commands was optimized for the HP71, messages were made clearer or added, and VELEX was expanded with two more statements: CHIRP and CUR$. The CHIRP statement, although trivial, was first written by John Baker [CHHU 618].

Flavio Casetta [CHHU 827], seeing a prototype version of VE71, lamented the impossibility of using user defined keys. To get this feature, Stefano Tendon added the MAPKEY$ function to VELEX and Stefano Piccardi re-wrote from scratch VE71's key handler routines, (September 1986). While doing this, Stefano Piccard added the MACRO ability to user defined key definitions. At the same time it was realized that key handling required the FKEY statement, which was developed back in May 1985 by Jean Pierre Bondu, a member of the French user group S.I.G..

In October 1986 Stefano Piccard analyzed, optimized and doucmented all code written in BASIC. During the process he discovered that the second mode of operation of VE, (as a screeeen oriented line editor for text files wider than the available screen width), could be added with little effort. To do this it was required to modify the CUR$ statement, in order to make it accept a third optional parameter. This was done jointly by Stefano Piccardi and Stefano Tendon.

During the same month Stefano Tendon assembled and documented the final version of VELEX, and wrote the present user's manual.

Version 1.1 of the package was released the first week of November 1986, a year after the first idea.

Appendix A - MKVEDB

The MKVEDB support program will allow you to create a new VE DATA BASE file. The VE DATA BASE file (VEDB) contains two types of information: the list of "invisible" characters used by the ERASE command, and a list of hardware-dependent escape sequences. When you run the MKVEDB program, all default inputs proposed by the program are suited for the HP82163 video interface and all compatible video interfaces. If you are using a different kind of video interface, the MKVEDB program will require you to know the escape sequences performing the following functions and applicable to the particular video interface you are using:

- Move cursor right one character.
- Move cursor down one line .
- Move cursor left one character.
- Make cursor visible.
- Make cursor invisible.
- Clear screen from current line to bottom of screen.
- Scroll screen down one line.
- Scroll screen up one line.
- Display replace mode cursor.
- Display insert mode cursor.
- Clear or reset video interface device.
- Clear screen page.

In VE version 1.1 all escape sequences, (except the last one), are two characters long, although MKVEDB will allow you to create escape sequences up to four characters long. Problems will arise with VE if you create longer escape sequences, (although we do not know of any HPIL video interface which needs longer escape sequences).

If you use MKVEDB only to change the list of invisible character, keep in mind that you can include any ASCII character in this list, from ASCII 0 to ASCII 255. (In this case remember NOT to modify the default escape sequences!). When you answer the input prompt, you must specify these characters as ASCII ordered subranges.

Appendix B - VEFOLD

VEFOLD is an optional run time support program. If VEFOLD resides in memory, it will be invoked automatically by VE before any editing session is opened. If you want to use VEFOLD, you only have to load it into memory before calling VE. Do not call VEFOLD by itself, but let VE do it automatically. Note that the VEFOLD program file MUST NOT BE RENAMED, unless you want VE to ignore it.

VEFOLD is a sub program that expects the following parameters:

```
VEFOLD (<WorkFileChannel#>,<FILESZR(WorkFile)-1>,<WindowWidth>)
```

VEFOLD was designed in order to overcome the incompatibility that exists between the active window width parameter and any file that was created with a wider window width.

VEFOLD will scan the work file and fold any lines that are longer than the current screen window width. The version of VEFOLD that comes on the distribution disc will fold lines at spaces and dashes; i.e. long lines are folded at the rightmost occurence, with respect to the current screen window width, of a space or a dash. (If you wish VEFOLD to fold lines even at other characters, along with or in place of spaces and dashes, you can modify the assignment of the C$ variable in line 20 of the VEFOLD BASIC program file). If no such occurrence exists then a "Line Too Long" warning message will be displayed. Members of C$ are left at the END of the folded portion of the line. If a line needs to be folded more than once to fit the current screen width, it will be if possible.

EXAMPLE: if the current screen window width is 7, C$=" " and the work file contains:

```
   this is the time
   for all good men
```

then the work file will become

```
   this_
   is the_
   time
   for_
   all_
   good_
   men
```

where underscores represent dangling spaces.

Appendix C - Quick Reference Guide

Key(s)	Function
[ATTN]	End program.
[g] [ON]	Restart editor.
[UP arrow]	Move cursor up one line.
[DOWN arrow]	Move cursor down one line.
[LEFT arrow]	Move cursor left one character.
[RIGHT arrow]	Move cursor right one character.
[g] [UP arrow]	Move cursor to first page.
[g] [DOWN arrow]	Move cursor to last page.
[g] [LEFT arrow]	Move cursor to first column.
[g] [RIGHT arrow]	Move cursor to last column.
[f] [(]	Move cursor to previous word on current line.
[f] [)]	Move cursor to next word on current line.
[g] [ENDLINE]	Move cursor down one page.
[f] [ENDLINE]	Move cursor up one page.
[ENDLINE]	Carriage return; start a new line.
[g] [CTRL]	CONTROL character following.
[f] [BACK]	Destructive back-space.
[f] [-CHAR]	Delete character under cursor.
[f] [I/R]	Toggle insert/replace mode.
[f] [LC]	Toggle upper/lower case mode.
[f] [-LINE]	Delete from cursor to end of current line.
[f] [USER]	Toggle USER defined keys on/off.
[f] [A]	AUTOMATIC wrap mode toggle.
[f] [B]	BACK up to previous occurence of next key.
[f] [C]	COPY marked block.
[f] [D]	DELETE marked block.
[f] [E]	ERASE invisible characters.

[f] [F]	FIND a given search pattern.
[f] [G]	GO to next occurence of next key.
[f] [H]	HIGHLIGHT mode toggle.
[f] [I]	INSERT from buffer/external file.
[f] [J]	JOIN following lines.
[f] [L]	LINE delete.
[f] [M]	MOVE marked block.
[f] [N]	NEXT occurence of search pattern.
[f] [O]	OPEN empty line above current line.
[f] [P]	POSITION to given line number.
[f] [Q]	QUERY replace pattern.
[f] [R]	REPLACE pattern.
[f] [S]	SELECT block of lines - Clear marks.
[f] [T]	TAB to next tab stop position.
[f] [V]	VIEW available memory.
[f] [W]	WORD delete, (from cursor to next word).
[f] [Y]	YANK to buffer/external file.

Flag annunciator	Meaning when visible
0	Automatic wrap around mode off.
1	Insert mode on.
2	Highlight mode on.
3	Query replace command in action.
4	Lower case mode on.

VELEX Keywords

The VELEX lex file implements the following keywords: CHIRP, CUR$, MAPKEY$ and VELIST, which are briefly described in the following paragraphs:

Name: CHIRP
Type: Statement
Purpose: Give a BASIC keyword to the mainframe chirp routine.
Syntax: CHIRP

Name: CUR$
Type: Function
Purpose: Produce a cursor positioning escape sequence string.
Syntax: CUR$(<row>,<col>[,<maxcol>])
where <row>, <col> and <maxcol> are in the range 0-255.
Detail: The CUR$ string is equivalent to:
CHR$(27)&"%"&STR$(<row>)&STR$(<col>)
If the optional <maxcol> parameter is given, the CUR$ string is equivalent to:
CHR$(27)&"%"&STR$(<row>+<col> DIV <maxcol>)& STR$(<col> MOD <maxcol>)

Name: MAPKEY$
Type: Function
Purpose: Map key to a unique one-byte code, and return code in a one-byte string.
Syntax: MAPKEY$(<keycode string>)
Algorithm:

```
      ! K$=KEYWAIT$
      if K$ is longer than a byte
         get keycode of K$
         case (keycode)
            null string   :   return (null string)
            unshifted key :   return (keycode + 90)
            f-shifted key :   return (keycode + 112)
            g-shifted key :   return (keycode)
         endcase
      else if is_alfa (K$) return (toggle_case (K$))
      else return (K$)
```

Name: VELIST.
Type: Statement.
Purpose: This command is similar to PLIST, but with a different syntax. Executes faster than PLIST.
Syntax: VELIST #<channel#>,<start record#>,<end record#>
where <start record#> and <end record#> may be variable expressions.

Resource Allocation

The following tokens are used by VELEX, as distributed on the distribution disc:

KEYWORD	TOKEN NUMBER IN HEX
CHIRP	5C 06

CUR$	5C 07
MAPKEY$	5C Ø8
VELIST	5C Ø9

The VE program also uses the FKEY keyword, (which can be found in FKEYLEX, from the french user group S.I.G.), with token 71 ØB (HEX).

```
================================================================================
00/09/14 17:04:52 FILE: VEDIT       BASIC   470 09/14/00 17:04
================================================================================
10 ! VEDIT: example of execution of VE suitable for the HP82163 video interface
or compatible
20 ! HPIL devices.
30 ! look for data base
40 ON ERROR GOTO 60
50 CAT VEDB @ OFF ERROR @ GOTO 70
60 OFF ERROR @ BEEP 1400,.075 @ DISP "VEDB must be in RAM:"
65 DISP "Load it form mass memory or create it using MKVEDB" @ END
70 ON ERROR GOTO 90
80 CAT VEFOLD @ OFF ERROR @ GOTO 100
90 OFF ERROR @ BEEP 1400,.075 @ DISP "WRN: VEFOLD not in RAM"
100 CALL VE(16,32,16,32,"%48","4") IN VE71
110 ! DON'T forget to put an END after CALLing VE; avoid a nasty HP71 BUG !
120 END
================================================================================
00/09/14 17:05:09 FILE: VE71        BASIC  9337 09/14/00 17:05
================================================================================
1 ! VE Copyright (C) Stefano Piccardi & Stefano Tendon, 1986; LAST REVISION: 198
61101
10 SUB VE(R0,C0,R1,C1,D$,T$)
15 POKE "2F441","F" @ D1=FLAG(-3,1)
20 DIM C1$[2],C2$[2],C3$[2],C4$[2],C5$[2],C6$[2],C7$[2],C$[2],E$[4],E1$[2],C0$[1
60],V$[6]
25 DIM R0$[2],R1$[2],F$[16],L$[160],S$[150],S1$[150],S2$[150],R$[150],Z$[1],Y$[1
6],G$[16]
30 DIM U1$[14],U4$[4],U5$[9],O$[44],M$[255],N$[97],K$[4],K1$[1],Q$[32],H$[4],J$[
1]
35 INTEGER S,S0,S1,Y,Y0,Y1,A,A1,I,C2,C3,C4,W,R,D,P,K,L,G,H,J,M0,Q,S2,T1,U,W,W9,X
40 V$="VE:1.1" @ IF NOT POS(VER$,V$) THEN 'VERERR'
45 V$="STR:A" @ IF NOT POS(VER$,V$) THEN 'VERERR'
50 V$="EDT:A" @ IF NOT POS(VER$,V$) THEN 'VERERR'
51 V$="CSTU:A" @ IF NOT POS(VER$,V$) THEN 'VERERR'
55 CALL VER(V$) @ ON ERROR GOTO 431
60 ASSIGN #1 TO VEDB @ S2=VAL('0'&T$)
70 R0=R0-1 @ R1=(R1-2)*(C1<=C0) @ M0=(R1+1)*CEIL(C1/C0) @ C2=C1 @ C3=C2-1 @ C4=C
3-10
80 C0$=RPT$(" ",C0*CEIL(C1/C0)) @ READ #1;M$,C1$,C2$,C3$,C4$,C5$,C$,R0$,R1$,C6$,
C7$,E1$,E$
85 A$=">line deleted<"[1,C2]
90 O$="ÂĖöiëæ ¥Ä¢Ü^ Ö¤˜ò àiùú¿ä¨è£ñ§ÙÛ¯`ï"
95 H$=""
100 Y$="BUF" @ G$=Y$ @ N$="" @ W=0 @ R=1 @ H=0 @ Q=0 @ CFLAG 0,1,2,3 @ GOSUB 470
 @ GOSUB 3005
105 U1$=PEEK$("2F78D",14) @ U4$=PEEK$("2F946",4) @ U5$=PEEK$("2F958",9)
120 DISPLAY IS D$ @ PRINTER IS D$ @ DELAY 0,0 @ PWIDTH INF @ ENDLINE @ STD
130 PRINT C5$;E$; @ GOSUB 135 @ GOTO 155
135 PRINT "Visual Editor - ";V$ @ PRINT "  Copyright (C) 1986" @ PRINT
140 PRINT "Stefano Piccardi" @ PRINT "Via A. Panizzi 13" @ PRINT "20146 MILANO,
Italy" @ PRINT
145 PRINT "Stefano Tendon" @ PRINT "Cantone Delle Asse 5" @ PRINT "29100 PIACENZ
A, Italy"
150 RETURN
155 F$=FNI$("File:","",16,0) @ PRINT C5$;E1$; @ IF NOT LEN(F$) THEN CAT ALL @ GO
TO 130
195 U=FILESZR(F$) @ IF U=-57 THEN CREATE TEXT F$ @ U=0
200 IF U<0 THEN PRINT C4$; @ GOSUB 915 @ BEEP @ DISP MSG$(ABS(U)) @ GOTO 905
205 ON ERROR GOTO 430 @ ASSIGN #1 TO F$ @ IF U=0 THEN PRINT #1;"" ELSE U=U-1
```

```
210 ON ERROR GOTO 215 @ CALL VEFOLD(#1,U,C2) IN VEFOLD
215 OFF ERROR
250 S$="" @ R$="" @ Y=0 @ Y0=0 @ GOSUB 520 @ GOSUB 1000
255 K$=KEYWAIT$ @ K1$=MAPKEY$(K$) @ IF FLAG(-9) THEN GOSUB 290 ELSE GOSUB 265
260 GOTO 255
265 K=POS(O$,K1$) @ IF K THEN 275
266 IF NUM(K1$)<128 THEN J$=CHR$(128*H+NUM(K1$)) @ GOSUB 800 ELSE CHIRP
270 RETURN
275 GOSUB "C"&STR$(K)
280 IF NOT POS(H$,K1$) THEN RETURN
285 IF KEYDOWN(K$) THEN 275 ELSE RETURN
290 N$=KEYDEF$(K$) @ IF NUM(N$)#59 THEN 265 ELSE N$[1,1]="" @ K$=""
295 IF LEN(N$) THEN K1$=CHR$(NUM(N$)) @ N$[1,1]="" @ GOSUB 265 @ GOTO 295 ELSE R
ETURN
300 DEF FNI$[96](P$,D$,L,F)
305 GOSUB 670 @ PRINT C4$;P$;
310 IF NUM(N$)#220 THEN LINPUT "",D$;R$ @ POKE "2F441","F" @ GOTO 325
315 T=POS(N$,CHR$(220),2) @ R$=N$[2,T-1] @ N$=N$[T+1] @ IF NOT LEN(R$) THEN R$=D
$
320 PRINT R$;
325 ON ERROR GOTO 335 @ IF NOT L THEN T=VAL(R$)
330 OFF ERROR @ IF (L#0)*LEN(R$)<=L THEN 340 ELSE G=37 @ GOSUB 420 @ GOTO 305
335 OFF ERROR @ PRINT FNE$(ERRM$); @ GOTO 305
340 IF F THEN GOSUB 710
345 GOSUB 1805 @ F=FLAG(4,FLAG(-15)) @ FNI$=R$ @ END DEF
350 DEF FNE$(M$) @ PRINT FNM$(M$); @ BEEP @ WAIT NOT FLAG(-9) @ GOSUB 670 @ FNE$
="" @ END DEF
355 DEF FNM$(M$) @ GOSUB 670 @ PRINT M$[1,C2-1]; @ FNM$="" @ END DEF
360 DEF FNY=Y0<=Y1 AND Y1<=Y0+R1
370 DEF FNK$ @ IF NUM(N$)#220 THEN FNK$=KEYWAIT$ @ GOTO 374
372 FNK$=N$[2,2] @ IF NUM(N$[3])#220 THEN N$[1,2]=CHR$(220) ELSE N$[1,3]=""
374 END DEF
420 PRINT FNE$(MSG$(ABS(G))); @ RETURN
430 GOSUB 915
431 BEEP @ DISP ERRM$ @ GOTO 905
440 IF U<0 THEN U=0 @ RESTORE #1,0 @ PRINT #1;""
441 RETURN
470 A=-1 @ A1=-1 @ RETURN
490 READ #1,Y1;S1$ @ IF S1$#L$ THEN REPLACE #1,Y1;L$
495 RETURN
510 PRINT CUR$(Y,0);C0$;CUR$(Y,0); @ RETURN
520 Y1=Y0+Y @ L$=C0$ @ READ #1,Y1;L$ @ RETURN
525 PRINT CUR$(Y,0);
530 PRINT C$; @ VELIST #1,Y1,Y0+R1 @ RETURN
540 T=MAX(0,SPAN(L$," ",MAX(1,POS(RTRIM$(L$)," ",X+1)))-1) @ RETURN
550 S$="" @ R$=FNI$(Q$,"",96,1) @ IF NOT SPAN(R$," ") THEN POP @ GOTO 1230
570 Q$=CHR$(NUM(R$)) @ R$=RTRIM$(R$)&Q$ @ P=POS(R$,Q$,2) @ S$=R$[2,P-1] @ R$=R$[
P+1]
580 P=POS(R$,Q$) @ L=LEN(R$)>P+1 @ R$=RTRIM$(R$[1,P],Q$)
590 D=0 @ I=0 @ FOR J=1 TO LEN(S$) @ S1=NUM(S$[J])
591 IF S1=92 THEN I=NOT I ELSE IF S1=94 AND I AND D=0 THEN D=1 ELSE IF S1=36 AND
 D=1 AND I THEN
592 NEXT J @ D=D=2 AND NOT LEN(R$) @ RETURN
600 PRINT FNM$("Working..."); @ RETURN
610 Q$=FNK$ @ IF LEN(Q$)#1 THEN FKEY Q$ @ POP
611 RETURN
615 P=POS("YNQ",UPRC$(FNK$)) @ IF NOT P THEN CHIRP @ GOTO 615 ELSE RETURN
620 PRINT A1-A+1;"line(s)..."; @ RETURN
625 IF FNY THEN Y=Y1-Y0 @ GOSUB 520 ELSE GOSUB 635 @ GOSUB 710
626 GOSUB 1230 @ RETURN
```

```
630 IF FNY THEN Y=Y1-Y0 @ RETURN
635 Y0=MAX(0,Y1-(R1 DIV 4)) @ Y=MIN(Y1,R1 DIV 4) @ RETURN
640 I=A*(A#-1) @ J=U*(A1=-1)+A1*(A1#-1) @ RETURN
670 PRINT C5$;CUR$(M0,0);C$; @ RETURN
710 GOSUB 520 @ PRINT C5$;E$; @ VELIST #1,Y0,Y0+R1 @ RETURN
740 GOSUB 490 @ IF A>=0 AND A1>=0 THEN RETURN
745 POP @ PRINT FNE$("Missing mark(s)"); @ GOTO 1230
750 IF Y1<A OR Y1>A1 THEN RETURN
755 POP @ PRINT FNE$("Inside block"); @ GOTO 1230
800 IF X=C4 THEN BEEP 3500,.05
801 IF R THEN X=X+1 @ L$[X,X]=J$ @ PRINT J$; @ GOTO 840
810 IF LEN(L$)=C2 THEN CHIRP @ RETURN
820 X=X+1 @ L$[X,0]=J$
830 PRINT C5$;L$[X];CUR$(Y,X,C0);C4$;
840 IF X#C2 THEN RETURN
841 IF W THEN CHIRP @ GOTO 1600
842 BEEP 100,.01 @ IF NUM(L$[X])#32 THEN GOSUB 4100 @ GOSUB 1600
843 IF NUM(L$[X+1])=32 THEN GOSUB 2505
844 W9=R @ IF R THEN GOSUB 1800
845 GOSUB 2400 @ GOSUB 2200 @ IF W9 THEN GOSUB 1800
850 BEEP 100,.01 @ RETURN
900 'C1': GOSUB 910 @ IF FILESZR(G$)>=0 THEN PURGE G$
901 CAT F$
905 T=FLAG(-3,D1) @ POKE "2F441","0" @ GOTO 9998
910 GOSUB 490 @ PRINT C6$;E$;
915 POKE "2F78D",U1$ @ POKE "2F946",U4$ @ POKE "2F958",U5$
925 CFLAG 0,1,2,3,4 @ RESTORE IO @ RETURN
1000 'C2': GOSUB 490
1005 Y0=0
1006 Y=0 @ X=0
1010 GOSUB 710 @ GOTO 1230
1100 'C3': GOSUB 490 @ Y0=MAX(0,U-R1) @ Y=MAX(0,U-Y0) @ X=0 @ GOTO 1010
1200 'C4': GOSUB 490
1205 IF Y1=U THEN CHIRP @ RETURN
1210 S=Y @ IF S=R1 THEN Y0=Y0+1 @ PRINT C5$;R0$; @ GOSUB 670 @ GOSUB 510 ELSE Y=
Y+1
1220 GOSUB 520 @ IF S=R1 THEN PRINT L$;
1230 PRINT CUR$(Y,X,C0);C4$; @ RETURN
1300 'C5': GOSUB 490
1305 IF NOT Y1 THEN CHIRP @ RETURN
1310 S=Y @ IF NOT S THEN Y0=Y0-1 @ PRINT C5$;R1$; @ GOSUB 670 @ GOSUB 510 ELSE Y
=Y-1
1320 GOSUB 520 @ IF NOT S THEN PRINT L$;
1330 GOTO 1230
1400 'C6': GOSUB 490 @ Y0=MIN(Y0+R0,U) @ GOTO 1006
1500 'C7': GOSUB 490 @ Y0=MAX(Y0-R0,0) @ GOTO 1006
1600 'C8': IF X THEN X=X-1 @ PRINT C3$; ELSE CHIRP
1610 RETURN
1700 'C9': IF X#C3 THEN X=X+1 @ PRINT C1$; ELSE CHIRP
1710 RETURN
1800 'C10': R=FLAG(1,R)
1805 IF R THEN PRINT C6$; ELSE PRINT C7$;
1810 RETURN
1900 'C11': GOSUB 540 @ X=T @ GOTO 1230
2000 'C12': IF NOT X THEN CHIRP @ RETURN
2005 IF X>LEN(L$) THEN 1600
2010 IF R THEN L$[X,X]=" " @ X=X-1 @ PRINT C3$;" ";C3$; @ RETURN
2015 L$[X,X]="" @ X=X-1
2020 PRINT C5$;C3$;L$[X+1];" ";CUR$(Y,X,C0);C4$; @ RETURN
2100 'C13': X=0 @ GOTO 1230
```

```
2200 'C14': X=MIN(C3,LEN(L$)) @ GOTO 1230
2300 'C15': H=NOT FLAG(2,NOT H) @ RETURN
2400 'C16': GOSUB 490 @ T1=MAX(0,SPAN(L$," ")-1) @ IF R THEN 2440
2405 GOSUB 470 @ R$=RPT$(" ",T1)&LTRIM$(L$[X+1]) @ L$=L$[1,X] @ PRINT C0$[1,C2-X
] @ GOSUB 490
2415 IF Y1#U THEN 2430
2420 IF Y#R1 THEN PRINT R$;
2425 U=U+1 @ X=T1 @ RESTORE #1,U @ PRINT #1;R$ @ GOTO 1210
2430 GOSUB 1210 @ GOSUB 2905 @ L$=R$ @ GOSUB 490
2435 PRINT L$; @ X=T1 @ GOTO 1230
2440 IF Y1=U THEN X=T1 @ GOSUB 470 @ RESTORE #1,Y1+1 @ PRINT #1;"" @ U=U+1 ELSE
X=0
2445 GOTO 1210
2500 'C17': IF X>=LEN(L$) THEN RETURN
2505 L$[X+1,X+1]="" @ PRINT C5$;L$[X+1];" "; @ GOTO 1230
2600 'C18': GOSUB 470 @ DELETE #1,Y1
2610 PRINT C5$; @ U=U-1 @ IF U<0 THEN GOSUB 440 @ GOTO 1005
2615 Y0=MAX(0,Y0-(Y0>U)) @ Y=MAX(0,Y-(Y1>U)) @ GOSUB 520 @ GOSUB 525 @ GOTO 1230
2700 'C19': L$=L$[1,X] @ GOSUB 490 @ PRINT C5$;C0$[1,C2-X]; @ GOTO 1230
2800 'C20': GOSUB 540 @ IF T<=X THEN T=LEN(L$)
2805 L$=L$[1,X]&L$[T+1] @ PRINT C5$;L$[X+1];C0$[1,C2-LEN(L$)]; @ GOTO 1230
2900 'C21': GOSUB 490
2905 GOSUB 470 @ PRINT C5$; @ INSERT #1,Y1;"" @ U=U+1 @ GOSUB 520 @ GOSUB 525 @
X=0 @ GOTO 1230
3000 'C22': LC
3005 T=FLAG(4,FLAG(-15)) @ RETURN
3100 'C23': GOSUB 490 @ Q$="F:" @ GOSUB 550
3105 T=SEARCH(S$,X+2,Y1,U,1) @ IF T THEN 3115
3110 T=SEARCH(S$,1,0,Y1,1)
3112 IF NOT T THEN BEEP @ PRINT FNM$("Pattern not found"); @ GOSUB 520 @ GOTO 12
30
3115 Y1=IP(T) @ X=IP(FP(T)*1000)-1 @ GOSUB 625 @ RETURN
3200 'C24': Q$="R:" @ CFLAG 3
3205 GOSUB 490 @ GOSUB 550 @ GOSUB 600 @ S=0 @ Q$="" @ S1$="" @ I=0
3215 GOSUB 3235 @ IF Z$#"\" THEN S1$=S1$&Z$ @ GOTO 3215
3220 GOSUB 3235 @ IF Z$="\" THEN S1$=S1$&Z$ @ GOTO 3215 ELSE IF Z$="&" THEN 3230
 ELSE S1$=S1$&Z
3225 GOSUB 3235 @ IF Z$="\" THEN 3215 ELSE IF Z$="&" THEN 3230 ELSE S1$=S1$&Z$ @
 GOTO 3225
3230 Q$=Q$&CHR$(LEN(S1$)) @ GOTO 3225
3235 I=I+1 @ IF I<=LEN(R$) THEN Z$=R$[I,I] @ RETURN ELSE POP @ R$=S1$
3240 GOSUB 640 @ S0=1 @ T=I
3245 T=SEARCH(S$,S0,T,J,1) @ IF NOT T THEN 3285
3250 S0=IP(FP(T)*1000) @ S1=S0+RMD(T*1000000,1000)-1 @ T=IP(T) @ READ #1,T;L$
3255 IF LEN(L$)-(S1-S0+1)+LEN(R$)+(S1-S0+1)*LEN(Q$)<=C2 THEN 3265
3260 PRINT FNE$("Replacement too long"); @ IF FLAG(3) THEN S1$=S$ @ GOTO 3280 EL
SE 3285
3265 S1$=R$ @ FOR I=LEN(Q$) TO 1 STEP -1
3270 S1$=S1$[1,NUM(Q$[I])]&L$[S0,S1]&S1$[NUM(Q$[I])+1] @ NEXT I @ IF D THEN S1$=
A$
3275 S2$=L$ @ GOSUB 5105
3276 IF Q THEN L$[S0,S1]=S1$ @ REPLACE #1,T;L$ @ S=S+1 @ GOSUB 5135
3280 S0=(S0+LEN(S1$)+(S2$=L$ AND NOT LEN(S1$)))*NOT L @ T=T+L @ GOTO 3245
3285 IF NOT D THEN 3290
3286 D=U @ FOR T=J TO I STEP -1 @ READ #1,T;L$
3287 IF L$=A$ THEN DELETE #1,T @ U=U-1 @ Y1=MAX(0,Y1-(T<=Y1))
3288 NEXT T @ IF D#U THEN GOSUB 470
3289 GOSUB 630 @ GOSUB 440
3290 GOSUB 710 @ PRINT FNM$(STR$(S)&" replacement(s)"); @ GOTO 1230
3300 'C25': GOSUB 490 @ PRINT FNM$("Pattern :"&S$&":"); @ GOTO 3105
```

```
3400 'C26': GOSUB 490 @ I=MAX(0,MIN(VAL(FNI$("Line #:",STR$(Y1),0,1)),U))
3435 X=X*(Y1=I) @ Y1=I @ GOSUB 625 @ RETURN
3500 'C27': PRINT FNM$("Memory: "&STR$(MEM)); @ GOTO 1230
3600 'C28': PRINT FNM$("Mark"); @ IF A<0 THEN PRINT 1; @ A=Y1 @ GOTO 1230
3605 IF A1<0 THEN PRINT 2; @ A1=A @ A=MIN(A,Y1) @ A1=MAX(A1,Y1) @ GOTO 1230
3610 PRINT "s cleared"; @ GOSUB 470 @ GOTO 1230
3700 'C29': GOSUB 740 @ GOSUB 750 @ T=Y1<A @ PRINT FNM$("Moving"); @ GOSUB 620
3710 FOR I=0 TO A1-A @ READ #1,A+I*T;L$ @ DELETE #1,A+I*T @ INSERT #1,Y1+I*T-NOT
 T;L$ @ NEXT I
3735 IF T THEN Y1=Y1+(A1-A+1)
3745 GOSUB 630 @ GOSUB 470 @ GOSUB 710 @ GOTO 1230
3800 'C30': GOSUB 740 @ GOSUB 750 @ PRINT FNM$("Copying"); @ GOSUB 620
3810 T=(Y1<A)+1 @ FOR I=0 TO A1-A @ READ #1,A+I*T;L$ @ INSERT #1,Y1+I;L$ @ NEXT
I
3835 T=T-1 @ S=A1-A+1 @ U=U+S @ A=A+S*T @ A1=A1+S*T @ Y1=Y1+S
3840 GOSUB 630 @ GOSUB 710 @ GOTO 1230
3900 'C31': GOSUB 740 @ PRINT FNM$("Delete? Y/N/Q"); @ GOSUB 1230
3910 GOSUB 615 @ GOSUB 670 @ IF P>1 THEN 1230
3915 PRINT "Deleting"; @ GOSUB 620 @ FOR I=A TO A1 @ DELETE #1,A @ NEXT I
3925 X=X*(Y1<A OR Y1>A1)
3930 Y0=Y0*(Y0<A)+MAX(0,A-(A1-Y0+1))*(A<=Y0 AND Y0<=A1)+(Y0-(A1-A+1))*(A1<Y0)
3935 Y1=Y1*(Y1<A)+MAX(0,A-(A1=U))*(A<=Y1 AND Y1<=A1)+(Y1-(A1-A+1))*(A1<Y1)
3940 GOSUB 630 @ U=U-(A1-A+1) @ GOSUB 440
3960 GOSUB 470 @ GOSUB 710 @ GOTO 1230
4000 'C32': GOSUB 910 @ POP @ GOTO 100
4100 'C33': X=LEN(L$)-POS(REV$(L$)," ",MAX(1,SPAN(REV$(L$)," ",LEN(L$)-X+1)))+1
4105 X=X*(X<=LEN(L$)) @ GOTO 1230
4200 'C34': PRINT C5$; @ GOSUB 490 @ T=0 @ GOSUB 640 @ PRINT FNM$("Erasing...");
 @ X=0
4205 FOR I=I TO J @ READ #1,I;L$ @ S=LEN(L$) @ P=1
4210 P=MEMBER(L$,M$,P) @ IF P THEN L$[P,P]="" @ GOTO 4210
4215 L$=RTRIM$(L$) @ IF S#LEN(L$) THEN REPLACE #1,I;L$ @ T=T+S-LEN(L$)
4220 NEXT I @ GOSUB 710 @ PRINT FNE$(STR$(T)&" byte(s) saved"); @ GOTO 1230
4300 'C35': GOSUB 610 @ T=POS(REV$(L$),MAPKEY$(Q$),LEN(L$)-X+1) @ IF T THEN X=LE
N(L$)-T
4310 GOTO 1230
4400 'C36': GOSUB 610 @ T=POS(L$,MAPKEY$(Q$),X+2) @ IF T THEN X=T-1
4410 GOTO 1230
4500 'C37': W=NOT FLAG(0,NOT W) @ RETURN
4600 'C38': IF Y1=U THEN RETURN ELSE R$=RTRIM$(L$) @ GOSUB 1200
4605 IF LEN(R$)+LEN(LTRIM$(L$))>=C2 THEN PRINT FNE$("Line too long"); @ GOTO 123
0
4610 L$=R$&" "&LTRIM$(L$) @ GOSUB 1300 @ GOTO 2600
4700 'C39': IF S2 THEN X=RMD((X+S2) DIV S2*S2,C1) @ X=X*(X>=S2) @ GOTO 1230
4705 IF X>=NUM(T$[LEN(T$)])-NUM(T$) THEN X=0 @ GOTO 1230
4710 FOR I=1 TO LEN(T$) @ T=NUM(T$[I])-NUM(T$) @ IF T>X THEN I=INF
4715 NEXT I @ X=RMD(T,C1) @ GOTO 1230
4800 'C40': GOSUB 490
4805 Y$=FNI$("Yank to:",Y$,16,1) @ IF NOT LEN(Y$) THEN 1230
4810 GOSUB 600 @ G=FILESZR(Y$) @ IF G>=0 THEN PURGE Y$
4815 ON ERROR GOTO 4825
4820 IF A<0 THEN COPY F$ TO Y$ @ OFF ERROR @ GOTO 4860
4825 OFF ERROR @ IF G<0 AND G#-57 THEN GOSUB 420 @ GOTO 4805
4845 CREATE TEXT Y$ @ ASSIGN #2 TO Y$ @ GOSUB 640
4850 FOR I=I TO J @ READ #1,I;L$ @ PRINT #2;L$ @ NEXT I @ ASSIGN #2 TO * @ GOSUB
 520
4860 GOSUB 670 @ GOTO 1230
4900 'C41': GOSUB 490
4905 Y$=FNI$("Insert from:",Y$,16,1) @ IF NOT LEN(Y$) THEN 1230
4920 G=FILESZR(Y$) @ IF G=0 THEN 1230
```

```
4930 GOSUB 600 @ IF G<0 THEN GOSUB 420 @ GOTO 4905
4935 ASSIGN #2 TO Y$ @ FOR I=0 TO G-1 @ READ #2;L$ @ INSERT #1,Y1+I;L$ @ NEXT I
4950 ASSIGN #2 TO * @ U=U+G @ Y1=Y1+G @ GOSUB 630 @ GOSUB 710 @ GOTO 1230
5000 'C42': GOSUB 610 @ P=POS("@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_",Q$)
5010 IF NOT P THEN FKEY Q$ @ RETURN ELSE J$=CHR$(128*H+P-1) @ GOTO 800
5100 'C43': Q$="Q:" @ SFLAG 3 @ GOSUB 3205 @ CFLAG 3 @ RETURN
5105 Q=1 @ IF NOT FLAG(3) THEN RETURN
5110 X=S0-1 @ Y1=T @ IF FNY THEN Y=Y1-Y0 ELSE GOSUB 635 @ GOSUB 710
5120 PRINT FNM$("Y/N/Q ? :"&L$[S0,S1]&": to :"&S1$&":"); @ GOSUB 1230
5130 GOSUB 615 @ GOSUB 670 @ IF P=3 THEN POP @ GOTO 3285 ELSE Q=P=1 @ RETURN
5135 IF FLAG(3) THEN PRINT C5$; @ GOSUB 510 @ PRINT L$;
5140 RETURN
5200 'C44': T=FLAG(-9,NOT FLAG(-9)) @ RETURN
9000 'VERERR': DISP "No ";V$;" LEX file." @ BEEP @ GOTO 905
9998 END SUB
9999 SUB VER(V$) @ V$="VE:1.1" @ END SUB
```

```
==========================================================================
00/09/14 17:08:50 FILE: MAKE9095   BASIC  2765 09/14/00 17:08
==========================================================================
1 ! MAKE9095 v.1.1- Make line 90 & 95 of VE
2 ! For DOCUMENTATION ONLY
3 ! O$ is a mapped (MAPKEY$) list of command keys
4 ! H$ is a mapped (MAPKEY$) list of repeating command keys
5 ! NOTE: see discussion about repeating keys in file IMPROVE
10 DIM O$[44+8],H$[9+8]
20 O$='90 O$="' @ H$='95 H$="'
100 O$=O$&CHR$(133) ! ON            exit VE
102 O$=O$&CHR$(162) ! g UP          cursor to beginning of file
104 O$=O$&CHR$(163) ! g DN          cursor to end of file
106 O$=O$&CHR$(141) ! DN            cursor down
107 H$=H$&CHR$(141)
108 O$=O$&CHR$(140) ! UP            cursor up
109 H$=H$&CHR$(140)
110 O$=O$&CHR$(150) ! g ENDLINE     forward one page
112 O$=O$&CHR$(206) ! f ENDLINE     backward one page
114 O$=O$&CHR$(137) ! LF            cursor left
115 H$=H$&CHR$(137)
116 O$=O$&CHR$(138) ! RT            cursor right
117 H$=H$&CHR$(138)
118 O$=O$&CHR$(217) ! f SPC (I/R)  toggle insert/replace mode
120 O$=O$&CHR$(205) ! f )           cursor to next word on line
121 ! H$=H$&CHR$(205) ! NOT IMPLEMENTED
122 O$=O$&CHR$(215) ! f LF (BACK)  back-space
123 ! H$=H$&CHR$(215) ! NOT IMPLEMENTED
124 O$=O$&CHR$(159) ! g LF          cursor to beginning of line
126 O$=O$&CHR$(160) ! g RT          cursor to end of line
128 O$=O$&CHR$(188) ! f H           toggle highlight mode
130 O$=O$&CHR$(128) ! ENDLINE       carriage return
132 O$=O$&CHR$(216) ! f RT (-CHAR) delete character
133 ! H$=H$&CHR$(216) ! NOT IMPLEMENTED
134 O$=O$&CHR$(191) ! f L           delete line
136 O$=O$&CHR$(219) ! f DN (-LINE) delete to end of line
138 O$=O$&CHR$(170) ! f W           delete word
140 O$=O$&CHR$(177) ! f O           open empty line
142 O$=O$&CHR$(218) ! f UP (LC)     toggle lower-/upper- case
144 O$=O$&CHR$(186) ! f F           find pattern
146 O$=O$&CHR$(172) ! f R           replace pattern
148 O$=O$&CHR$(202) ! f N           find next occurrence of pattern
150 O$=O$&CHR$(178) ! f P           go to line #
152 O$=O$&CHR$(200) ! f V           view free memory
154 O$=O$&CHR$(184) ! f S          (un)set mark(s)
156 O$=O$&CHR$(203) ! f M           move block
158 O$=O$&CHR$(199) ! f C           copy block
160 O$=O$&CHR$(185) ! f D           delete block
162 O$=O$&CHR$(155) ! g ON (OFF)    edit another file
164 O$=O$&CHR$(204) ! f (           cursor to previous word on line
165 ! H$=H$&CHR$(204) ! NOT IMPLEMENTED
166 O$=O$&CHR$(171) ! f E           erase invisible characters
168 O$=O$&CHR$(201) ! f B           cursor to previous occurrence of character on
 line
170 O$=O$&CHR$(187) ! f G           cursor to next occurrence of character on lin
e
172 O$=O$&CHR$(183) ! f A           toggle wrap-around mode
174 O$=O$&CHR$(189) ! f J           join two lines
176 O$=O$&CHR$(173) ! f T           tab
177 ! H$=H$&CHR$(173) ! NOT IMPLEMENTED
```

```
178 O$=O$&CHR$(174) ! f Y          yank to buffer/device
180 O$=O$&CHR$(176) ! f I          insert from buffer
182 O$=O$&CHR$(158) ! g RUN (CTRL) enter control characters
184 O$=O$&CHR$(169) ! f Q          conditionally replace pattern
186 O$=O$&CHR$(221) ! f Ø (USER)   toggle user mode
1ØØØ O$=O$&'"' @ H$=H$&'"'
1Ø1Ø CREATE TEXT OSTR
1Ø2Ø ASSIGN #1 TO OSTR
1Ø3Ø PRINT #1;O$ @ PRINT #1;H$
1Ø4Ø ASSIGN #1 TO *
1Ø5Ø CAT OSTR @ PLIST OSTR
1Ø6Ø TRANSFORM OSTR INTO BASIC
1Ø7Ø CAT OSTR @ PLIST OSTR
```

```
=========================================================================
00/09/14 17:09:47 FILE: XREFCMDS   TEXT   4096 09/14/00 17:09
=========================================================================
```

The following lists all functions and commands available in VE71, with references to program entry points (BASIC line number), VE75 equivalent key sequences, HP71 key sequences and keycodes (in standard HP71 format, not MAPKEY$ format).

Line#	HP75 key	HP71 key	Keycode	Function
800	-	-	-	Any alpha-numeric key.
900	TIME	ON	#43	End program.
1000	s ^	g ^	#162	Move cursor to first page.
1100	s v	g v	#163	Move cursor to last page.
1200	v	v	#51	Move cursor down one line.
1300	^	^	#50	Move cursor up one line.
1400	c v	g EOL	#150	Move cursor down one page.
1500	c ^	f EOL	#94	Move cursor up one page.
1600	<	<	#47	Move cursor left.
1700	>	>	#48	Move cursor.
1800	I/R	I/R	f<spc>	Toggle insert/replace mode.
1900	TAB	f)	f)	Move cursor to next word.
2000	BACK	BACK	#103	Destructive back_space.
2100	s <	g <	#159	Move cursor to first column.
2200	s >	g >	#160	Move cursor to last column.
2300	c I/R	f H	fH	HIGHLIGHT video.
2400	RET	EOL	#38	Return.
2500	DEL	-CHAR	#104	Delete character under cursor.
2600	CLR	f L	fL	LINE delete.
2700	s DEL	-LINE	#107	Delete from cursor to EOL.
2800	c TAB	f W	fW	WORD delete (from cursor to next word).
2900	s I/R	f O	fO	OPEN empty line above cursor.
3000	LOCK	LC	#106	Toggle upper/lower case.
3100	FETCH	f F	fF	FIND pattern.
3200	c FETCH	f R	fR	REPLACE pattern.
3300	c >	f N	fN	NEXT occurence of pattern.
3400	n/a	f P	fP	POSITION to line number.
3500	s FETCH	f V	fV	VIEW available memory.
3600	APPT	f S	fS	SELECT block of lines.
3700	EDIT	f M	fM	MOVE marked block.
3800	s EDIT	f C	fC	COPY marked block.
3900	c DEL	f D	fD	DELETE marked block.
4000	c TIME	g ON	#155	Restart editor.
4100	s TAB	f (	f(	Move cursor to previous word.
4200	s CLR	f E	fE	ERASE invisible marks.
4300	s c <	f B	fB	BACK up to previous occurence of key.
4400	s c >	f G	fG	GO to next occurence of key.
4500	c CLR	f A	fA	AUTOMATIC wrap mode toggle.
4600	c BACK	f J	fJ	JOIN following lines.
4700	n/a	f T	fT	TAB.
4800	n/a	f Y	fY	YANK to buffer.
4900	n/a	f I	fI	INSERT from buffer.
5000	n/a	CTRL	#158	CONTROL character following.
5100	n/a	f Q	fQ	QUERY replace pattern.

Mnemonic alpahabetical characters used by commands:

A   AUTOMATIC wrap toggle.

```
B       BACK up to key.
C       COPY block.
D       DELETE block.
E       ERASE invisible characters.
F       FIND pattern.
G       GO to key.
H       HIGHLIGHT video toggle.
I       INSERT from buffer.
J       JOIN lines.
K       ---------------- not used.
L       LINE delete.
M       MOVE block.
N       NEXT occurence of pattern.
O       OPEN line.
P       POSITION to line number.
Q       QUERY replace pattern.
R       REPLACE pattern.
S       SELECT block.
T       TAB.
U       ---------------- not used.
V       VIEW available memory.
W       WORD delete.
X       ---------------- not used.
Y       YANK to buffer.
Z       ---------------- not used.
================================================================================
00/09/14 17:10:39 FILE: XREFVE     TEXT   2816 09/14/00 17:10
================================================================================
```

This file contains a cross-reference of VE by line. Much of this information - but not all - is duplicated in the commentaries of each routine, for the programmer's convenience.
DO check this table before moving things around!
All inter-routine references are considered as entry points and are identified with an 'e' followed by a number.

```
e1   100: 4000
     130: 155
     135: 130
     155: 130
     215: 210
     255: 260
e2   265: 255,290,295
e3   275: 265,285
e4   290: 255
     295: 295
     305: 330,335
     325: 310
     335: 325
     340: 330
e5   420: 330,4825,4930
e6   430: 205
e7   431: 55
e8   440: 2610,3288,3940
e9   470: 100,2405,2440,2600,2905,3610,3745,3960
e10  490: 480,740,1000,1100,1200,1300,1400,1500,2400,2405,2430,2700,2900,3100,
          3205,3300,3400,4200,4800,4900
e12  510: 1210,1310,5135
e13  520: 250,625,710,1220,1320,2615,2905,3112,4850
e14  525: 2615,2905 (lines 525 and 530 could be joined)
```

```
e15  540: 1900,2800
e16  550: 3100,3205
e17  600: 3205,4810,4930
e18  610: 4300,4400,5000
e19  615: 3910,5130
e20  620: 3700,3800,3915
e21  625: 3115,3435
e22  630: 3288,3745,3840,3940,4950
e23  635: 625,5110
e24  640: 3240,4200,4845
e25  670: 305,350,355,1210,1310,3910,4860,5130
e26  710: 340,625,1010,3290,3745,3840,3960,4220,4950,5110
e27  740: 3700,3800,3900
e28  750: 3700,3800
e29  800: 266,5010
     840: 801
e30  905: 200,431,9000
e31  910: 900,4000
e32  915: 200,430
e33 1000: 250
e34 1005: 2610
e35 1006: 1400,1500
e36 1010: 1100
e37 1200: 4600
e38 1210: 2425,2430,2445
e39 1230: 1010,1330,1900,2100,2200,2435,2505,2615,2700,2805,2905,3112,3290,3500,
          3600,3605,3610,3745,3840,3900,3960,4105,4220,4310,3910,4410,4605,4700,
          4705,4715,4805,4860,4905,4920,4950,5120
e40 1300: 4610
e41 1600: 841,842,2005
e42 1800: 844,845
e43 1805: 345
e44 2200: 845
e45 2400: 845
    2430: 2415
    2440: 2400
e57 2505: 843
e46 2600: 4610
e47 2905: 2430
e48 3005: 100
e49 3105: 3300
    3115: 3105
e50 3205: 5100
    3215: 3215,3220,3225
    3225: 3225,3230
    3230: 3220,3225
    3235: 3215,3220,3225
    3245: 3280
    3265: 3255
    3280: 3260
e51 3285: 3245,3260,5130
    3290: 3285
e52 4100: 842
    4210: 4210
    4805: 4825
    4825: 4815
    4860: 4820
    4905: 4930
e53 5105: 3275
e54 5135: 3276
```

```
e55 9000: 40,45,50,51
e56 9998: 905
================================================================================
00/09/14 17:11:27 FILE: VEVARS     TEXT   5632 09/14/00 17:11
================================================================================
This is not a cross-reference of VE by variable. It is instead a list of all
relevant variables of the program. Together with each variable you will find a
classification of the data type it represents, and a description of the purpose
of the variable.
The following symbols will be used in classifying data types:
$  : string variable
$c : string constant
$p : string subprogram parameter
c  : character variable (1 byte)
cc : character constant (1 byte)
i  : integer variable
ic : integer constant
r  : real variable
rc : real constant
np : numeric subprogram parameter
Name Type Description
A    i    1st mark (A=-1:unset, 0<=A<=U set)
A1   i    2nd mark (A1=-1:unset, 0<=A1<=U set)
A$   $c   holds ">line deleted<" for messages and REPLACE#
C0   np   # of columns of display device (ex. C0=32 for HP82163)
C1   np   # of columns of window (C1<=C0 or C0<C1<=142)
C2   ic   max window column # (C2=C1)
C3   ic   max w. column in screen coordinates (0<=C3=C2-1)
C4   ic   right-margin bell limit (C4=C3-10)
C$   $c   escape sequence for clearing to bottom of screen
C0$  $c   a string of blanks used to (partially) clear a line
C1$  $c   escape sequence to move cursor right
C2$  $c   escape sequence to move cursor down
C3$  $c   escape sequence to move cursor left
C4$  $c   escape sequence to turn cursor on
C5$  $c   escape sequence to turn cursor off
C6$  $c   escape sequence to select replace cursor
C7$  $c   escape sequence to select insert cursor
D    i    delete-line flag (in replacements) + scratch
D1   rc   holds the status of flag -3 (enable battery timeout)
D$   $p   display device specifier
E$   $c   escape sequence to clear screen page. NOTE: in MKVEDB the default
          value for E$ is "cursor home+clear to bottom of screen". This choice
          was dictated by speed considerations when redrawing (especially with
          the PACSCREEN). On the other hand a "clear display device" sequence
          (ESC E) could be used. This would slow down I/O operations, but it
          would reduce the amount of garbage entering the screen when scrolling.
E1$  $c   escape sequence to clear display device (see above)
F$   $c   name of current edit file
G    i    scratch + size of buffer file (in records)
G$   $c   default buffer file name ("BUF")
H    i    highlight flag
H$   $c   list of repeating keys (in MAPKEY$ format)
I    i    scratch + lower block boundary
J    i    scratch + upper block boundary
J$   c    at any time the last TEXT key handled (highlighted and MAPKEYed)
K    i    at any time "C"&STR$(K) is the last command key routine executed (K=0
          means routine 800 (text key handler)
K$   $    at any time K$ is the last key pressed (in KEYWAIT$ format)
K1$  c    at any time K1$ is the last key pressed (in MAPKEY$ format)
```

L	i	one-replacement-per-line flag
L$	$	usually the current line (CL), seldom used for line operations
M0	ic	the row # of the message line (in screen coordinates)
M$	$c	a list of invisible characters
N$	$	(when USER active) the key definition (with parameters), see MKVEKEYS
O$	$c	a list of command keys (in MAPKEY$ format)
P	i	scratch (usually for POS())
Q	i	scratch (usually do-replacement flag)
Q$	$	scratch, list of dittoes (in replacements), key-press
R	i	I/R flag (R=1 in replace mode)
R0	np	# of rows of display device (ex. R0=16 for HP82163)
R1	np	# of lines of window including message line (R1=R0 & C1<=C0 ¦ R1=2 & C1>C0)
R$	$	replacement string + scratch
R0$	$c	escape sequence for scrolling down one row
R1$	$c	escape sequence for scrolling up one row
S	i	scratch
S0	i	scratch + start of match (in replacements)
S1	i	scratch + end of match (in replacements)
S2	ic	value of tab-stop width (S2=0: absolute tab-stops in T$)
S$	$	at any time S$ is the last search pattern specified
S1$	$	scratch
S2$	$	scratch
T	r	scratch (must be real)
T1	i	scratch
T$	$p	tab-stop width or list of absolute tab-stops
U	i	at any time U points to the last record of the edit file
U1$	$c	holds system status previous to running VE
U4$	$c	holds system status previous to running VE
U5$	$c	holds system status previous to running VE
V$	$	version id
W	i	wrap-around flag (W=0: w-a enabled)
W9	i	scratch, used to hold R in text key handler
X	i	cursor column in screen coordinates (0<=X<=C3)
Y	i	cursor row in screen coordinates (0<=Y<M0)
Y0	i	record # of line corresponding to current TOS (Top Of Screen)
Y1	i	record # of current line (before any command Y1=Y0+Y)
Y$	$	name of current buffer file/device
Z$	c	used for parsing replacement string

```
================================================================================
00/09/14 17:12:39 FILE: ABBREV     TEXT    256 09/14/00 17:12
================================================================================
The following abbreviations will be used throughout the documentation:
CL  : current line (where the cursor is)
BOL : beginning of line
EOL : end of line
BOF : beginning of file
EOF : end of file
TOS : top of screen
BOS : bottom of screen
================================================================================
00/09/14 17:12:43 FILE: P0000      TEXT   3840 09/14/00 17:12
================================================================================
```

```
Subprogram VE - version 1.1 - Visual editor
This program was ported from a similar program for the HP75.
Necessary hardware: HP71, HPIL display device (HP82163, PACSCREEN were tested;
other interfaces - compatible with HP escape sequences - should work just
fine), memory modules.
Necessary software:
   at run-time: EDLEX lex file, CUSTUTIL lex (or any lex file containing
   KEYWAIT$), STRINGLX lex file, VELEX lex file, FKEYLEX lex file,
   VEDB data file, VEFOLD basic file (optional)
   at set-up time: MKVEDB basic file, MKVEKEYS basic file.
```

```
1 ! VE Copyright (C) Stefano Piccardi & Stefano Tendon, 1986
   R0,C0: # of rows and # of colums of video interface (ex. R0=16 C0=32 for
   HP82163 video interface).
   R1,C1: # of lines (including message line) and # of columns of
   working window; (2<=R1<=R0 AND C1<=C0 OR R1=2 AND C0<C1<=142). PACSCREEN
   users: don't set C1=80 due to h/w bug (use C1=79).
   D$: display device specifier (ex. :DISPLAY).
   T$: tab-stop specifier: use the string representation of a number for
   relative addressing (tab-stop width) (ex. "8") or use a string of characters
   - where each ASCII code means a column # - for absolute addressing (ex.
   "AHOZ" for the FORTH assembler).
10 SUB VE(R0,C0,R1,C1,D$,T$)
   disable ATTN key
15 POKE "2F441","F"
   save status of battery timeout flag and disable timeout (otherwise key #99
   could be inserted into edit file if VE is running and the HP71 timeouts)
   @ D1=FLAG(-3,1)
   see VEVARS for variable definition and usage
20 DIM C1$[2],C2$[2],C3$[2],C4$[2],C5$[2],C6$[2],C7$[2],C$[2],E$[4],E1$[2],
   C0$[160],V$[6]
25 DIM R0$[2],R1$[2],F$[16],L$[160],S$[150],S1$[150],S2$[150],R$[150],Z$[1],
   Y$[16],G$[16]
30 DIM U1$[14],U4$[4],U5$[9],O$[44],M$[255], N$[97],K$[4],K1$[1],Q$[32],
   H$[4],J$[1]
35 INTEGER S,S0,S1,Y,Y0,Y1,A,A1,I,C2,C3,C4,W,R,D,P,K,L,G,H, J,M0,Q,S2,
   T1,U,W,W9,X
   check if lex files are available (this only works for lex files answering
   to the version poll)
   VELEX
40 V$="VE:1.1" @ IF NOT POS(VER$,V$) THEN 'VERERR'
   STRINGLX
45 V$="STR:A" @ IF NOT POS(VER$,V$) THEN 'VERERR'
   EDLEX
50 V$="EDT:A" @ IF NOT POS(VER$,V$) THEN 'VERERR'
```

```
   CUSTUTIL (used only for KEYWAIT$, but could also be used for KEYNUM and
   KEYNAM$ [see IMPROVE file])
51 V$="CSTU:A" @ IF NOT POS(VER$,V$) THEN 'VERERR'
   load V$ with VE version number
55 CALL VER(V$)
   on error report error, restore system status and end VE (off error on line
   205)
   @ ON ERROR GOTO 431
   look for data base
60 ASSIGN #1 TO VEDB
   set tab-stop flag/width
   @ S2=VAL('0'&T$)
   transform R0 into screen coordinates
70 R0=R0-1
   subtract one (message) line from R1 and transform it in screen coord's or
   force one editable line if C1>C0, that is if lines must be folded
   @ R1=(R1-2)*(C1<=C0)
   compute row # (screen coord's) of message line
   @ M0=(R1+1)*CEIL(C1/C0)
   I suggest to let this (unecessary) variable live
   @ C2=C1
   C2 in screen coord's
   @ C3=C2-1
   margin
   @ C4=C3-10
   buffer to erase to closest screen right side
80 C0$=RPT$(" ",C0*CEIL(C1/C0))
   load escape sequences from data base (this db will allow future extensions
   to other interfaces [recoding CUR$()] - as a matter of fact, VE has already
   been ported to an HP150 Touchscreen)
   @ READ #1;M$,C1$,C2$,C3$,C4$,C5$,C$,R0$,R1$,C6$,C7$,E1$,E$
85 A$=">line deleted<"[1,C2]
   see MAKE9095 file
   command keys
90 O$=...
   repeating keys
95 H$=...
==================================================================================
00/09/14 17:13:45 FILE: P0100      TEXT   3584 09/14/00 17:13
==================================================================================
ENTRY #1: restart editor
    default buffer file name (Y$ might be changed at run-time)
100 Y$="BUF" @ G$=Y$
    init key definition (NUM is performed on N$ even if not USER)
    @ N$=""
    init wrap-around flag to true
    @ W=0
    init I/R flag to replace
    @ R=1
    init highlight flag to false
    @ H=0
    ?
    @ Q=0
    init annunciators
    @ CFLAG 0,1,2,3
    init marks to unset
    @ GOSUB 470
    init annunciator of upper-/lower- case
    @ GOSUB 3005
    save DISPLAY IS and PRINTER IS device specifiers
```

```
105 U1$=PEEK$("2F78D",14)
    save DELAY and SCROLL rates
    @ U4$=PEEK$("2F946",4)
    save PWIDTH and ENDLINE
    @ U5$=PEEK$("2F958",9)
    all output to D$ (display for LINPUT, printer for VELIST and PRINT)
120 DISPLAY IS D$ @ PRINTER IS D$
    for input and CAT
    @ DELAY 0,0
    @ PWIDTH INF @ ENDLINE
    for STR$
    @ STD
    turn off cursor + clear screen page
130 PRINT C5$;E$;
    delete the following lines to remove the run-time copyright notice
    @ GOSUB 135 @ GOTO 155
135 PRINT "Visual Editor - ";V$
    @ PRINT "  Copyright (C) 1986" @ PRINT
140 PRINT "Stefano Piccardi"
    @ PRINT "Via A. Panizzi 13"
    @ PRINT "20146 MILANO, Italy" @ PRINT
145 PRINT "Stefano Tendon"
    @ PRINT "Cantone Delle Asse 5"
    @ PRINT "29100 PIACENZA, Italy"
150 RETURN
    get edit file name without redrawing screen; user can end VE by keying in
    '*' ENDLINE and disregarding the error message
155 F$=FNI$("File:","",16,0)
    turn off cursor + clear display device (not just screen page)
    @ PRINT C5$;E1$;
    if user didn't supply a name then catalog all files (pressing arrow keys)
    then retry input (pressing any other key)
    @ IF NOT LEN(F$) THEN CAT ALL @ GOTO 130
    init line counter with size of edit file
195 U=FILESZR(F$)
    if file doesn't exist then create it and init line counter to ONE line (U
    points to the last record and zero is a valid record number)
    @ IF U=-57 THEN CREATE TEXT F$ @ U=0
    if file name was invalid then complain and end VE
200 IF U<0 THEN PRINT C4$; @ GOSUB 915 @ BEEP @ DISP MSG$(ABS(U)) @ GOTO 905
    trap errors involving channels or data base (off error on line 210)
205 ON ERROR GOTO 430
    edit file is #1
    @ ASSIGN #1 TO F$
    if new file then insert one empty record (VE always leaves files with at
    least one record)
    @ IF U=0 THEN PRINT #1;""
    else make U point to last file line (since FILESZR returns the NUMBER of
    rec's in the file; this is also the reason why U is often referred to as
    'line counter' while it should really be called 'pointer to last line')
    ELSE U=U-1
    ignore missing sub
210 ON ERROR GOTO 215
       if file VEFOLD is in RAM then execute subroutine to fold lines which
       don't fit within working window (SUB VEFOLD used to be in VE, but
       since it is slow and seldom necessary it was taken off; the user then
       can simply PURGE it or RENAME it when it isn't needed)
       @ CALL VEFOLD(#1,U,C2) IN VEFOLD
215 OFF ERROR
    *** VE traps errors only locally (inside routines) not globally ***
    init search pattern (necessary for [f][N])
250 S$=""
    ?
    @ R$=""
    init pointer to TOS and screen y-coordinate
    @ Y=0 @ Y0=0
    init CL pointer, fetch CL (1st) line
    @ GOSUB 520
    display first page
    @ GOSUB 1000
================================================================================
00/09/14 17:14:43 FILE: P0255      TEXT   3328 09/14/00 17:14
================================================================================
--------------------------------------------------------------------------------
Main loop - key process
This is were VE spends most of its time (but not of its energies)!
MODIFIED: K$,K1$,N$ (if USER),K
--------------------------------------------------------------------------------
    loop forever
       get a key
255    K$=KEYWAIT$
       encode key using a unique one-byte code
       @ K1$=MAPKEY$(K$)
       if USER then process_user
       @ IF FLAG(-9) THEN GOSUB 290
       else process_code
       ELSE GOSUB 265
    endloop
260 GOTO 255
ENTRY #2: process_code: INPUT: K1$ (code)
    put position of key code within list of command keys into K
265 K=POS(O$,K1$)
    if key code doesn't represent a command key then
    @ IF K THEN 275
       if key isn't prefixed ([f]/[g])
266    IF NUM(K1$)<128 THEN
          then it's a text key: K1$ is 1-byte long and already case-dependent
          (upper/lower); set high bit if highlight mode is active
          NOTE: if highlight mode is seldom used, then it may pay off to
          recode the [f][H] command in a way similar to [g][RUN] (highlight
          next key only) and simplify this statement into J$=K1$ (DO use J$:
          it's a uniquely interface with the text key handler). [g][RUN] should
          then be modified to handle the new highlight mode.
          J$=CHR$(128*H+NUM(K1$))
          and pass byte to text key handler
          @ GOSUB 800
       else complain (unknown command)
       ELSE CHIRP
270    RETURN
    else
ENTRY #3: execute a (repeating) command key
       algorithm: repeat
                     execute key
                  until key not repeating or key up
       execute code for K-th command key
275    GOSUB "C"&STR$(K)
       if it is a repeating command key...
280    IF NOT POS(H$,K1$) THEN RETURN
          then if key is still down
          NOTE: see file IMPROVE to learn more about repeating keys
```

```
285       IF KEYDOWN(K$)
             then execute code for K-th command key again (possibly repeating)
             THEN 275
          else return (possibly after many repetitions)
          ELSE RETURN
    endif
ENTRY #4: process_user: INPUT: K$
    fetch key definition (possibly none) into N$
290 N$=KEYDEF$(K$)
    if key is not defined or definition isn't a typing-aid
    @ IF NUM(N$)#59
       then process_code (even if USER)
       THEN 265
    else
    ELSE
       trim typing-aid identifier ';'
       N$[1,1]=""
       clear key buffer
       NOTE: this is for robustness when handling repeating keys; we don't want
       a spurious KEYDOWN on line 285
       @ K$=""
       while definition not empty
295    IF LEN(N$) THEN
          get 1 byte from encoded definition
          K1$=CHR$(NUM(N$))
          remove code from definition
          NOTE: other code relies on this trimming (FNI$, FNK$)
          @ N$[1,1]=""
          process_code
          @ GOSUB 265
       endwhile
       @ GOTO 295
    endif
    ELSE RETURN
==============================================================================
00/09/14 17:15:32 FILE: P0300      TEXT   5376 09/14/00 17:15
==============================================================================
------------------------------------------------------------------------------
Function FNI$ - perform input
Whenever input is requested use this function and nothing else.
ASSERT ON ENTRY: this code relies on the fact that no more than 96 characters
                 can be entered from the keyboard.
INPUT: P$: prompt; D$: default return value; L: maximum length of return
       string (<=96), if L=0 then request to return a valid numeric
       expression (VAL is used for testing); F: redraw flag (F=1 means redraw),
       use F=0 whenever screen parameters are invalid.
OUTPUT: a string containing user's input. Key definitions with parameters are
accounted for. Errors are trapped.
MODIFIED: N$ (if USER)
TRASHED: T,R$,G (on input error)
------------------------------------------------------------------------------
300 DEF FNI$[96](P$,D$,L,F)
    erase message line
305 GOSUB 670
    turn cursor on and put out prompt
    @ PRINT C4$;P$;
    if no parameter is pending in a key definition (notice that USER off
    implies N$="" => NUM(N$)=0)
    NOTE: ASCII 220 is used as a delimiter for parameters in key definitions
    since MAPKEY$ can't return it.
```

```
310 IF NUM(N$)#220 THEN
        do input
        LINPUT "",D$;R$
        disable ATTN key (this is necessary when user turns off the HP71 during
        input)
        @ POKE "2F441","F"
     else
     @ GOTO 325
       compute end of parameter in key definition
315    T=POS(N$,CHR$(220),2)
       load return string with parameter
       @ R$=N$[2,T-1]
       trim parameter off key definition
       @ N$=N$[T+1]
       if parameter = null string then load return string with default return
       value
       @ IF NOT LEN(R$) THEN R$=D$
       put out return string to fake keyboard input
320    PRINT R$;
    endif
    check validity of input:
325 ON ERROR GOTO 335
    if requested to return a numeric value then test for numeric
       @ IF NOT L THEN T=VAL(R$)
330 OFF ERROR
    else {return string} if string is not too long then return value
    @ IF (L#0)*LEN(R$)<=L THEN 340
    else put out 'string too long' and retry
    ELSE G=37 @ GOSUB 420 @ GOTO 305
    endif
    put out 'bad argument' and retry (VAL failed)
335 OFF ERROR @ PRINT FNE$(ERRM$); @ GOTO 305
    if redraw-screen flag then redraw screen
340 IF F THEN GOSUB 710
    restore I/R cursor
345 GOSUB 1805
    restore upper-/lower- case lcd flag
    @ F=FLAG(4,FLAG(-15))
    return value
    @ FNI$=R$ @ END DEF
```

```
Function FNE$ - display error message
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT: message line is clear
INPUT: M$: message; FLAG -15: inhibit WAIT
OUTPUT: null string; intended usage PRINT FNE$("message");
MODIFIED:
TRASHED:
```

```
350 DEF FNE$(M$)
    put out message (making sure it fits within the window)
    @ PRINT FNM$(M$);
    beep (and waste some time)
    @ BEEP
    waste 1 second but only if not USER (this comes handy especially in key
    definitions involving many [f][J]'s)
    @ WAIT NOT FLAG(-9)
    erase message line
    @ GOSUB 670
    return dummy value (logically this is a procedure)
    @ FNE$="" @ END DEF
```

```
Function FNM$ - display message
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT: message on screen
INPUT: M$: message
OUTPUT: null string; intended usage PRINT FNM$("message");
MODIFIED:
TRASHED:
```

```
355 DEF FNM$(M$)
    erase message line
    @ GOSUB 670
    put out message making sure it fits within the window
    @ PRINT M$[1,C2-1];
    return dummy value (logically this is a procedure)
    @ FNM$="" @ END DEF
```

```
Function FNY - check if CL pointer points inside current page
INPUT: Y1
OUTPUT: 1 if Y1 points inside page, otherwise 0
```

```
360 DEF FNY=Y0<=Y1 AND Y1<=Y0+R1
```

```
Function FNK$ - return key pressed.
Whenever waiting for a key use this function and nothing else
ASSERT ON ENTRY: it can handle a list of key presses in a key definition
ASSERT ON EXIT:
INPUT: none
OUTPUT: key
MODIFIED: N$ (if USER)
```

```
370 DEF FNK$
    if no parameter pending in a key definition (see FNI$)
    @ IF NUM(N$)#220 THEN
       return keyboard input
       FNK$=KEYWAIT$
    else
    @ GOTO 374
       extract key from list of key presses in key definition
372    FNK$=N$[2,2]
       if that wasn't the last item of the list
       @ IF NUM(N$[3])#220 THEN
          then trim first item from list
          N$[1,2]=CHR$(220)
       else
       ELSE
          remove empty list from key definition
          N$[1,3]=""
       endif
    endif
374 END DEF
================================================================================
00/09/14 17:17:02 FILE: P0400      TEXT   6400 09/14/00 17:16
================================================================================
```

```
Routines - a huge collection of routines to do mostly everything
ENTRY POINTS: (see XREFVE) #5-#28
```

```
ENTRY # 5: print error message # G
```

```
420 PRINT FNE$(MSG$(ABS(G))); @ RETURN
ENTRY #6: display last error message and end VE
          ASSERT ON ENTRY: display and printer have already been assigned
430 GOSUB 915
ENTRY #7: display last error message and end VE
          ASSERT ON ENTRY: display and printer haven't been assigned yet
431 BEEP @ DISP ERRM$ @ GOTO 905
ENTRY #8: if file has been depleted then append one empty record and adjust
          pointer to last line (U)
440 IF U<0 THEN U=0 @ RESTORE #1,0 @ PRINT #1;""
441 RETURN
ENTRY #9: set both marks to 'undefined'
470 A=-1 @ A1=-1 @ RETURN
ENTRY #10: store CL in file iff it isn't already there (to avoid a lengthy
           operation)
490 READ #1,Y1;S1$ @ IF S1$#L$ THEN REPLACE #1,Y1;L$
495 RETURN
ENTRY #12: erase line Y (usually CL) from screen
510 PRINT CUR$(Y,0);C0$;CUR$(Y,0); @ RETURN
ENTRY #13: given Y,Y0, update CL pointer (Y1) and fetch CL in L$
520 Y1=Y0+Y
    erase previos contents of L$
    NOTE: this assignment is probably useless. It comes from the HP75 version
    of VE. Anyway, neither does it do any bad, nor does it affect speed sensibly
    @ L$=C0$
    fetch CL and return
    @ READ #1,Y1;L$ @ RETURN
ENTRY #14: redraw lower portion of screen (lower with respect to CL);
           usually called by [f][O],[f][L] and other line operations
525 PRINT CUR$(Y,0);
530 PRINT C$; @ VELIST #1,Y1,Y0+R1 @ RETURN
ENTRY #15: compute position of next word on current line; OUTPUT: T
    from the cursor position (X+1) search first space (POS(" ",) and from there
    - ignoring trailing spaces (RTRIM) - or from the beginning of the line - if
    there are no spaces - search for first non-blank character (SPAN) or stop
    at first column if such a character doesn't exist (MAX)
540 T=MAX(0,SPAN(L$," ",MAX(1,POS(RTRIM$(L$)," ",X+1)))-1) @ RETURN
ENTRY #16: do input and parse it for search and replace
           OUTPUT: S$,R$,D,L
           MODIFIED: N$ (if USER)
           TRASHED: Q$,P,I,J,S1,G (on input error),T
    init search pattern
550 S$=""
    do input
    @ R$=FNI$(Q$,"",96,1)
    if nothing useful was entered then take control, display cursor and exit
    @ IF NOT SPAN(R$," ") THEN POP @ GOTO 1230
    put delimiter into Q$
570 Q$=CHR$(NUM(R$))
    don't be too sensitive to missing closing delimiters: add yours
    @ R$=RTRIM$(R$)&Q$
    parse search pattern into S$
    @ P=POS(R$,Q$,2) @ S$=R$[2,P-1]
    trim it from input string
    @ R$=R$[P+1]
    find where replace string ends
580 P=POS(R$,Q$)
    set one-replacement-per-line flag if user entered anything after the third
    delimiter
    @ L=LEN(R$)>P+1
    parse replace string
    @ R$=RTRIM$(R$[1,P],Q$)
    compute delete-line flag (\^ and \$ in S$ and R$="")
    init flag
590 D=0
    init 'special-characters-active' flag to false
    @ I=0
    foreach character in search pattern
    @ FOR J=1 TO LEN(S$)
       @ S1=NUM(S$[J])
       if it's a '\' then toggle s-c-a flag
591    IF S1=92 THEN I=NOT I
       else if it's the first '^' and s-c-a then D='\^ found'
       ELSE IF S1=94 AND I AND D=0 THEN D=1
       else if it's '$' preceded by '^' and s-c-a then D='\$ found'
       ELSE IF S1=36 AND D=1 AND I THEN D=2
    endfor
592 NEXT J
    set D to true iff D=2 AND null replace string
    @ D=D=2 AND NOT LEN(R$)
    NOTE: this code ignores special '^'s following the first special '^'
    because that's exactly what the SEARCH keyword does; it treats exceeding
    special '^'s as literal '^'s and so do we.
    @ RETURN
ENTRY #17: put 'Working...' message on message line
600 PRINT FNM$("Working..."); @ RETURN
ENTRY #18: get a text key and return or don't return at all
           OUTPUT: Q$
           MODIFIED: N$ (if USER)
    get that key
610 Q$=FNK$
    if it's a prefixed key then take control, put it back in the key buffer
    NOTE: in a key definition with parameters any byte can be used and will
    be returned
    @ IF LEN(Q$)#1 THEN FKEY Q$ @ POP
611 RETURN
ENTRY #19: wait until Y/N/Q is pressed and return P
615 P=POS("YNQ",UPRC$(FNK$)) @ IF NOT P THEN CHIRP @ GOTO 615 ELSE RETURN
ENTRY #20: put out message '(size of block) lines'
620 PRINT A1-A+1;"line(s)..."; @ RETURN
ENTRY #21: fetch CL and if it's outside of current page update pointers and
           redraw screen giving some context to the user
           INPUT: Y1
           OUTPUT: L$
           MODIFIED: Y,Y0
    if CL pointer is within page then update cursor y-coordinate, fetch CL
625 IF FNY THEN Y=Y1-Y0 @ GOSUB 520
    else adjust Y,Y0, fetch CL and redraw screen
    ELSE GOSUB 635 @ GOSUB 710
    display cursor and exit
626 GOSUB 1230 @ RETURN
ENTRY #22: similar to entry #21, but CL is not fetched and screen is not
           refreshed
630 IF FNY THEN Y=Y1-Y0 @ RETURN
ENTRY #23: also part of entry #22
    adjust Y,Y0 so that Y1 is found at 1/4 of the screen page
635 Y0=MAX(0,Y1-(R1 DIV 4)) @ Y=MIN(Y1,R1 DIV 4) @ RETURN
ENTRY #24: compute 'lose' block boundaries
           INPUT: A,A1
           OUTPUT: I,J
           ASSERT ON EXIT: 0<=I<=J<=U
    if 1st mark set then I=1st mark else I=0
640 I=A*(A#-1)
    if 2nd mark set then J=2nd mark else J=U (last line)
    @ J=U*(A1=-1)+A1*(A1#-1)
    @ RETURN
ENTRY #25: clear message line AND bottom of screen
670 PRINT C5$;CUR$(M0,0);C$; @ RETURN
ENTRY #26: fetch CL and redraw screen
710 GOSUB 520 @ PRINT C5$;E$; @ VELIST #1,Y0,Y0+R1 @ RETURN
ENTRY #27: store CL, check if both marks are set, if not then take control and
           error out
740 GOSUB 490 @ IF A>=0 AND A1>=0 THEN RETURN
745 POP @ PRINT FNE$("Missing mark(s)"); @ GOTO 1230
ENTRY #28: check if CL is inside block, if so take control and error out
750 IF Y1<A OR Y1>A1 THEN RETURN
755 POP @ PRINT FNE$("Inside block"); @ GOTO 1230
================================================================================
00/09/14 17:18:40 FILE: P0800      TEXT   3072 09/14/00 17:18
================================================================================

--------------------------------------------------------------------------------
Routine: text key handler
ENTRY POINTS: #29 (266,5010)
ASSERT ON ENTRY: case and high bit are already set
ASSERT ON EXIT:
INPUT: J$,R,W
OUTPUT:
MODIFIED: L$,X,(FILE,Y1,possibly Y,Y0 (if W & X=C2))
TRASHED: W9,T
--------------------------------------------------------------------------------
This routine handles text keys. Wrap-around and I/R modes
cause different actions:
W I  Action
0 0  replace cursor with key, if not at right margin then
     advance cursor else complain
1 0  just like W=0 I=0, but if at right margin do wrap-around in insert mode
0 1  if room left then insert key under the cursor else complain
1 1  just like W=0 I=1, but if at right margin do wrap-around (in insert mode)
In addition, this routine handles a right-margin bell,
which is set at a constant distance from the screen right
margin. The bell is unsatisfactory for two reasons: it is
not similar to a MARGIN statement, it can't be turned off
unless BEEP is turned OFF or line 800 is deleted.
ENTRY #29:
    ring right-margin bell if necessary
800 IF X=C4 THEN BEEP 3500,.075
    if replace mode...
801 IF R THEN
        then advance cursor logically
        X=X+1
        place key under PREVIOUS cursor position
        @ L$[X,X]=J$
        display key
        @ PRINT J$;
    else (insert mode)
    @ GOTO 840
        if no room left then complain and exit
810     IF LEN(L$)=C2 THEN CHIRP @ RETURN
        advance cursor
820     X=X+1
```

```
       insert new key
       @ L$[X,0]=J$
       turn cursor off, display new key + right portion of line, display cursor
830    PRINT C5$;L$[X];CUR$(Y,X,C0);C4$;
    endif
    if at right margin...
840 IF X#C2 THEN RETURN
       then if no wrap-around (W=1)...
841    IF W THEN
          then complain
          CHIRP
          back cursor and exit
          @ GOTO 1600
       else (wrap-around active)
          signal about to split line
842       BEEP 100,.01
          if not on a ' ' then
          @ IF NUM(L$[X])#32 THEN
             go to beginning of previous word
             GOSUB 4100
             and back cursor one column (possibly to a ' ')
             @ GOSUB 1600
          if on a ' ' then delete it (this test is
          necessary for lines without blanks)
843       IF NUM(L$[X+1])=32 THEN GOSUB 2505
          save I/R flag status
844       W9=R
          and set insert mode
          @ IF R THEN GOSUB 1800
          do a [RTN] in insert mode
845       GOSUB 2400
          go to end of new line
          @ GOSUB 2200
          restore previous I/R status
          @ IF W9 THEN GOSUB 1800
          signal end of wrap-around
850       BEEP 100,.01
          and exit
          @ RETURN
       endif (wrap-around)
    endif (at right margin)
=================================================================================
00/09/14 17:19:25 FILE: P0900      TEXT   1280 09/14/00 17:19
=================================================================================
```

```
Command key [ON] - exit editor
ENTRY POINTS: #31 (900,4000), #32 (200,430)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: FLAGS 0-4,-3
TRASHED: S1$
```

```
    restore system and loop status
900 'C1': GOSUB 910
    purge default buffer file if it exists
    @ IF FILESZR(G$)>=0 THEN PURGE G$
901 CAT F$
    restore battery timeout status
```

```
905 T=FLAG(-3,D1)
    enable ATTN key
    @ POKE "2F441","0"
    end subprogram
    @ GOTO 9998
ENTRY #31:
    store current line and display replace cursor + clear screen
910 GOSUB 490 @ PRINT C6$;E$;
ENTRY #32:
    restore original DISPLAY IS and PRINTER IS devices
915 POKE "2F78D",U1$
    restore original SCROLL and DELAY rates
    @ POKE "2F946",U4$
    restore original PWIDTH and ENDLINE
    @ POKE "2F958",U5$
925 CFLAG 0,1,2,3,4
    @ RESTORE IO
    @ RETURN
================================================================================
00/09/14 17:19:43 FILE: P1000      TEXT   1792 09/14/00 17:19
================================================================================

--------------------------------------------------------------------------------
Command key [g][UP] - display first page
ENTRY POINTS: #33 (250), #34 (2610), #35 (1400,1500), #36 (1100)
ASSERT ON ENTRY:
ASSERT ON EXIT: Y=0 & X=0 (not for #36)
INPUT: see entry point #33
OUTPUT:
MODIFIED: X,Y,Y0,Y1,L$
TRASHED: S1$
--------------------------------------------------------------------------------
ENTRY #33: INPUT: none
     store current line
1000 'C2': GOSUB 490
ENTRY #34: same as #33
     place first line of file on TOS
1005 Y0=0
ENTRY #35: INPUT: Y0
     cursor home
1006 Y=0 @ X=0
ENTRY #36: INPUT: X,Y,Y0
     update line pointer, fetch current line, redraw screen
1010 GOSUB 710
     display cursor and exit
     @ GOTO 1230
================================================================================
00/09/14 17:20:02 FILE: P1100      TEXT   1792 09/14/00 17:20
================================================================================

--------------------------------------------------------------------------------
Command key [g][DN] - display last page
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT: X=0 & ((U=0 & Y0=0 & Y=0) ! (1<=U<=R1 & Y0=0 & Y=U) !
                (U>R1 & Y=R1))
INPUT:
OUTPUT:
MODIFIED: X,Y,Y0,Y1,L$
TRASHED: S1$
--------------------------------------------------------------------------------
     store current line
```

```
1100 'C3': GOSUB 490
     back one (possibly) full page from EOF and place the resulting line on TOS
     @ Y0=MAX(0,U-R1)
     cursor to bottom line
     @ Y=MAX(0,U-Y0) @ X=0
     update line pointer, fetch current line, redraw screen, display cursor,exit
     @ GOTO 1010
===============================================================================
00/09/14 17:20:16 FILE: P1200      TEXT   2304 09/14/00 17:20
===============================================================================

-------------------------------------------------------------------------------
Command key [DN] - move cursor down one line
ENTRY POINTS: #37 (4600), #38 (2425,2430,2445), #39 (see XREFVE)
ASSERT ON ENTRY: consistent pointers (Y,Y0,Y1,U)
ASSERT ON EXIT: consistent pointers
INPUT:
OUTPUT:
MODIFIED: Y (iff Y#R1 on entry),Y0 (iff Y=R1 on entry),Y1,L$
TRASHED: S,S1$
-------------------------------------------------------------------------------
ENTRY #37: INPUT: none
     store current line
1200 'C4': GOSUB 490
     beep and exit at EOF
1205 IF Y1=U THEN CHIRP @ RETURN
ENTRY #38: ASSERT ON ENTRY: Y1<U
     save y-coordinate of current line
1210 S=Y
     if at bottom line then scroll up one line
     @ IF S=R1 THEN
        push TOS down one line
        @ Y0=Y0+1
        turn cursor off then scroll up
        @ PRINT C5$;R0$;
        clear whatever entered the message line
        @ GOSUB 670
        clear whatever moved from the message line into the bottom line
        @ GOSUB 510
     else simply increment the screen y-coordinate
     ELSE Y=Y+1
     update line pointer, fetch current line
1220 GOSUB 520
     if a scroll took place then display new bottom line
     @ IF S=R1 THEN PRINT L$;
ENTRY #39: ASSERT ON ENTRY: the physical cursor is off
     move cursor at its logical screen coordinates and turn it on
1230 PRINT CUR$(Y,X,C0);C4$;
     and exit
     @ RETURN
===============================================================================
00/09/14 17:20:41 FILE: P1300      TEXT   1792 09/14/00 17:20
===============================================================================

-------------------------------------------------------------------------------
Command key [UP] - move cursor up one line
ENTRY POINTS: #40 (4610)
ASSERT ON ENTRY: consistent pointers (Y,Y0,Y1,U)
ASSERT ON EXIT: consistent pointers
INPUT:
OUTPUT:
MODIFIED: Y (iff Y#0 on entry),Y0 (iif Y=0 on entry),Y1,L$
```

```
TRASHED: S,S1$
--------------------------------------------------------------------------------
ENTRY #40: INPUT: none
     store current line
1300 UP: GOSUB 490
     beep and exit at TOF
1305 IF NOT Y1 THEN CHIRP @ RETURN
     save y-coordinate of current line
1310 S=Y
     if at TOS then scroll down one line
     @ IF NOT S THEN
        push TOS up one line
        @ Y0=Y0-1
        turn cursor off then scroll down
        @ PRINT C5$;R1$;
        clear whatever moved from the bottom line into the message line
        @ GOSUB 670
        clear whatever entered the top line
        @ GOSUB 510
     else simply decrement the screen y-coordinate
     ELSE Y=Y-1
     update line pointer, fetch current line
1320 GOSUB 520
     if a scroll took place then display new top line
     @ IF NOT S THEN PRINT L$;
     display cursor and exit
1330 GOTO 1230
================================================================================
00/09/14 17:21:03 FILE: P1400      TEXT   1024 09/14/00 17:21
================================================================================
--------------------------------------------------------------------------------
Command key [g][ENDLINE] - move forward one page
ENTRY POINTS:
ASSERT ON ENTRY: consisten pointers (Y,Y0,Y1,U)
ASSERT ON EXIT: consistent pointers
INPUT:
OUTPUT:
MODIFIED: X,Y,Y0,Y1,L$
TRASHED: S1$
--------------------------------------------------------------------------------
     store current line
1400 'C6': GOSUB 490
     update TOS pointer
     @ Y0=MIN(Y0+R0,U)
     cursor home, update pointers, fetch CL, redraw and exit
     @ GOTO 1006
================================================================================
00/09/14 17:21:18 FILE: P1500      TEXT   1024 09/14/00 17:21
================================================================================
--------------------------------------------------------------------------------
Command key [f][ENDLINE] - move backward one page
ENTRY POINTS:
ASSERT ON ENTRY: consistent pointers (Y,Y0,Y1,U)
ASSERT ON EXIT: consistent pointers
INPUT:
OUTPUT:
MODIFIED: X,Y,Y0,Y1,L$
TRASHED: S1$
--------------------------------------------------------------------------------
     store current line
```

```
1500 'C7': GOSUB 490
     update TOS pointer
     @ Y0=MIN(Y0-R0,U)
     cursor home , update pointers, fetch CL, redraw and exit
     @ GOTO 1006
================================================================================
00/09/14 17:21:30 FILE: P1600      TEXT    768 09/14/00 17:21
================================================================================

--------------------------------------------------------------------------------
Command key [LF] - move cursor left one column
ENTRY POINTS: #41 (841,842,2005)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X
TRASHED:
--------------------------------------------------------------------------------
ENTRY #41:
     if not at left margin decrement cursor x-coordinate
1600 'C8': IF X THEN X=X-1
        move physical cursor left
        @ PRINT C3$;
     else complain
     ELSE CHIRP
1610 RETURN
================================================================================
00/09/14 17:21:42 FILE: P1700      TEXT    768 09/14/00 17:21
================================================================================

--------------------------------------------------------------------------------
Command key [RT] - move cursor right one column
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X
TRASHED:
--------------------------------------------------------------------------------
     if not at right margin increment cursor x-coordinate
1700 'C9': IF X#C3 THEN X=X+1
        move physical cursor right
        @ PRINT C1$;
     else complain
     ELSE CHIRP
1710 RETURN
================================================================================
00/09/14 17:21:53 FILE: P1800      TEXT   1280 09/14/00 17:21
================================================================================

--------------------------------------------------------------------------------
Command key [f][SPC] - toggle I/R mode
ENTRY POINTS: #42 (844,845), #43 (345)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: R
OUTPUT: R
MODIFIED: R, FLAG 1
TRASHED:
--------------------------------------------------------------------------------
ENTRY #42: INPUT: R
```

```
     R (initially set to 0) is the run-time Replace flag;
     flag 1 (initially false) is the lcd Insert flag.
     NOTE: if the user manually toggles flag 1 (for instance, during input) I/R
     status will be freezed. Manually toggling the flag again will restore
     normal operations. I think the safest way to manage the flag would be:
     R=NOT R @ IF R THEN CFLAG 1 ELSE SFLAG 1 but this solution is shorter and
     not really dangerous (SP).
     toggle those flags...
1800 'C10': R=FLAG(1,R)
ENTRY #43: INPUT: none
     display an appropriate cursor
1805 IF R THEN PRINT C6$; ELSE PRINT C7$;
     and exit
1810 RETURN
================================================================================
00/09/14 17:22:12 FILE: P1900      TEXT    768 09/14/00 17:22
================================================================================
--------------------------------------------------------------------------------
Command key [f][)] - move to next word
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: X
OUTPUT: X
MODIFIED: X
TRASHED: T
--------------------------------------------------------------------------------
     compute new cursor position in T
1900 'C11': GOSUB 540
     set new cursor position within current line
     @ X=T
     display cursor and exit
     @ GOTO 1230
================================================================================
00/09/14 17:22:25 FILE: P2000      TEXT   1536 09/14/00 17:22
================================================================================
--------------------------------------------------------------------------------
Command key [BACK] - back cursor
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X,L$
TRASHED:
--------------------------------------------------------------------------------
     if at left margin then complain
2000 'C12': IF NOT X THEN CHIRP
        and exit
        @ RETURN
     if cursor lies beyond text then move it leftward
2005 IF X>LEN(L$) THEN 1600
     else if replace mode...
2010 IF R THEN
        replace char to the left of the cursor with a ' '
        L$[X,X]=" "
        back cursor logically
        @ X=X-1
        back it physically and display a ' '
        @ print C3$;" ";C3$;
```

```
          and exit
          @ RETURN
       else insert mode...
       delete char to the left of the cursor
2015 L$[X,X]=""
       back cursor logically
       @ X=X-1
       turn cursor off, display right portion of CL + erase dangling character,
       display cursor
       NOTE: CUR$( );C4$; @ RETURN could be replaced by GOTO 1230. This
       saves memory but it is a bit slower.
2020 PRINT C5$;C3$;L$[X+1];" ";CUR$(Y,X,C0);C4$;
       and exit
       @ RETURN
==============================================================================
00/09/14 17:22:43 FILE: P2100      TEXT    512 09/14/00 17:22
==============================================================================
______________________________________________________________________________
Command key [g][LF] - cursor to beginning of line
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X
TRASHED:
______________________________________________________________________________
       cursor at BOL
2100 'C13': X=0
       display cursor and exit
       @ GOTO 1230
==============================================================================
00/09/14 17:22:56 FILE: P2200      TEXT    768 09/14/00 17:22
==============================================================================
______________________________________________________________________________
Command key [g][RT] - cursor to end of line
ENTRY POINTS: #44 (845)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X
TRASHED:
______________________________________________________________________________
ENTRY #44:
       cursor at EOL
2200 'C14': X=MIN(C3,LEN(L$))
       display cursor and exit
       @ GOTO 1230
==============================================================================
00/09/14 17:23:05 FILE: P2300      TEXT    768 09/14/00 17:23
==============================================================================
______________________________________________________________________________
Command key [f][H] - toggle highlight mode
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: H
OUTPUT: H
MODIFIED: H, FLAG 2
```

```
TRASHED:
_______________________________________________________________________________
     H (initially set to 0) is the run-time highlight flag;
     flag 2 (initially false) is the lcd highlight flag;
     NOTE: see note about flag 1 in [f][SPC]
     toggle those flags...
2300 'C15': H=NOT FLAG(2,NOT H)
     and exit
     @ RETURN
===============================================================================
00/09/14 17:23:19 FILE: P2400      TEXT   3072 09/14/00 17:23
===============================================================================
_______________________________________________________________________________
Command key [ENDLINE] - end line
ENTRY POINTS: #45 (845)
ASSERT ON ENTRY: consistent ointers (Y,Y0,Y1,U)
ASSERT ON EXIT: consistent pointers, X=0 & R
INPUT: R
OUTPUT:
MODIFIED: X,Y,Y0,Y1,U,L$,A,A1
TRASHED: T1,R$,S1$
_______________________________________________________________________________
ENTRY #45:
     store current line
2400 'C16': GOSUB 490
     compute indentation in T1
     @ T1=MAX(0,SPAN(L$," ")-1)
     if insert mode...
     @ IF R THEN 2440
        then clear marks
2405    GOSUB 470
        split current line at cursor and indent right (R$) portion
        @ R$=RPT$(" ",T1)&LTRIM$(L$[X+1])
        @ L$=L$[1,X]
        erase right portion from screen and do a line-feed
2410    PRINT C0$[1,C2-X]
        and store left portion
        @ GOSUB 490
        if on last file line...
2415    IF Y1#U THEN 2430
           then if not on last page line display right portion
2420       IF Y#R1 THEN PRINT R$;
           increment line count
2425       U=U+1
           indent cursor
           @ X=T1
           store right portion
           @ RESTORE #1,U
           @ PRINT #1,R$
           move cursor one line down (if false at 2420
           then no scroll will take place, otherwise right
           portion will be scrolled up) and exit
           @ GOTO 1210
        else (not on last file line) open new line
           move cursor down one line
2430       GOSUB 1210
           open a new line above the cursor
           @ GOSUB 2905
           store right portion
           @ L$=R$ @ GOSUB 490
```

```
          display right portion
2435      PRINT L$;
          indent cursor
          @ X=T1
          display cursor and exit
          @ GOTO 1230
     else (replace mode)
        if on last line then
2440    IF Y1=U THEN
          indent cursor
          X=T1
          clear all marks
          @ GOSUB 470
          append an empty line
          @ RESTORE #1,Y1+1
          @ PRINT #1,""
          increment line count
          @ U=U+1
        else cursor at BOL
        ELSE X=0
        move cursor down one line and exit
2445    GOTO 1210
     endif
================================================================================
00/09/14 17:23:53 FILE: P2500      TEXT    768 09/14/00 17:23
================================================================================

--------------------------------------------------------------------------------
Command key [f][RT] - delete character
ENTRY POINTS: #57 (843)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: X
OUTPUT: L$
MODIFIED: L$
TRASHED:
--------------------------------------------------------------------------------
     if no text under the cursor then exit
2500 'C17': IF X>=LEN(L$) THEN RETURN
ENTRY #57: ASSERT ON ENTRY: X<LEN(L$)
           INPUT: X
     delete character from line
2505 L$[X+1,X+1]=""
     turn cursor off, erase character from screen, display cursor and exit
     @ PRINT C5$;L$[X+1];" ";
     @ GOTO 1230
     NOTE: GOTO 1230 could be replaced by CUR$(Y,X,C0);C4$; to gain speed (but
     it's an I/O-bound operation anyway)
================================================================================
00/09/14 17:24:14 FILE: P2600      TEXT   1536 09/14/00 17:24
================================================================================

--------------------------------------------------------------------------------
Command key [f][L] - delete current line
ENTRY POINTS: #46 (4610)
ASSERT ON ENTRY:
ASSERT ON EXIT: Y0 is decremented iff Y0>U & Y=0
INPUT: Y1
OUTPUT:
MODIFIED: Y,Y0,Y1 (iff deleting last line of file),U,L$,A,A1
TRASHED:
--------------------------------------------------------------------------------
```

```
ENTRY #46:
     clear all marks
2600 'C18': GOSUB 470
     delete line from file
     @ DELETE #1,Y1
     turn cursor off
2610 PRINT C5$;
     decrement line count
     @ U=U-1
     if file is empty then
     @ IF U<0 THEN
        put an empty line
        GOSUB 440
        display first page and exit
        @ GOTO 1005
     else (file is not empty)
        if no lines exist below TOS then decrement TOS pointer
2615    Y0=MAX(0,Y0-(Y0>U))
        if EOF was deleted then back cursor one line
        @ Y=MAX(0,Y-(Y1>U))
        update line pointer, fetch current line
        @ GOSUB 520
        clear from current line to BOS, redraw lower portion of screen
        @ GOSUB 525
        display cursor and exit
        @ GOTO 1230
     endif
===============================================================================
00/09/14 17:24:33 FILE: P2700      TEXT   1024 09/14/00 17:24
===============================================================================
_______________________________________________________________________________
Command key [f][DN] - delete to end of line
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT: LEN(L$)<=X | LEN(L$)>X
INPUT:
OUTPUT:
MODIFIED: L$
TRASHED: S1$
_______________________________________________________________________________
     delete exceeding characters
2700 'C19': L$=L$[1,X]
     store modified line
     @ GOSUB 490
     turn cursor off, erase tail of line
     @ PRINT C5$;C0$[1,C2-X];
     display cursor and exit
     @ GOTO 1230
===============================================================================
00/09/14 17:24:50 FILE: P2800      TEXT   1024 09/14/00 17:24
===============================================================================
_______________________________________________________________________________
Command key [f][W] - delete to beginning of next word
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: X
OUTPUT:
MODIFIED: L$
TRASHED: T
```

```
--------------------------------------------------------------------------------
     compute new cursor position in T
2800 'C20': GOSUB 540
     if a wrap-around took place then force cursor at EOL
     @ IF T<=X THEN T=LEN(L$)
     delete word from line (characters in [X+1,T])
2805 L$=L$[1,X]&L$[T+1]
     refresh (shortened) line tail
     @ PRINT C5$;L$[X+1];C0$[1,C2-LEN(L$)];
     display cursor and exit
     @ GOTO 1230
================================================================================
00/09/14 17:25:05 FILE: P2900      TEXT   1280 09/14/00 17:25
================================================================================

--------------------------------------------------------------------------------
Command key [f][O] - open a new line
ENTRY POINTS: #47 (2430)
ASSERT ON ENTRY:
ASSERT ON EXIT: Y,Y0,Y1 not changed
INPUT: Y1
OUTPUT:
MODIFIED: L$,A,A1,X
TRASHED: S1$
--------------------------------------------------------------------------------
     store current line
2900 'C21': GOSUB 490
ENTRY #47:
     clear all marks
2905 GOSUB 470
     turn cursor off
     @ PRINT C5$;
     insert new empty line in file
     @ INSERT #1,Y1;""
     increment line counter
     @ U=U+1
     update line pointer, fetch CL
     @ GOSUB 520
     clear from CL to BOS, redraw lower portion of screen
     @ GOSUB 525
     cursor to BOL, display cursor and exit
     @ X=0 @ GOTO 1230
================================================================================
00/09/14 17:25:21 FILE: P3000      TEXT    768 09/14/00 17:25
================================================================================

--------------------------------------------------------------------------------
Command key [f][UP] - toggle between lower- and upper- case
ENTRY POINTS: #48 (100)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: FLAG -15
OUTPUT: FLAG -15
MODIFIED: FLAG 4, FLAG -15
TRASHED: T
--------------------------------------------------------------------------------
     toggle case
3000 'C22': LC
ENTRY #48: ASSERT ON EXIT: no toggling is performed
     update status of lcd case flag (flag 4)
3005 T=FLAG(4,FLAG(-15))
     and exit
     @ RETURN
===============================================================================
00/09/14 17:25:33 FILE: P3100      TEXT   2048 09/14/00 17:25
===============================================================================
-------------------------------------------------------------------------------
Command key [f][F] - find pattern
ENTRY POINTS: #49 (3300)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X,Y,Y0,Y1,L$,S$,N$ (if USER)
TRASHED: Q$,T,R$,P,L,D,I,J,S1,G (on input error),S1$
-------------------------------------------------------------------------------
     store CL
3100 'C23': GOSUB 490
     set prompt parameter for input routine
     @ Q$="F:"
     do input, parse it and redraw screen
     @ GOSUB 550
ENTRY #49: INPUT: X,Y1,S$
           ASSERT ON ENTRY: screen is already redrawn
           TRASHED: T
     search pattern S$, starting at col. X+2 (immediately right of cursor)
     line Y1 (current line), ending at last file line U in file #1
3105 T=SEARCH(S$,X+2,Y1,U,1)
     if not found...
     @ IF T THEN 3115
        then wrap around EOF and retry at BOF
        search pattern S$, starting at column 1 line 0
        (BOF), ending at line Y1 (CL) in file #1
3110    T=SEARCH(S$,1,0,Y1,1)
        if not found then message out
3112    IF NOT T THEN BEEP @ PRINT FNM$("Pattern not found");
           and fetch CL (maybe we moved in a previous scan)
           @ GOSUB 520
           display cursor and exit
           @ GOTO 1230
        endif
     endif
     {pattern found}
     update CL pointer
3115 Y1=IP(T)
     and cursor x-coordinate
     @ X=IP(FP(T)*1000)-1
     given Y1 update Y,Y0, fetch CL and redraw screen only if a page change is
     necessary (giving some context to the user)
     @ GOSUB 625
     exit
     @ RETURN
===============================================================================
00/09/14 17:26:04 FILE: P3200      TEXT   8960 09/14/00 17:26
===============================================================================
-------------------------------------------------------------------------------
Command key [f][R] - find and replace pattern
ENTRY POINTS: #50 (5100),#51 (3285)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: A,A1
OUTPUT:
```

```
MODIFIED: FILE, (X,Y,Y0,Y1 iff D & L$ matches R$), U (iff D), S$,R$,L$ (iff it
          matches R$),L,D,N$ (if USER),(A,A1 (iff D))
TRASHED: Q$,I,J,FLAG 3,S,S0,S1,S1$,S2$,Z$,T,Q,G (on input error),D
```

```
     set prompt parameter for input routine
3200 'C24': Q$="R:"
     clear Query flag
     @ CFLAG 3
ENTRY #50: INPUT: FLAG 3
     store CL
3205 GOSUB 490
     do input, parse it and redraw screen
     we expect sensible values for:
     S$: search pattern
     R$: replace string
     D : delete-line flag
     L : one-replacement-per-line flag
     @ GOSUB 550
     initialize counter of replacements
     @ S=0
     the following routine parses the replace string (R$) trying to find
     out whether ampersands (&) are special or not. While doing so, it
     builds in S1$ the actual replace string - deprived of special
     ampersands and back-slashes - and it stores the positions of
     dittoes in Q$ (which is treated as a list). At the end of this
     process it will be possible to build the whole replace string by
     inserting substrings matched by the search pattern S$ into S1$ at positions
     recorded by Q$.
     The input string is actually parsed with the following (poorly
     represented) DFA:
                    {\}
             .--------------. ~{\,&}
             !              !.-.
             ! {\}  ~{\,&} !! v
           _ !  _   .----->3225-.
          ! !v ! v  !       ^    !
     ~{\}! 3215 3220        !{}  !{&}
          !_^  ^_!  ! {&}   !    !
                    !_____>3230<'
                {\}
     initialize list of positions of dittoes
     @ Q$=""
     and literal part of replace string
     @ S1$=""
     and character pointer (for advancing in S1$)
     @ I=0
     DFA spans lines 3215-3235
     state &_not_special: advance
3215 GOSUB 3235
     if not '\' then add literal and don't change state
     @ IF Z$#"\" THEN S1$=S1$&Z$ @ GOTO 3215
     else enter state odd_\
     state odd_\: advance
3220 GOSUB 3235
     if '\' then add literal '\' and go to &_not_special
     @ IF Z$="\" THEN S1$=S1$&Z$ @ GOTO 3215
        else if '&' then go to do_special_&
        ELSE IF Z$="&" THEN 3230
           else add literal (and enter state &_special)
           ELSE S1$=S1$&Z$
     state &_special: advance
3225 GOSUB 3235
     if '\' then go to &_not_special
     @ IF Z$="\" THEN 3215
        else if '&' then go to do_special_&
        ELSE IF Z$="&" THEN 3230
           else add literal and don't change state
           ELSE S1$=S1$&Z$ @ GOTO 3225
     state do_special_&:
     append position of ditto to K$
3230 Q$=Q$&CHR$(LEN(S1$))
     and do a transition to &_special on the null string
     @ GOTO 3225
     subroutine advance: advance input character pointer
3235 I=I+1
     if input string not exhausted then return character
     (Z$) and control to caller
     @ IF I<=LEN(R$) THEN Z$=R$[I,I] @ RETURN
     else take control
     ELSE POP
        and save literal replace string
        @ R$=S1$
     determine block boundaries [I,J]
3240 GOSUB 640
     initialize current search coordinates (S0,T)
     @ S0=1 @ T=I
     repeat
        search for pattern S$ in block [T,J]
3245    T=SEARCH(S$,S0,T,J,1)
        if found...
        @ IF NOT T THEN 3285
           determine where matching substring lies [S0,S1]
3250       S0=IP(FP(T)*1000)
           @ S1=S0+RMD(T*1000000,1000)-1
           and pointer to matching line (search-line pointer)
           @ T=IP(T)
           fetch matching line
           @ READ #1,T;L$
           if there isn't enough room for replacement...
3255       IF LEN(L$)-(S1-S0+1)+LEN(R$)+(S1-S0+1)*LEN(K$)<=C2 THEN 3265
              then complain
3260          PRINT FNE$("Replacement too long");
              if doing replace_with_query ([f][Q]) then...
              @ IF FLAG(3) THEN
                 skip to next line
                 S1$=S$ @ GOTO 3280
              else exit
              ELSE 3285
              endif
           else (do replacement)
           we simply need to build the actual replace
           string into S1$, using the literal part (S1$)
           and the list of dittoes (K$) from the DFA
           section, as well as the matched substring
           L$[S0,S1].
           If D then user requested to delete lines from the file. In that case
           the replacement string is substituted with A$ and matching lines
           will be effectively deleted only later on in the loop starting at
           line 3286.
           NOTE: this isn't the best possible algorithm for deleting lines,
          since matching lines are first REPLACEed then DELETEd, but it fits
          easily in the frame of [f][R] and [f][Q] without requiring too much
          code. Moreover, massive line deletion using [f][R]/[Q] is a very
          unlikely event. (SP)
             initialize replace string
3265         S1$=R$
             for each element of list of dittoes
             @ FOR I=LEN(Q$) TO 1 STEP -1
                insert matched substring
3270            S1$=S1$[1,NUM(Q$[I])]&L$[S0,S1]&S1$[NUM(Q$[I])+1]
             endfor
             @ NEXT I
             if deleting lines then temporarily substitute matching line with
             a predefined pattern
             @ IF D THEN S1$=A$
             save CL to verify that some replacement really took place
3275         S2$=L$
             ask user's confirmation (meaningful only with [f][Q])
             @ GOSUB 5105
             if ok then
3276         IF Q THEN
                do replacement !
                L$[S0,S1]=S1$
                @ REPLACE #1,T;L$
                increment counter of replacements
                @ S=S+1
                display modified line (meaningful only with [f][Q])
                @ GOSUB 5135
             endif
          endif (if there isn't room)
          update current search column pointer and line pointer:
          the two assignaments on line 3280 are equivalent to the following
          piece of code:
             IF L (one replacement per line) THEN
                S0=0 column pointer
                T=T+1 line pointer
             ELSE IF LEN(S1$) (not deleting matching strings) THEN
                S0=S0+LEN(S1$) (skip S1$ in L$)
             ELSE IF S2$=L$ (no TRUE replacement took place) THEN
                S0=S0+1 (to avoid infinite loops, since if L$ is not changed
                        then S0 MUST change [don't preserve status quo!]).
          Notice that a simple flag after the above REPLACE statement
          could not replace the function of S2$. In fact, it would cause an
          endless loop if the user specified "replace anything with nothing"
          (R:/\@//).
3280      S0=(S0+LEN(S1$)+(S2$=L$ AND NOT LEN(S1$)))*NOT L
          @ T=T+L
       endif (if found)
    until not found
    @ GOTO 3245
    if deleting matching lines (\^ and \$ in search pattern AND null replace
    string)...
3285 IF NOT D THEN 3290
       then delete all lines matching \^A$\$
       save line counter for later test
3286   D=U
       @ FOR T=J TO I STEP -1
          @ READ #1,T;L$
          @ IF L$=A$ THEN
             DELETE #1,T
```

```
              decrement line counter
              @ U=U-1
              and CL pointer, if CL is below/within block [I,J]
              @ Y1=MAX(0,Y1-(T<=Y1))
            endif
3288      NEXT T
          ! an alternative, more logical (but memory-consuming) algorithm for the
          ! above FOR-NEXT loop:
          ! INIT: T=I
          ! LOOP: T=SEARCH("\^"&A$&"\$",0,T,J,1)
          ! IF T THEN
          !    DELETE #1,T
          !    @ U=U-1 @ J=J-1 @ Y1=MAX(0,Y1-(T<=U))
          !    @ GOTO LOOP
          if at least one deletion then clear all marks
          @ IF D#U THEN GOSUB 470
          given Y1 adjust Y,Y0
3289      GOSUB 630
          if file was depleted then insert one (empty) line
          @ GOSUB 440
      endif
      redraw screen
3289 GOSUB 710
      display # of replacements
      @ PRINT FNM$(STR$(S)&" replacement(s)");
      display cursor and exit
      @ GOTO 1230
================================================================================
00/09/14 17:28:14 FILE: P3300      TEXT    768 09/14/00 17:28
================================================================================
--------------------------------------------------------------------------------
Command key [f][N] - find next occurrence of search pattern
ENTRY POINTS:
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT:
INPUT: S$
OUTPUT:
MODIFIED: (X,Y,Y0,Y1,L$ (if pattern found))
TRASHED: T,S1$
--------------------------------------------------------------------------------
      store current line
3300 'C25': GOSUB 490
      display search pattern
      @ PRINT FNM$("Pattern :"&S$&":");
      do search then exit
      @ GOTO 3105
================================================================================
00/09/14 17:28:26 FILE: P3400      TEXT   1536 09/14/00 17:28
================================================================================
--------------------------------------------------------------------------------
Command key [f][P] - go to absolute line number (position)
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT: line # is rounded to an integer in the range [0,U], X=0 if Y1
                changed
INPUT: user input: a constant or an expression involving Y1 (current line),
       Y0 (top of screen), Y (for moving within the current page), U (last
       file line), A (first mark), A1 (second mark)... any more ideas?
OUTPUT:
MODIFIED: X,Y,Y0,Y1,L$,N$ (if USER)
```

```
TRASHED: I,S1$,T,R$,G (on input error)
------------------------------------------------------------------------------
     store current line
3400 'C26': GOSUB 490
     get user's input (FNI$) with CL as default, evaluate expression (VAL),
     reduce result within [0,U] and round it to an integer (I=)
     @ I=MAX(0,MIN(VAL(FNI$("Line #:",STR$(Y1),0,1)),U))
     set X=0 iff a line change was requested
3435 X=X*(Y1=I)
     set CL pointer to (new) line
     @ Y1=I
     given Y1 update Y,Y0, fetch CL and redraw screen only if a page change
     is necessary (giving some context to the user)
     @ GOSUB 625
     exit
     @ RETURN
==============================================================================
00/09/14 17:28:47 FILE: P3500      TEXT    768 09/14/00 17:28
==============================================================================

------------------------------------------------------------------------------
Command key [f][V] - view amountof free memory left
ENTRY POINTS:
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED:
TRASHED:
------------------------------------------------------------------------------
     prompt user with MEM
3500 'C27': PRINT FNM$("Memory: "&STR$(MEM));
     display cursor and exit
     @ GOTO 1230
==============================================================================
00/09/14 17:28:58 FILE: P3600      TEXT   1280 09/14/00 17:28
==============================================================================

------------------------------------------------------------------------------
Command key [f][S] - (un)set mark(s)
ENTRY POINTS:
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT:
INPUT:
OUTPUT: A,A1
MODIFIED: A (on first press), A1 (on second press), (A,A1 on third press)
TRASHED:
------------------------------------------------------------------------------
     put "Mark" on message line...
3600 'C28': PRINT FNM$("Mark");
     if first mark is unset...
     @ IF A<0 THEN
        put " 1" on message line
        PRINT 1;
        set first mark to CL
        @ A=Y1
        display cursor and exit
        @ GOTO 1230
     else if 2nd mark is unset...
3605 IF A1<0 THEN
        put " 2" on message line
        PRINT 2;
        set 1st mark to min(1st mark,CL) and 2nd mark to max(1st mark,CL)
        @ A1=A @ A=MIN(A,Y1) @ A1=MAX(A1,Y1)
        display cursor and exit
        @ GOTO 1230
     else...
        put "s cleared" on message line
3610    PRINT "s cleared";
        clear 1st and 2nd mark
        @ GOSUB 470
        display cursor and exit
        @ GOTO 1230
     endif
================================================================================
00/09/14 17:29:17 FILE: P3700      TEXT   2304 09/14/00 17:29
================================================================================
________________________________________________________________________________
Command key [f][M] - move block
ENTRY POINTS:
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT: A,A1 unset
INPUT: A,A1
OUTPUT:
MODIFIED: FILE,Y,Y0,Y1 (if Y1<A),A,A1,L$ (if Y1 is within block)
TRASHED: S1$,T,I
________________________________________________________________________________
     store CL, check if both marks are set and error out if not
3700 'C29': GOSUB 740
     check if CL is within block and error out if not
     @ GOSUB 750
     T=1 if block is below CL otherwise T=0 (T is the offset multiplier [see
     below])
     @ T=Y1<A
     put out message
     @ PRINT FNM$("Moving"); @ GOSUB 620
     for each line in the block (I is the offset of the line within the block)
3710 FOR I=0 TO A1-A
        fetch line at an offset which is I for lines below CL (insertion point)
        and constantly 0 for lines above CL (because no lines will be inserted
        between BOF and block in this case)
        @ READ #1,A+I*T;L$
        delete that line (this is why the offset can be constantly 0: if A
        points to a line and that line is deleted and NOT replaced by another
        one then A now points to the line following the one which was deleted)
        @ DELETE #1,A+I*T
        now insert line above CL at an offset which is constantly -1 if the
        line was deleted above CL (since Y1 doesn't really float) and I-1 if
        the line was deleted below CL (since the number of lines between BOF
        and CL effectively increses)
        @ INSERT #1,Y1+I*T-NOT T;L$
     endfor
     @ NEXT I
     if no lines were deleted above CL then increment CL pointer by size of
     block
3735 IF T THEN Y1=Y1+(A1-A+1)
     given Y1, adjust Y,Y0
3745 GOSUB 630
     clear all marks
     @ GOSUB 470
     redraw screen
     @ GOSUB 710
     display cursor and exit
     @ GOTO 1230
==============================================================================
00/09/14 17:29:53 FILE: P3800      TEXT   1792 09/14/00 17:29
==============================================================================

------------------------------------------------------------------------------
Command key [f][C] - copy block
ENTRY POINTS:
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT: A and A1 still reference the same lines as before
INPUT: A,A1
OUTPUT:
MODIFIED: FILE,U,A,A1,Y,Y0,Y1
TRASHED: S1$,T,I,S
------------------------------------------------------------------------------
     store CL, check if both marks are set and error out if not
3800 'C30': GOSUB 740
     check if CL is within block and error out if not
     @ GOSUB 750
     put out message
     @ PRINT FNM$("Copying"); @ GOSUB 620
     NOTE: see [f][M] for its similarity
     T=2 if block is below CL otherwise T=1 (T is the offset multiplier [see
     below])
3810 T=(Y1<A)+1
     for each line of the block (I is the offset within the block)
     @ FOR I=0 TO A1-A
        fetch line at offset I if inserting below block or at offset 2*I if
        inserting above block
        @ READ #1,A+I*T;L$
        insert line right above CL
        @ INSERT #1,Y1+I;L$
     end for
     @ NEXT I
     T=1 if block is below CL otherwise T=0
3835 T=T-1
     compute size of block in S
     @ S=A1-A+1
     increment line counter by S
     @ U=U+S
     if lines were inserted above block then increment marks by S (to preserve
     their references)
     @ A=A+S*T @ A1=A1+S*T
     update CL pointer
     @ Y1=Y1+S
     given Y1, adjust Y,Y0
3840 GOSUB 630
     redraw screen
     @ GOSUB 710
     display cursor and exit
     @ GOTO 1230
==============================================================================
00/09/14 17:30:20 FILE: P3900      TEXT   3072 09/14/00 17:30
==============================================================================

------------------------------------------------------------------------------
Command key [f][D] - delete block
ENTRY POINTS:
ASSERT ON ENTRY: screen is already redrawn
ASSERT ON EXIT:
INPUT: A,A1
```

OUTPUT:
MODIFIED: N$ (if USER), X (if A<=Y1<=A1),U,A,A1,L$ (if CL is deleted)
TRASHED: S1$,P,I

```
     store CL, check if both marks are set and error out if not
3900 'C31': GOSUB 740
     put out message and cursor
     @ PRINT FNM$("Delete? Y/N/Q"); @ GOSUB 1230
     ask for user's confirmation in P
3910 GOSUB 615
     turn off cursor and clear message line
     @ GOSUB 670
     if user didn't press "Y" then display cursor and exit
     @ IF P>1 THEN 1230
     put out message
3915 PRINT "Deleting"; @ GOSUB 620
     for each line in the block
     @ FOR I=A TO A1
        delete that line
        @ DELETE #1,A
     endfor
     @ NEXT I
     if CL was deleted then set cursor x-coordinate to zero
3925 X=X*(Y1<A OR Y1>A1)
     the following assignament can be understood as follows:
        if block is below TOS                   IF Y0<A
           don't move TOS to let lines enter
           screen from below                       Y0=Y0
        else if block includes TOS              ELSE IF A<=Y0<=A1
           compute size of portion of block
           containing BOS                          T=A1-Y0+1
           set TOS T lines above 1st mark to
           let T lines enter screen from above     Y0=A-T
           if not enough lines are available
           above deleted block then set
           TOS to BOF                              Y0=MAX(0,Y0)
        else if block is completely above TOS   ELSE IF A1<Y0
           decrement Y0 by size of block to let
           lines enter screen from above           Y0=Y0-(A1-A+1)
3930 Y0=Y0*(Y0<A)+MAX(0,A-(A1-Y0+1))*(A<=Y0 AND Y0<=A1)+(Y0-(A1-A+1))*(A1<Y0)
     the following assignament can be understood as follows:
        if block is below CL                    IF Y1<A
           don't move CL                           Y1=Y1
        else if block includes CL               ELSE IF A<=Y1<=A1
           if block includes last file line        IF A1=U
              set CL to last line                     Y1=A-1
           else                                    ELSE
              set CL at first line below
              deleted block                           Y1=A
        else if block is completely above CL    ELSE IF A1<Y1
           decrement Y1 by size of block           Y1=Y1-(A1-A+1)
3935 Y1=Y1*(Y1<A)+MAX(0,A-(A1=U))*(A<=Y1 AND Y1<=A1)+(Y1-(A1-A+1))*(A1<Y1)
     given Y1, adjust Y,Y0
3940 GOSUB 630
     decrement line counter by size of block
     @ U=U-(A1-A+1)
     if file was depleted then insert one empty line
     @ GOSUB 440
     clear all marks
3960 GOSUB 470
```

```
     redraw screen
     @ GOSUB 710
     display cursor and exit
     @ GOTO 1230
================================================================================
00/09/14 17:31:15 FILE: P4000      TEXT   1024 09/14/00 17:31
================================================================================
--------------------------------------------------------------------------------
Command key [g][ON] - edit another file
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT: buffer file isn't purged
INPUT:
OUTPUT:
MODIFIED: this is a complete restart; most variables are modified
TRASHED:
--------------------------------------------------------------------------------
     store CL, display replace cursor + clear screen, restore I/O devices,
     scroll & delay rates, pwidth, endline, clear flags
4000 'C32': GOSUB 910
     clear return stack
     @ POP
     reinitialize variables and go to input edit file name
     @ GOTO 100
================================================================================
00/09/14 17:31:22 FILE: P4100      TEXT   1024 09/14/00 17:31
================================================================================
--------------------------------------------------------------------------------
Command key [f][(] - move to previous word
ENTRY POINTS: #52 (842)
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X
TRASHED:
--------------------------------------------------------------------------------
ENTRY #52:
     reverse CL (REV$) to search leftwards, starting from the column
     immediately left of the cursor (LEN(L$)-X+1) find position of first
     non-blank character (SPAN) or default to 1 if no such character exists
     (MAX(1,...), starting from that position search first ' ' (POS), convert
     this value into x-coordinates (LEN(L$)-...+1) ad assign it to X
4100 'C33': X=LEN(L$)-POS(REV$(L$)," ",MAX(1,SPAN(REV$(L$)," ",LEN(L$)-X+1)))+1
     if X>LEN(CL) then set X to 0 (to catch 1st word on line)
4105 X=X*(X<=LEN(L$))
     display cursor and exit
     @ GOTO 1230
================================================================================
00/09/14 17:31:39 FILE: P4200      TEXT   1792 09/14/00 17:31
================================================================================
--------------------------------------------------------------------------------
Command key [f][E] - erase invisible characters and trailing blanks
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: FILE,X
TRASHED: S1$,T,I,J,S,P
```

```
     turn cursor off
4200 'C34': PRINT C5$;
     store current line
     @ GOSUB 490
     initialize counter of bytes saved
     @ T=0
     get block boundaries [I,J]
     @ GOSUB 640
     put out message
     @ PRINT FNM$("Erasing...");
     set cursor to 1st column (to a known position, since if invisible
     characters were deleted from CL and the cursor weren't moved then the user
     may not understand what's going on)
     @ X=0
     for each line in the block
4205 FOR I=I TO J
        fetch that line
        @ READ #1,I;L$
        save line length for later
        @ S=LEN(L$)
        set pointer to beginning of line
        @ P=1
        while an invisible character is a member of the line
4210    P=MEMBER(L$,M$,P)
           delete that character and increment counter of bytes saved
           @ IF P THEN L$[P,P]="" @ T=T+1
        endwhile
        @ GOTO 4210
        now trim trailing spaces
4215    L$=RTRIM$(L$)
        if any characters were removed from the line then store line and
        increment counter by the number of trimmed spaces
        @ IF S#LEN(L$) THEN REPLACE #1,I;L$ @ T=T+S-LEN(L$)
     endfor
4220 NEXT I
     redraw screen
     @ GOSUB 710
     put out count of bytes saved
     @ PRINT FNE$(STR$(T)&" byte(s) saved");
     display cursor and exit
     @ GOTO 1230
==============================================================================
00/09/14 17:32:07 FILE: P4300      TEXT   1024 09/14/00 17:32
==============================================================================

------------------------------------------------------------------------------
Command key [f][B] - cursor to previous occurrence of character on line
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X (not always),N$ (if USER)
TRASHED: Q$,T
------------------------------------------------------------------------------
get text key or abort if it is a command key
4300 'C35': GOSUB 610
     find position of key taking case into account (MAPKEY)
     NOTE: this means that specifying case in key definitions is impossible
     @ T=POS(REV$(L$),MAPKEY$(Q$),LEN(L$)-X+1)
     if key found then update cursor position
     @ IF T THEN X=LEN(L$)-T
     display cursor and exit
4310 GOTO 1230
================================================================================
00/09/14 17:32:24 FILE: P4400      TEXT   1024 09/14/00 17:32
================================================================================
--------------------------------------------------------------------------------
Command key [f][G] - cursor to next occurrence of character on line
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: X (not always),N$ (if USER)
TRASHED: Q$,T
--------------------------------------------------------------------------------
4400 'C36': GOSUB 610
     find position of key taking case into account (MAPKEY)
     NOTE: this means that specifying case in key definitions is impossible
     @ T=POS(L$,MAPKEY$(Q$),X+2)
     if key found then update cursor position
     @ IF T THEN X=T-1
     display cursor and exit
4410 GOTO 1230
================================================================================
00/09/14 17:32:37 FILE: P4500      TEXT    768 09/14/00 17:32
================================================================================
--------------------------------------------------------------------------------
Command key [f][A] - toggle wrap-around (automatic) mode
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: W
OUTPUT: W
MODIFIED: W, FLAG 0
TRASHED:
--------------------------------------------------------------------------------
     W (initially set to 0) is the run-time wrap-around flag (W=0 means w-a
     enabled);
     flag 0 (initialy false) is the lcd w-a flag;
     NOTE: see note about flag 1 in [f][SPC]
     toggle those flags...
4500 'C37': W=NOT FLAG(0,NOT W)
     and exit
     @ RETURN
================================================================================
00/09/14 17:32:49 FILE: P4600      TEXT   1536 09/14/00 17:32
================================================================================
--------------------------------------------------------------------------------
Command key [f][J] - join two lines
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT: Y1 unchanged if lines can be joined, otherwise Y1 incremented by
                1
INPUT:
OUTPUT:
MODIFIED: (Y,Y0,Y1,L$ (if Y1#U))
TRASHED: (S1$,R$,S (if Y1#U))
--------------------------------------------------------------------------------
```

```
     if CL isn't the last file line...
4600 'C38': IF Y1=U THEN RETURN
     then
     ELSE
        remove trailing blanks from CL and save it in R$ (NOTE: CL is left
        UNCHANGED in FILE)
        R$=RTRIM$(L$)
        move cursor down one line (this also modifies L$)
        @ GOSUB 1200
        if concatenate(norightblanks(lineabove), blank, noleftblanks(CL))
        doesn't fit within the window
4605    IF LEN(R$)+LEN(LTRIM$(L$))>=C2 THEN
           then put out message
           PRINT FNE$("Line too long");
           display cursor and exit (NOTE: cursor stays on offending line to
           allow incoming [f][J]'s to continue joining)
           @ GOTO 1230
        else
           store joined lines in L$
4610       L$=R$&" "&LTRIM$(L$)
           store L$, move cursor up one line
           @ GOSUB 1300
           delete CL (which is a duplicate of the left part of the line
           below), redraw lower portion of screen and exit
           @ GOTO 2600
        endif
     endif
================================================================================
00/09/14 17:33:12 FILE: P4700      TEXT   1536 09/14/00 17:33
================================================================================

--------------------------------------------------------------------------------
Command key [f][T] - go to next tab-stop on line
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: S2,T$
OUTPUT:
MODIFIED: X
TRASHED: I,T
--------------------------------------------------------------------------------
     if relative tab-stops then put cursor to next tab-stop...
4700 'C39': IF S2 THEN X=RMD((X+S2) DIV S2*S2,C1)
        with wrap-around
        @ X=X*(X>=S2)
        display cursor and exit
        @ GOTO 1230
     else
        if cursor position is grreater than maximum tab stop then wrap-around
4705    IF X>=NUM(T$[LEN(T$)])-NUM(T$) THEN X=0
           display cursor and exit
           @ GOTO 1230
        else
           i=0; repeat
              increment i
4710          FOR I=1 TO LEN(T$)
              compute i-th tab-stop in T
                 @ T=NUM(T$[I])-NUM(T$)
           until tab-stop T is greater than cursor position
                 @ IF T>X THEN I=INF
4715       NEXT I
```

```
          reduce new cursor position within window (to prevent user's mistakes)
          @ X=RMD(T,C1)
          display cursor and exit
          @ GOTO 1230
       endif
    endif
================================================================================
00/09/14 17:33:35 FILE: P4800      TEXT   2816 09/14/00 17:33
================================================================================

--------------------------------------------------------------------------------
Command key [f][Y] - yank block/file to buffer or file to device
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: A,A1
OUTPUT:
MODIFIED: N$ (if USER), Y$,FILE Y$
TRASHED: S1$,T,R$,G,#2,I,J
--------------------------------------------------------------------------------
     store current line
4800 'C40': GOSUB 490
     input buffer/device name
4805 Y$=FNI$("Yank to:",Y$,16,1)
     if null input then display cursor and exit
     @ IF NOT LEN(Y$) THEN 1230
     else
        put out message
4810    GOSUB 600
        store buffer size (or error #) in G
        @ G=FILESZR(Y$)
        if buffer already exists then purge it
        @ IF G>=0 THEN PURGE Y$
4815    ON ERROR GOTO 4825
        if no marks are set
4820       IF A<0 THEN
           then ATTEMPT to copy whole file to buffer/device
              COPY F$ TO Y$
           and if everything went well then do some house-keeping and exit
              @ OFF ERROR
              @ GOTO 4860
        endif
4825    OFF ERROR
        if any errors occurred (but 'file not found')
        @ IF G<0 AND G#-57 THEN
           put out error message
           GOSUB 420
           and retry
           @ GOTO 4805
        else (no errors & mark(s) set & not a device)
           create buffer
4845       CREATE TEXT Y$
           @ ASSIGN #2 TO Y$
           get block boundaries [I,J]
           @ GOSUB 640
           foreach line in the block
4850       FOR I=I TO J
              fetch that line
              @ READ #1,I;L$
              and append it to the buffer
              @ PRINT #2;L$
           endfor
           @ NEXT I
           @ ASSIGN #2 TO *
           fetch CL (for-loop changed L$)
           NOTE: PROBABLY using S1$ would allow deleting this instruction
           @ GOSUB 520
           clear message line
4860       GOSUB 670
           display cursor and exit
           @ GOTO 1230
        endif
     endif
=================================================================================
00/09/14 17:34:10 FILE: P4900      TEXT   2048 09/14/00 17:34
=================================================================================
---------------------------------------------------------------------------------
Command key [f][I] - insert from buffer
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: A,A1
OUTPUT:
MODIFIED: N$ (if USER),Y$,FILE,U,Y,Y0,Y1
TRASHED: S1$,T,R$,G,#2,I
---------------------------------------------------------------------------------
     store current line
4900 'C41': GOSUB 490
     input buffer name
4905 Y$=FNI$("Insert from:",Y$,16,1)
     if null input then display cursor and exit
     @ IF NOT LEN(Y$) THEN 1230
     else
        store buffer size (or error#) in G
4920    G=FILESZR(Y$)
        if buffer empty then display cursor and exit
        @ IF G=0 THEN 1230
        else
           put out message
4930       GOSUB 600
           if something's wrong with the buffer then put out error message
           @ IF G<0 THEN GOSUB 420
              and retry
              @ GOTO 4905
           else
4935          ASSIGN #2 TO Y$
              foreach line in the buffer
              @ FOR I=0 TO G-1
                 fetch that line
                 @ READ #2;L$
                 insert it right above CL
                 @ INSERT #1,Y1+I;L$
              endfor
              @ NEXT I
4950          ASSIGN #2 TO *
              increment line counter
              @ U=U+G
              and CL pointer
              @ Y1=Y1+G
              given Y1, adjust Y,Y0 giving some context to the user
              @ GOSUB 630
```

```
            redraw screen
            @ GOSUB 710
            display cursor and exit
            @ GOTO 1230
         endif
      endif
   endif
================================================================================
00/09/14 17:34:40 FILE: P5000      TEXT   1280 09/14/00 17:34
================================================================================
--------------------------------------------------------------------------------
Command key [g][RUN] - enter control characters
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED: J$ (not always)
TRASHED: Q$,P
--------------------------------------------------------------------------------
     get key or abort if it is a command key
5000 'C42': GOSUB 610
     see if it can be a control character and which
     NOTE: unfortunately the standard keystrokes for control characters involve
     two keys which are not found in the HP71 keyboard (backslash, underscore).
     This means that key definitions are necessary to exploit this routine.
     @ P=POS("@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_",Q$)
     if not a control character
5010 IF NOT P THEN
        undo that key
        FKEY Q$
        and exit
        @ RETURN
     else
     ELSE
        compute byte corresponding to control character taking into account
        highlight mode
        J$=CHR$(128*H+P-1)
        pass control to text key handler
        @ GOTO 800
     endif
================================================================================
00/09/14 17:35:04 FILE: P5100      TEXT   2816 09/14/00 17:35
================================================================================
--------------------------------------------------------------------------------
Command key [f][Q] - find and replace pattern with query (conditionally)
This is a collection of subroutines to be used in conjunction with [f][R].
ENTRY POINTS: #53 (3275), #54 (3276)
[f][Q] has the same effects as [f][R].
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT:
OUTPUT:
MODIFIED:
TRASHED:
--------------------------------------------------------------------------------
     Main routine:
     set prompt parameter for input routine
5100 'C43': Q$="Q:"
     set Query flag
     @ SFLAG 3
     go do [f][R] (entry #50)
     @ GOSUB 3205
     clear Query flag
     @ CFLAG 3
     exit
     @ RETURN
ENTRY #53: ask for user's confirmation
           INPUT: S0,T from [f][R]
           OUTPUT: Q=1 if confirming or [f][R] (flag 3 clear), loss of control
           if quitting, otherwise Q=0
     initialize response to true
5105 Q=1
     if doing [f][Q] (flag 3 true)...
     @ IF NOT FLAG(3) THEN RETURN
        set cursor to column where match is found
5110    X=S0-1
        and CL pointer to line of match
        @ Y1=T
        if CL is on screen already
        @ IF FNY THEN
           simply update cursor y-coordinate
           Y=Y1-Y0
        else
        ELSE
           update both TOS pointer and y-coordinate giving some context to the
           user
           GOSUB 635
           and redraw screen
           @ GOSUB 710
        endif
        put out message
5120    PRINT FNM$("Y/N/Q ? :"&L$[S0,S1]&": to :"&S1$&":");
        and cursor
        @ GOSUB 1230
        wait for a key-press
5130    GOSUB 615
        clear message line
        @ GOSUB 670
        if 'quit' was specified
        @ IF P=3 THEN
           pass control to exit routine of [f][R] (sorry, I know it's
           disgusting, but...)
           POP
           @ GOTO 3285
        else
        ELSE
           if 'yes' then Q=1 else Q=0
           Q=P=1
           return (to [f][R])
           @ RETURN
        endif
     endif
ENTRY #54: display modified line
           ASSERT ON ENTRY: screen is already redrawn
           INPUT: FLAG 3,L$
     if doing [f][Q]
5135 IF FLAG(3) THEN
        turn cursor off
        PRINT C5$;
```

```
          erase CL from screen
          @ GOSUB 510
          display modified line
          @ PRINT L$;
       endif
       return (to [f][Q])
5140 RETURN
================================================================================
00/09/14 17:35:48 FILE: P5200      TEXT    768 09/14/00 17:35
================================================================================

--------------------------------------------------------------------------------
Command key [f][0] - toggle user mode
ENTRY POINTS:
ASSERT ON ENTRY:
ASSERT ON EXIT:
INPUT: FLAG -9
OUTPUT: FLAG -9
MODIFIED: FLAG -9
TRASHED: T
--------------------------------------------------------------------------------
     NOTE: this is not a 1 USER command.
     toggle that flag and exit
5200 'C44': T=FLAG(-9,NOT FLAG(-9)) @ RETURN
================================================================================
00/09/14 17:35:59 FILE: P9000      TEXT    768 09/14/00 17:35
================================================================================

--------------------------------------------------------------------------------
Routine: version subroutines
ENTRY POINTS: #55 (40,45,50,51), #56 (905)
ASSERT ON ENTRY:
ASSERT ON EXIT: all lines but 9999 are exit points of VE.
INPUT:
OUTPUT:
MODIFIED:
TRASHED:
--------------------------------------------------------------------------------
ENTRY #55: INPUT: V$
     complain if lex files which can respond to the version poll aren't in RAM
9000 'VERERR': DISP "No ";V$;" LEX file." @ BEEP
     do house-keeping and end
     @ GOTO 905
ENTRY #56: Abandon every hope Ye who exit...
9998 END SUB
     return version of VE
9999 SUB VER(V$) @ V$="VE:1.1" @ END SUB
```

```
================================================================================
00/09/14 17:36:16 FILE: VEFOLD     BASIC   312 09/14/00 17:36
================================================================================
10 SUB VEFOLD(#1,U,C2)
20 DIM L$[150],C$[9] @ C$=" -"
30 PRINT "Wait..." @ C1=C2+1 @ I=0
40 'LP': IF I>U THEN END
50 READ #1,I;L$ @ IF LEN(L$)<=C2 THEN 'NL'
60 'NP': IF NOT MEMBER(L$[1,C2],C$) THEN BEEP @ DISP "WRN L";STR$(I);":";MSG$(65
) @ GOTO 'SL'
70 T=C1-MEMBER(REV$(L$[1,C2]),C$) @ INSERT #1,I;L$[1,T] @ L$=L$[T+1] @ I=I+1 @ U
=U+1
80 IF LEN(L$)>C2 THEN 'NP'
90 'SL': REPLACE #1,I;L$
100 'NL': I=I+1 @ GOTO 'LP'
================================================================================
00/09/14 17:36:26 FILE: VEFOLDT    TEXT   3072 09/14/00 17:36
================================================================================

--------------------------------------------------------------------------------
Subprogram VEFOLD - version 1.1 - optional run-time subprogram for VE
This subprogram MUST reside in file VEFOLD
Purpose: to fold lines that exceed C2 characters in length.
ASSERT ON ENTRY: channel #1 assigned to a text file, U is not greater than the
                 last record # (usually it IS the last record #)
ASSERT ON EXIT: no lines in [0,U] are longer than C2 characters
INPUT: #1, U (see above), C2 (see above), C$ (a list of characters at which
lines can be folded)
OUTPUT: U
MODIFIED: U, file #1
--------------------------------------------------------------------------------
10 SUB VEFOLD(#1,U,C2)
   text line + list of characters
20 DIM L$[150],C$[9]
   the following assignment is only a suggestion: fold at spaces and dashes
   NOTE: long lines are folded at the rightmost occurrence of a member of C$
   (in [1,C2]). If no such occurrence exists then a warning is issued. Members
   of C$ are left at the END of the folded portion of the line.
   ex. if C2=7, C$=" " and #1 contains:
   this is the time
   for all good men
   then #1 will become
   this_
   is the_
   time
   for_
   all_
   good_
   men
   where underscores represent dangling spaces
   @ C$=" -"
   assume output device is assigned as a PRINTER (this is perfectly compatible
   with VE)
30 PRINT "Wait..."
   C1 will come handy
   @ C1=C2+1
   init pointer to current line
   @ I=0
   line_process:
   while I doesn't point past the last line
40 'LP': IF I>U THEN END
```

```
      fetch I-th line
50    READ #1,I;L$
      if it needs folding then
      @ IF LEN(L$)<=C2 THEN 'NL'
         next_portion:
         repeat
            if there isn't a character at which to fold
60          'NP': IF NOT MEMBER(L$[1,C2],C$) THEN
               then complain
               BEEP
               issue warning
               @ DISP "WRN L";STR$(I);":";MSG$(65)
               save current portion of line and continue
               @ GOTO 'SL'
            else
               put in T the position of the rightmost member of C$ (in [1,C2])
70             T=C1-MEMBER(REV$(L$[1,C2]),C$)
               insert left portion of string into the file
               @ INSERT #1,I;L$[1,T]
               trim left portion
               @ L$=L$[T+1]
               point right below inserted portion
               @ I=I+1
               increment block boundary
               @ U=U+1
            endif
         until line doesn't need folding
80       IF LEN(L$)>C2 THEN 'NP'
         store_portion:
         store portion left after folding
90       'SL': REPLACE #1,I;L$
       endif
       next_line:
       point to next line
100    'NL': I=I+1
    endwhile
    @ GOTO 'LP'
```

```
==========================================================================
00/09/14 17:37:17 FILE: IMPROVE    TEXT   5120 09/14/00 17:37
==========================================================================
```
This file describes some improvements which were planned but, for many reasons, were never implemented.

REPEATING KEYS
Version 1.1 has four repeating keys: [UP],[DN],[LF] and [RT]. File MAKE9095 lists other five keys as possible candidates for repetition ([f][(], [f][)], [f][LF], [f][RT] and [f][T]). Text keys are not repeating. This situation is primarily due to KEYDOWN, which doesn't accept other than primary keys. Line 285, therefore, can be executed only if K$ is a primary key, otherwise the program halts on error. If K$ could be replaced by an expression returning the value of the primary key associated with K$ then much of command key repetition could be saved, at least for those five command keys and for all text keys. Of course, testing the primary counterpart of any key for being down isn't logically correct for prefixed keys, but that's all we have (even the lcd editor has the same problem). The string expression should be as simple as possible, because speed is a primary concern for VE, and should map key numbers [57,112] and [113,168] onto [1,56]. A piece of code like
```
   t=key#(key) mod 56
   if t=0 then t=56
   if keydown(keyname(t)) then repeat key
```
would do, but it is surely too slow. Therefore, I suggest something more involved and less correct, but faster, such as
```
   if keydown(keyname(key#(key) mod 56)) then repeat key
```
This solution works for all keys but key # 56,[+], which isn't repeating.
A similar scheme could be applied to text keys, as well, thus removing a serious limitation of VE. To sum up, repeating keys can be implemented as follows:
1) DIMension and extend (using MAKE9095) H$, the list of repeating command keys
2) Rewrite line 285 as follows:
```
   285 IF KEYDOWN(KEYNAM$(MOD(KEYNUM(K$)),56) THEN 275 ELSE RETURN
```
3) Add line 267 as follows:
```
   267 IF KEYDOWN(KEYNAM$(MOD(KEYNUM(K$),56)) THEN GOSUB 800 @ GOTO 265
```
where KEYNAM$ and KEYNUM are keywords found in the CUSTUTIL lex file.
Of course, I wish KEYDOWN were rewritten to be less xenophobic: for instance, it could accept 'fX' or 'gX' and understand that 'X' was really meant (it couldn't be anything else anyway). Then step (2) could be left out and step (3) could be simplified to
```
   267 IF DOWNKEY(K$) THEN GOSUB 800 @ GOTO 267
```
*****
Two after-thoughts:
(1) command key [f][J] could be made repeating; the user could 'pack' the whole file just by keeping [J] depressed.
(2) Variable H$ can be eliminated from VE and MAKE9095 in a simple way:
 (a) rearrange O$ (in MAKE9095) so that all repeating keys are grouped together at the beginning of O$ (ex. O$[1,4])
 (b) rename the labels for command keys in VE to reflect the new order in O$ (you don't need to move code)
 (c) substitute line 280 with
```
      280 IF K>4 THEN RETURN
```
 where 4 represents the number of repeating keys. Since this modification is a bit cumbersome I suggest to apply it only ater all repeating command keys have been chosen and implemented.

MISSING COMMANDS
Certainly many commands were left out from VE, but that's primarily due to memory constraints or dubious utility. However, there's at least one command which I wish had been implemented, namely the 'execute BASIC and insert output'

command. I feel that such a command would tremendously enhance VE as a working tool. The command could be assigned to [f][X] (execute) and could be implemented as follows:

(1) DIMension O$

(2) Add CHR$(NUM(MAPKEY$([f][X]))) to the list of command keys in MAKE9095; run MAKE9095; EDIT VE; MERGE OSTR

(3) add the following subroutine

```
     store CL
5300 'C45': GOSUB 490
     put out message
     @ GOSUB 600
     ignore missing sub
     @ ON ERROR GOTO 5310
        call user's program
        @ CALL USERPRGM(#1,Y1,C2)
5310 OFF ERROR
     compute file increase in T
     @ T=FILESZR(F$)-1-U
     update CL pointer
     @ Y1=Y1+T
     update pointer to last line
     @ U=U+T
     given Y1, update TOS pointer and y-coordinate
     @ GOSUB 630
     redraw screen and exit
     @ GOTO 710
```

Obviously, the above code puts all responsibility upon the user's program USERPRGM which:

(a) MUST NOT mess up with flags 0-4, -3, DISPLAY IS & PRINTER IS device assignments

(b) MUST send all relevant output to the edit file (#1) using ONLY the following statement

```
     INSERT #1,Y1,output_string[1,C2]
```

(c) doesn't have any other constraints apart from (a) and (b).

Possible applications:

(A) the easiest one is to recall the results of a computation into the edit file. The user could write something like this:

```
10 SUB USERPRGM(#1,Y1,C2)
20 DIM S$[255]
30 LINPUT "Expression:";S$
40 IF S$="" THEN END SUB
50 S$=STR$(VAL(S$))
60 INSERT #1,Y1,S$[1,C2]
70 GOTO 30
```

(B) I am sure you can think of many other applications...

____

```
===============================================================================
00/09/14 17:38:32 FILE: MKVEDB     BASIC  1219 09/14/00 17:38
===============================================================================
1 ! MKVEDB v.1.1 - MaKE VE Data Base
10 F$="VEDB"
20 DIM C1$[4],C2$[4],C3$[4],C4$[4],C5$[4],C$[4],R0$[4],R1$[4],C6$[4],C7$[4],E1$[
4],E$[4]
30 DIM S$[96],M$[256],O$[10]
40 INTEGER S,E,T
50 T=FLAG(-1,1) @ PURGE F$ @ T=FLAG(-1,T)
70 CREATE DATA F$ @ ASSIGN #1 TO F$
80 LINPUT "Invisible char's (ASCII ordered subranges):","0,31 @ 127,159 @ 255,25
5";S$
90 GOSUB 310
100 LINPUT "Cursor right (ASCII):","27,67";S$ @ C1$=FNE$(S$)
110 LINPUT "Cursor down (ASCII):","27,66";S$ @ C2$=FNE$(S$)
120 LINPUT "Cursor left (ASCII):","27,68";S$ @ C3$=FNE$(S$)
130 LINPUT "Cursor ON (ASCII):","27,62";S$ @ C4$=FNE$(S$)
140 LINPUT "Cursor OFF (ASCII):","27,60";S$ @ C5$=FNE$(S$)
150 LINPUT "Clear to bottom (ASCII):","27,74";S$ @ C$=FNE$(S$)
160 LINPUT "Scroll DOWN (ASCII):","27,83";S$ @ R0$=FNE$(S$)
170 LINPUT "Scroll UP (ASCII):","27,84";S$ @ R1$=FNE$(S$)
180 LINPUT "Display replace cursor (ASCII):","27,82";S$ @ C6$=FNE$(S$)
190 LINPUT "Display insert cursor (ASCII):","27,81";S$ @ C7$=FNE$(S$)
200 LINPUT "Clear display device (ASCII):","27,69";S$ @ E1$=FNE$(S$)
220 LINPUT "Clear screen page (ASCII):","27,72,27,74";S$ @ E$=FNE$(S$)
230 PRINT #1;M$,C1$,C2$,C3$,C4$,C5$,C$,R0$,R1$,C6$,C7$,E1$,E$
240 ASSIGN #1 TO *
250 END
260 DEF FNE$(I$)
270 O$="" @ S$=I$&","
280 IF LEN(S$)>1 THEN O$=O$&CHR$(VAL(S$)) @ S$=S$[POS(S$,",")+1] @ GOTO 280
290 FNE$=O$
300 END DEF
310 M$="" @ S$=S$&"@"
320 IF LEN(S$)<2 THEN RETURN
330 S=VAL(S$) @ E=VAL(S$[POS(S$,",")+1]) @ S$=S$[POS(S$,"@")+1]
340 FOR I=S TO E @ M$=M$&CHR$(I) @ NEXT I @ GOTO 320
===============================================================================
00/09/14 17:39:00 FILE: MKVEDBT    TEXT   2560 09/14/00 17:38
===============================================================================
_______________________________________________________________________________
Program MKVEDB - version 1.1 - set up data base for VE
OUTPUT: DATA file VEDB which must reside in memory whenever VE is executed
_______________________________________________________________________________
1 ! MKVEDB v.1.1 - MaKE VE Data Base
10 F$="VEDB"
   same variables as in VE
   NOTE: notice variable DIMensioning: it can be a potential problem in a ROM
   version of VE when using an interface with LONG escape sequences (>4). Such
   is not the case for HP82163-compatible interfaces, but a bit of foresight of
   future extensions won't do any damage. Maybe even 4 isn't enough and
   variables should be overdimensioned to greater extent.
20 DIM C1$[4],C2$[4],C3$[4],C4$[4],C5$[4],C$[4],
   R0$[4],R1$[4],C6$[4],C7$[4],E1$[4],E$[4]
   scratch variables
30 DIM S$[96],M$[256],O$[10]
40 INTEGER S,E,T
50 T=FLAG(-1,1) @ PURGE F$ @ T=FLAG(-1,T)
70 CREATE DATA F$ @ ASSIGN #1 TO F$
```

```
   default value is for HP82163. More advanced interfaces (i.e. PACSCREEN) can
   enter the null string (meaning that all characters are displayable)
80 LINPUT "Invisible char's (ASCII ordered subranges):",
   "0,31 @ 127,159 @ 255,255";S$
90 GOSUB 310
100 LINPUT "Cursor right (ASCII):","27,67";S$ @ C1$=FNE$(S$)
110 LINPUT "Cursor down (ASCII):","27,66";S$ @ C2$=FNE$(S$)
120 LINPUT "Cursor left (ASCII):","27,68";S$ @ C3$=FNE$(S$)
130 LINPUT "Cursor ON (ASCII):","27,62";S$ @ C4$=FNE$(S$)
140 LINPUT "Cursor OFF (ASCII):","27,60";S$ @ C5$=FNE$(S$)
150 LINPUT "Clear to bottom (ASCII):","27,74";S$ @ C$=FNE$(S$)
160 LINPUT "Scroll DOWN (ASCII):","27,83";S$ @ R0$=FNE$(S$)
170 LINPUT "Scroll UP (ASCII):","27,84";S$ @ R1$=FNE$(S$)
180 LINPUT "Display replace cursor (ASCII):","27,82";S$ @ C6$=FNE$(S$)
190 LINPUT "Display insert cursor (ASCII):","27,81";S$ @ C7$=FNE$(S$)
200 LINPUT "Clear display device (ASCII):","27,69";S$ @ E1$=FNE$(S$)
220 LINPUT "Clear screen page (ASCII):","27,72,27,74";S$ @ E$=FNE$(S$)
230 PRINT #1;M$,C1$,C2$,C3$,C4$,C5$,C$,R0$,R1$,C6$,C7$,E1$,E$
240 ASSIGN #1 TO *
250 END
    parse numbers in a list
260 DEF FNE$(I$)
270 O$="" @ S$=I$&","
280 IF LEN(S$)>1 THEN O$=O$&CHR$(VAL(S$)) @ S$=S$[POS(S$,",")+1] @ GOTO 280
290 FNE$=O$
300 END DEF
    parse subranges in a list
310 M$="" @ S$=S$&"@"
320 IF LEN(S$)<2 THEN RETURN
330 S=VAL(S$) @ E=VAL(S$[POS(S$,",")+1]) @ S$=S$[POS(S$,"@")+1]
340 FOR I=S TO E @ M$=M$&CHR$(I) @ NEXT I @ GOTO 320
```

```
===============================================================================
00/09/14 17:39:43 FILE: MKVEKEYS   BASIC  1389 09/14/00 17:39
===============================================================================
1 ! MKVEKEYS v.1.1 - MaKe KEY assignaments for VE
2 ! DON'T RENUMBER - DON'T use CHR$(220) in input definitions
3 ! use STOP to stop at inputs
4 ! use DEFAULT to accept default inputs
5 ! else enter your own input string
6 ! use STOP to stop at key-presses
7 ! else enter your own list (a string) of key-presses
8 ! fQ requires two input parameters (input $, list of key-presses)
9 ! BUGS: fQ STOP (list) won't work in VE
10 ! specifying DEFAULT as a key-press doesn't make sense and halts VE on error
11 ! fG, fB and #158 need LEN(list)=1 (or STOP)
19 ! The key to be defined...
20 DATA f7
29 ! and the definition...
30 DATA -,-,-,-,#48,-,#48,#48,-,#48,-,#48,#48,-,#48,-,#48,#48,-,fY,DEFAULT,fI,DE
FAULT
40 DATA #47,#47,-,#47,#47,-,fO
50 DATA |,#48,#48,#48,#48,|,#48,#48,|,#48,|,#48,#48,|,#48,|,#48,#48,|,fS,fS,#51,
fC,#50,fO
60 DATA |,#48,#48,#48,#48,-,-,-,-,#48,-,-,-,-,#48,|,#48,#48,|
70 DATA #48,#50,#50,!,#51,#47,!,#51,#47,!,#51,#51,#47,*
1000 DIM N$[95],K$[4],K1$[4],I$[92],O$[64]
1005 O$="#155#158  FF  FR  FQ  FY  FI  FP  FD  FG  FB  "
1010 READ K$ @ DISP "Defining [";K$;"]" @ N$=""
1020 ON ERROR GOTO 2000
1030 DISP @ READ K1$ @ DISP "[";K1$;"] "; @ GOSUB 1035
1031 IF UPRC$(K1$)="FQ" THEN GOSUB 1045
1032 GOTO 1030
1035 IF LEN(K1$)>1 THEN N$=N$&MAPKEY$(K1$) ELSE N$=N$&K1$
1040 IF NOT POS(O$,(UPRC$(K1$)&"   ")[1,4]) THEN RETURN
1045 READ I$ @ IF I$="STOP" THEN DISP I$; @ RETURN
1050 N$=N$&CHR$(220) @ IF I$#"DEFAULT" THEN N$=N$&I$ @ DISP "(";
1055 DISP I$; @ N$=N$&CHR$(220) @ IF I$#"DEFAULT" THEN DISP ")";
1060 RETURN
2000 OFF ERROR @ IF ERRN#32 OR ERRL#1030 THEN BEEP @ DISP ERRM$ @ END
2020 DEF KEY K$,N$; @ DISP "DONE"
===============================================================================
00/09/14 17:40:09 FILE: MKVEKEYT   TEXT   5376 09/14/00 17:40
===============================================================================
```

Program MKVEKEYS - version 1.1 - make one key assignment for VE
Necessary software: VELEX lex file.
INPUT: DATA string items in lines 1,999: 1st item is the name of the key to be
       defined; all following items make the definition
OUTPUT: a key definition (typing-aid)

Algorithm: in definitions command keys are stored in MAPKEY$ format (one byte per key) while text keys are represented literally.
Some command keys need one or two parameters (an input string or a list of key-presses, a list of key-presses). It is possible to specify inputs simply by keying them in as the item following the name of the command key. It is also possible to make VE stop at inputs - by keying in the item 'STOP' (capital letters) - or accept default values supplied at inputs - by keying in the item 'DEFAULT'. In definitions inputs are enclosed between a pair of CHR$(220)'s; DEFAULT is represented as CHR$(220)&CHR$(220) while STOP is represented as the null string. Byte 220 was chosen since MAPKEY$ can't return it.
EXAMPLE:

```
  'delete the first three empty lines' can be expressed as:
  DATA fQ,/\^$//,YYYQ
  and is represented as
  CHR$(169)&CHR$(220)&"/\^$//"&CHR$(220)&CHR$(220)&"YYYQ"&CHR$(220)
  NOTE: incidentally, I chose the worst example: the intrinsic complexity of fQ
  in key definitions is apparent (fQ is the only command which requires 2
  parameters). The above definition will work correctly only if there ARE at
  least three empty lines in the file). Also, due to the method of
  representation of key defitions, there is no way to represent  fQ,STOP,...
  correctly (anyway, I think it really doesn't make any sense).
specifying STOP as a list of key-presses will work, but DEFAULT won't (what's
its meaning anyway). Finally, only fQ should specify more than one key-press.
1 ! MKVEKEYS v.1.1 - MaKe KEY assignaments for VE
2 ! DON'T RENUMBER - DON'T use CHR$(220) in input definitions
3 ! use STOP to stop at inputs
4 ! use DEFAULT to accept default inputs
5 ! else enter your own input string
6 ! use STOP to stop at key-presses
7 ! else enter your own list (a string) of key-presses
8 ! fQ requires two input parameters (input $, list of key-presses)
9 ! BUGS: fQ STOP (list) won't work in VE
10 ! specifying DEFAULT as a key-press doesn't make sense and halts VE on
error
11 ! fG, fB and #158 need LEN(list)=1 (or STOP)
   example of key definition
19 ! The key to be defined...
20 DATA f7
29 ! and the definition...
30 DATA -,-,-,-,#48,-,#48,#48,-,#48,-,#48,#48,-,#48,-,#48,#48,-,
        fY,DEFAULT,fI,DEFAULT
40 DATA #47,#47,-,#47,#47,-,fO
50 DATA |,#48,#48,#48,#48,|,#48,#48,|,
        #48,|,#48,#48,|,#48,|,#48,#48,|,fS,fS,#51,fC,#50,fO
60 DATA |,#48,#48,#48,#48,-,-,-,-,#48,-,-,-,-,#48,|,#48,#48,|
70 DATA #48,#50,#50,!,#51,#47,!,#51,#47,!,#51,#51,#47,*
     N$: definition
     K$,K1$: key names
     I$: command parameter
     O$: list of command keys requiring parameters
1000 DIM N$[95],K$[4],K1$[4],I$[92],O$[64]
1005 O$="#155#158  FF  FR  FQ  FY  FI  FP  FD  FG  FB  "
1010 READ K$ @ DISP "Defining [";K$;"]"
     init definition
     @ N$=""
     while there's a DATA item
1020 ON ERROR GOTO 2000
        fetch it
1030    DISP @ READ K1$
        @ DISP "[";K1$;"] ";
        and process it
        @ GOSUB 1035
        if it's [f][Q] then fetch 2nd parameter
1031    IF UPRC$(K1$)="FQ" THEN GOSUB 1045
     endwhile
1032 GOTO 1030
     process item:
     if it's a command key
     NOTE: if you want to treat a command key as a text key then you'll need to
     enter ONE byte representing the key, not its KEY$ representation
1035 IF LEN(K1$)>1 THEN
```

```
        then encode it and store it
        N$=N$&MAPKEY$(K1$)
     else {it's a text key} store it
     ELSE N$=N$&K1$
     if it is a command key and it needs one parameter
1040 IF NOT POS(O$,(UPRC$(K1$)&"   ")[1,4]) THEN RETURN
        fetch next item
1045    READ I$
        if it's STOP
        @ IF I$="STOP" THEN
           put out STOP
           DISP I$;
        else
        @ RETURN
           store left separator
1050       N$=N$&CHR$(220)
           if it isn't DEFAULT (or STOP)
           @ IF I$#"DEFAULT" THEN
              store parameter
              N$=N$&I$
              put out a symbol of literal parameter
              @ DISP "(";
           endif
           put out parameter
1055       DISP I$;
           store right separator
           @ N$=N$&CHR$(220)
           and close literal input if not DEFAULT
           @ IF I$#"DEFAULT" THEN DISP ")";
        endif
1060    RETURN
     endif
     if 'no data' when expecting a parameter definition then abort
2000 OFF ERROR @ IF ERRN#32 OR ERRL#1030 THEN BEEP @ DISP ERRM$ @ END
     else define key as a typing-aid
2020 DEF KEY K$,N$; @ DISP "DONE"
================================================================================
00/09/14 17:41:23 FILE: KEYDATA    BASIC  1112 09/14/00 17:41
================================================================================
1 ! This file contains some examples of key definitions for use with MKVEKEYS
2 !
7 ! the following macro can be used to delete a block spanning all lines from th
e current line
8 ! to the first line containing an occurrence of a user-specified pattern
9 ! N.B. fO,fL unconditionally clears all marks.
10 DATA fO,fL,fS,fF,STOP,fS,fD,STOP
18 ! this macro deletes a block of 10 lines beginning at the current line
19 ! N.B. in fP Y1 is the current line.
20 DATA fO,fL,fS,fP,'Y1+9',fS,fD,STOP
29 ! This macro copies your file to mass memory and ends the editing session.
30 DATA fO,fL,fY,:MASSMEM,#43
37 ! This macro exploits the on-error behaviour of fJ to help you "compact" your
 text; simply
38 ! position the cursor on line 1 and press a few times the redefined key: the
macro attempts
39 ! to join two lines 20 times.
40 DATA fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ,fJ
49 ! this macro does some clean-up in your file (no empty lines or invisible cha
racters
50 DATA fV,fR,/\^$//,fE,fV
```

```
59 ! an essential macro for [f][S], [f][R] and [f][Q]
60 DATA CHR$(92)
69 ! [g][RUN] may need the following
70 DATA CHR$(95)
```

```
==========================================================================
00/09/14 17:44:13 FILE: VELEXS     TEXT  44544 09/14/00 17:44
==========================================================================
        LEX     'VELEX'
        TITLE   Visual Editor LEX file.
        ID      #5C
        MSG     0
        POLL    polhnd
        *
        ENTRY   chirp
        CHAR    #D
        ENTRY   cursor
        CHAR    #F
        ENTRY   mapkey
        CHAR    #F
        ENTRY   velist
        CHAR    #D
        *
        KEY     'CHIRP'
        TOKEN   06
        KEY     'CUR$'
        TOKEN   07
        KEY     'MAPKEY$'
        TOKEN   08
        KEY     'VELIST'
        TOKEN   09
        *
        ENDTXT
        *
****************************************************
* EQUATE TABLE
****************************************************
        ********************************************
        * SYSTEM ROUTINES
        ********************************************
               *************************************
               * >>>>>>>>> W A R N I N G <<<<<<<<< *
               *                                   *
               * !! means UNSUPPORTED ENTRY POINT.  *
               *                                   *
               * The listed addresses are good for *
               * version 1BBBB of the HP71 ROMS.   *
               *************************************
?PRFI+  EQU     #17380  Check file protection for security.
?PRFIL  EQU     #1737E  Check file protection for privacy.
ADHEAD  EQU     #181B7  Add string header to mathstack.
ARANGE  EQU     #045D1  !! Check if upper case character.
BSERR   EQU     #0939A  BASIC System Error.
CHIRP   EQU     #0EC5A  Do an annoying little beep.
COMCK+  EQU     #032AE  Check comma and output comma token.
CONVUC  EQU     #152AA  Convert to upper case.
D=AVMS  EQU     #1A460  Set D0 = address in AVMEMS.
FIBADR  EQU     #11457  Find FIB entry address for channel#
FILSK+  EQU     #06F1D  File skip.
FIXDC   EQU     #05493  Expression list decompile.
FIXP    EQU     #02A6E  Parse numeric expression and stop.
GETARG  EQU     #1113E  !! Get arguments from program line.
GETCH#  EQU     #11427  Get channel number.
GTKYC+  EQU     #08D9B  Get key code.
IDIVA   EQU     #0EC6E  A-field integer division.
```

```
MFERR   EQU    #09393  Mainframe BASIC system error.
NCH     EQU    #03356  Check for "#" (original name: #CH).
NUMCK   EQU    #0369D  Parse numeric expression.
NXTSTM  EQU    #08A48  Scan to next stmt, return to BASIC.
OUTBYT  EQU    #02CE8  Output one byte from C(B).
OUTELA  EQU    #05303  Output end of statement terminator.
POP1S   EQU    #0BD38  Pop 1 string argument off stack.
PRPSND  EQU    #06B17  Prepare to send buffer to display.
RNDAHX  EQU    #136CB  Pop-test-round-convert dec  to hex.
RSTD1   EQU    #1C596  !! Restore D1 from F-R0-1.
SAVD1   EQU    #1C578  !! Save D1 in F-R0-1.
SFLAG?  EQU    #1364C  Test system flag.
STKCHR  EQU    #18504  Add a character to a stack item.
SWPBYT  EQU    #17A24  Swap bytes.
SYNTXe  EQU    #02E2B  "Syntax" parse error.
aRANGE  EQU    #150FD  !! Check if lower case character.
        ***********************************************
        * SYSTEM RAM LOCATIONS
        ***********************************************
CHN#SV  EQU    #2F96F  Channel number save.
MLFFLG  EQU    #2F870  Multi-Line Function Flag.
OUTBS   EQU    #2F58F  Output Buffer Start.
STMTD1  EQU    #2F896  Statement scratch area.
S-R1-1  EQU    #2F886  Statement scratch area.
        ***********************************************
        * ERROR CODES
        ***********************************************
eIVARG  EQU    #0000B  "Invalid Arg".
eEOFIL  EQU    #00036  "End of File".
eFACCS  EQU    #0003C  "Invalid Access".
eFTYPE  EQU    #0003F  "Invalid File Type".
        ***********************************************
        * OFFSETS
        ***********************************************
oFTYPh  EQU    #00010  to file type in file header.
oFLAGh  EQU    #00014  to FLAG nib in file header.
        *
oPROTb  EQU    #00009  to protection nib in FIB.
oDEVCb  EQU    #0000C  to device type nib in FIB.
oFBEGb  EQU    #0000D  to file begin address in FIB.
        ***********************************************
        * STATUS FLAGS
        ***********************************************
sEOF    EQU    #00007  End of file.
sBADRC  EQU    #00008  Bad record.
        ***********************************************
        * SYSTEM FLAGS
        ***********************************************
flLC    EQU    00-15   Lower case mode flag.
        ***********************************************
        * PRINT CLASS STATEMENT TYPE
        ***********************************************
PRINTt  EQU    #00001  PRINT type.
*******************************************************
* END OF EQUATE TABLE.
*******************************************************
*******************************************************
* POLL HANDLER FOR VER$.
*******************************************************
polhnd  ?B=0   B       VER$ poll?
```

```
        GOYES  hver$   Yes: handle it!
        RTNSXM         No: return "Not handled".
hver$   C=R3           Recall stack pointer.
        D1=C           D1 @ mathstack.
        A=R2           Recall available memory.
        D1=D1- (veren)-(verst)-2
        * Make space for VER$, D1 @ new TOS.
        CD1EX          Recall new TOS.
        ?A>C   A       Memory ok?
        GOYES  hver$1  No: don't continue.
        D1=C           D1 @ new TOS.
        R3=C           R3 @ new TOS.
verst   LCASC  ' VE:1.1'
veren   DAT1=C (veren)-(verst)-2
hver$1  RTNSXM         Done with VER$ poll! DO NOT HANDLE!
**********************************************************
* END OF POLL HANDLER.
**********************************************************
        *
**********************************************************
**********************************************************
**
** Name: CHIRP
**
** Category: System command.
**
** File: VELEXS.DOC
**
** Purpose: Give a BASIC keyword to the mainframe chirp
**      routine.
**
** Syntax: CHIRP
**
** Entry: as for any statement execution.
**
** Exit: Through NXTSTM.
**
** Calls: CHIRP
**
** Uses: A, B, C, D, P, D0.
**
** Stack levels: 3
**
** History:
**
**   Date    Programmer             Modification
** --------  ----------  ---------------------------------
** ??/??/??  J.R. Baker  Designed and coded in BEEPLEX.
** 08/04/86  S. Tendon   Adapted for VELEX, changed
**                       tokens from 5E01 to 5C06.
**
**********************************************************
**********************************************************
        *
chirpd  GOVLNG OUTELA  Decompile routine.
        *
chirpp  RTNCC          Parse routine.
        *
        REL(5) chirpd  Offset to decompile routine.
        REL(5) chirpp  Offset to parse routine.
```

```
chirp   SETHEX          Needed by CHIRP routine.
        GOSBVL CHIRP    Call mainframe routine.
        GOVLNG NXTSTM   Return to BASIC interpreter.
        *
**********************************************************
**********************************************************
**
** Name: cursor.
**
** Category: Display utility.
**
** File: VELEXS.DOC
**
** Purpose: Produce a cursor positioning escape sequence
**      string.
**
** Syntax:
**      CUR$(<row>,<col>[,<maxcol>])
**
**      where <row>, <col> and <maxcol> are in the
**      range 0-255.
**
** Entry: Function execution entry conditions.
**
** Exit: Through ADHEAD or error exit.
**
** Calls:
**      RNDAHX, IDIVA, D=AVMS, STKCHR, ADHEAD,
**      or errors out via MFERR.
**
** Uses:
**      A, B, C, D, R0, R3, P, SB, XM, S7-S11
**
** Stack levels: 4
**
** Detail:
**      The CUR$ string is equivalent to:
**
**      CHR$(27)&"%"&STR$(<row>)&STR$(<col>)
**
**      If the optional <maxcol> parameter is given,
**      the CUR$ string is equivalent to:
**
**      CHR$(27)&"%"&
**        STR$(<row>+<col> DIV <maxcol>)&
**        STR$(<col> MOD <maxcol>)
**
** History:
**
**   Date    Programmer             Modification
** -------- ----------  ---------------------------------
** 09/12/86  S. Tendon   Designed and coded.
** 10/08/86  S. Tendon,  Recoded, optimized mainframe
**           S. Piccardi subroutine calls and catered for
**                       the <maxcol> optional parameter.
**
**********************************************************
**********************************************************
        *
invarg  LC(4)  eIVARG   Invalid argument error.
```

```
        GOVLNG MFERR   Mainframe error exit.
        *
        NIBHEX 88823   Parameter type and count
cursor  A=0    W       Initialize <maxcol> in case only
        R0=A             two parameters.
        P=C    15      Read number of parameters.
        ?P=    2       If only two parameters
        GOYES  cur2    then go and read 2 parameters.
        GOSBVL RNDAHX  Read <maxcol> into A(A).
        **************************************************
        * Entry: Number to be rounded and converted on
        *   top of math stack.
        * Exit:
        *   A(A) = rounded hex integer.
        *   Carry clear if negative. Carry set if non-neg.
        *   Fatal error if array or complex type, or NaN.
        *   HEXMODE, XM=0, P=0.
        * Uses: A, B(S,A), C(A), D(A), P, SB, XM
        **************************************************
        D1=D1+ 16      Pop <maxcol> item off mathstack.
        ?A=0   A       If <maxcol> is zero
        GOYES  invarg  then error out "Invalid Argument".
        R0=A           R0 = <maxcol>.
        *
        * Only two items are left on the stack now.
cur2    GOSBVL RNDAHX  Read <col> into A(A).
        D1=D1+ 16      Pop <col> item off mathstack.
        R3=A           R3 = <col>.
        GOSBVL RNDAHX  Read <row> into A(A).
        D1=D1+ 16      Pop <row> item off mathstack.
        *
        * Now D1 is pointing to the new TOS, where
        * the output string will be constructed.
        CD1EX          Prepare R1(A) for ADHEAD.
        R1=C
        D1=C           Restore D1 @ TOS.
        AR3EX          R3(A) = <row>, A(A) = <col>.
        C=R0           C(A) = <maxcol>.
        ?C=0   A       If <maxcol> is zero
        GOYES  skip    then assume no <maxcol>.
        GOSBVL IDIVA   A(W) = <col> DIV <maxrow>
        *              B(W) = <col> MOD <maxrow>.
        **************************************************
        * Entry: HEX or DEC mode according to arguments.
        *   Dividend in A(A), divisor in C(A).
        * Exit: Quotient in A(W), Remainder in B(W), C(W)
        *   Mode preserved, P=15, Carry clear.
        * Uses: A, B, C, P.
        **************************************************
        C=R3           C(A) = <row>.
        C=C+A  A       C(A) = <row>+<col> DIV <maxcol>.
        R3=C           R3 = <row>+<col> DIV <maxcol>.
        A=B    A       A(A) = <col> MOD <maxrow>.
        *
        * Now R3 = actual <row>, and A(A) = actual <col>.
skip    P=     0       Needed by following ADHEAD exit.
        GOSBVL D=AVMS  D(A) @ AVMEMS.
        LC(2)  27      ESCAPE character in C(B)
        GOSUB  stkchr    and to math stack.
        LCASC  '%'     '%' character in C(B)
        GOSUB  stkchr    and to math stack.
        C=A    A        <col> byte in C(B)
        GOSUB  stkchr    and to math stack.
        C=R3            <row> byte in C(B)
        GOSUB  stkchr    and to math stack.
        ST=0   0        Do not return from ADHEAD.
adhead  GOVLNG ADHEAD  Add string header and resume BASIC.
        *
stkchr  GOVLNG STKCHR  This saves a few bytes!
        *
**********************************************************
**********************************************************
**
** Name: mapkey
**
** Category: Keyboard utilities.
**
** File: VELEXS.DOC
**
** Purpose: Map key to a unique one-byte code, and return
**      code in a one-byte string.
**
** Syntax: MAPKEY$(<keycode string>)
**
** Entry: Function execution entry conditions.
**
** Exit: Through ADHEAD.
**
** Calls: SAVD1, POP1S, RSTD1, D=AVMS, STKCHR, ARANGE,
**      aRANGE, SFLAG?, CONVUC, GTKYC+
**
** Uses: A, B, C, D, P, D1, D0, S0-S11, function scratch.
**
** Stack levels: 7
**
** NOTE: idea came from KEYNUM function disassembly.
**
** Algorithm:
**      ! K$=KEYWAIT$
**      if K$ is longer than a byte
**         get keycode of K$
**         case (keycode)
**            null string   :  return (null string)
**            unshifted key :  return (keycode + 90)
**            f-shifted key :  return (keycode + 112)
**            g-shifted key :  return (keycode)
**         endcase
**      else if is_alfa (K$) return (toggle_case (K$))
**      else return (K$)
**
** History:
**
**   Date    Programmer             Modification
** --------  ----------  ---------------------------------
** 09/12/86  S. Tendon   Designed and coded.
**
**********************************************************
**********************************************************
        *
        NIBHEX 411     One string argument.
```

```
mapkey  SETHEX          Required by following POP1S
        GOSBVL SAVD1    Save TOS pointer.
        D1=C            .
        GOSBVL POP1S    Check if argument is a string.
        *************************************************
        * Entry: HEXMODE, D1 at string header.
        * Exit: P=0, D1 at last character of string,
        *       A(A) = string length, HEXMODE
        * Uses: A(W), D1, P.
        *************************************************
        C=0    A        If len(K$) # 1
        LCHEX  2        .
        ?A=C   A        .
        GOYES  onebyt   .
        GOSBVL RSTD1    then restore TOS pointer
        GOSUB  domap    . and do map.
        GONC   exit     B.E.T.
        *
        * else there is only one byte:
onebyt  A=DAT1 B        Read the only character.
        GOSUB  isalfa   If not alphabetic,
        GOC    exit     then leave it unchanged,
        GOSUB  ckcase   else toggle case if necessary.
        *
        * enter here with A(B) = character to return.
exit    GOSBVL RSTD1    Restore original TOS pointer.
        D1=D1+ 2        Skip stack signature.
        C=DAT1 A        Read string length
        D=C    A        .
        D1=D1+ 14       Skip rest of header
        CD1EX           Read current TOS pointer.
        C=C+D  A        Calculate new TOS, i.e. pop string
        D1=C            . and update pointer.
        R1=C            Prepare R1 for ADHEAD.
        GOSBVL D=AVMS   D(A) @ AVMEMS, C(A)=D1
        C=A    B        C(B) = charcter to be returned.
        GOSUB  stkchr   Write character to mathstack.
        *************************************************
        * Entry: C(B) = char, D(A) = (AVMEMS), D1 @ stack.
        * Exit: D1 at new stack char, carry clear.
        * Uses: D1
        *************************************************
        ST=0   0        Do not return from ADHEAD.
        GOTO   adhead   Add string header and resume.
        *************************************************
        * Entry: R1(A) @ start of stack item (hi mem)
        *       D1 @ End of stack item (low mem)
        *       S0 set if return, else exit via EXPR
        *       D(A) @ AVMEMS
        *       P=0.
        * Exit: in this case via EXPR.
        *************************************************
        *
*************************************************************
* Name: isalfa.
* Purpose: check if a charecter is alfabetic.
* Entry: P=0, A(B) = byte to be checked.
* Exit: P=0, Carry clear if charater is alfabetic.
* Calls: ARANGE, aRANGE
* Uses: C(A)
* Stack levels: 1
************************************************************
       *
isalfa  GOSBVL ARANGE
        RTNNC
        GOVLNG aRANGE
        *
************************************************************
* Name: ckcase.
* Purpose: Toggle case of a character if LC mode is on.
* Entry: HEXMODE, P=0, A(B)=Alfabetic character.
* Exit: HEXMODE, C(A)=D0, D(A)=D0, P=0.
* Calls: SFLAG?
* Uses: A(A), B(B), C(15, 5-0), D(A)
* Stack levels: 2
************************************************************
        *
ckcase  B=A    B       Store byte in B(B) during flag test
        LC(2)  flLC    if LC mode on...
        GOSBVL SFLAG?  .
        ****************************************************
        * Entry: C(B) hex flag number, HEXMODE, P=0.
        * Exit: Cary set if flag set, else carry clear,
        *      D(A) set to D0, HEXMODE, P=0.
        * Uses: A(A), C(15, 5-0), D(A).
        ****************************************************
        GONC   lcoff   .
        A=B    B       ...then toggle case:
        GOSBVL CONVUC  if lower convert to upper, else
        ****************************************************
        * Entry: A(B) = character, P=0, HEXMODE.
        * Exit: P=0, Carry set if no conversion required,
        *      A(B) = converted letter, not changed if
        *      carry set.
        * Uses: A(B), C(A).
        ****************************************************
        GONC   caseok  .
        LCHEX  20      .
        A=A+C  B       if upper, convert to lower.
        GONC   caseok  B.E.T.
        *
lcoff   A=B    B       Restore byte from flag test.
caseok  C=D    A       Restore D0 after SFLAG?.
        D0=C           .
        RTN
        *
************************************************************
* Name: domap.
* Purpose: map a multi-byte keycode string in one byte.
* Entry: D1 @ string on mathstack.
* Exit: A(B) = one byte key code.
* Calls: GTKYC+
* Uses: A-D, R0-R3, S0-S11.
* Stack levels: 6
************************************************************
        *
domap   ST=1   10      Null string legal.
        GOSBVL GTKYC+  Get key code.
        ****************************************************
        * Entry: Evaluated string on stack,
        *       S10=1: null string legal.
        * Exit: Carry clear: B(A)=keycode between 1 and A8
        *               A(A) = shift value (0, 56, 112).
        *       Carry set: B(A)=0, if null string was
        *               passed.
        * Uses: A-D, R0-R3, S0-S11.
        *************************************************
        GOC     null$   Exit if null string.
        ?A=0    A       Switch key...
        GOYES   nofg    .
        C=0     A       .
        LC(2)   56      .
        ?A=C    A       .
        GOYES   fshift  .
        A=B     A       g shift: leave unchanged.
        RTNCC           .
nofg    LC(2)   90      no fg shift: add 90.
        A=C     A       .
        A=A+B   A       .
        RTNCC           .
fshift  C=C+C   A       f shift: add 112.
        A=C     A       .
        A=A+B   A       .
        RTNCC
null$   A=0     A
        RTNCC
        *
*********************************************************
*********************************************************
**
** Name: vep
**
** Category: Parse routines.
**
** File: VELEXS.DOC
**
** Purpose: Parse VELIST statement.
**
** Entry: standard parse entry requirement.
**
** Exit: through FIXP, or error exit.
**
** Calls:
**      #CH (here called NCH), NUMCK, COMCK+, FIXP.
**
** Uses:
**  Inclusive: A-C, D(15-5), D0, D1, R0, R1, R3, P,
**              S0-S3, S7, S8, S11.
**
** Stack levels: 6
**
** History:
**
**   Date    Programmer             Modification
** --------  ----------  --------------------------------
** 07/09/86  S. Tendon   Designed and coded.
**
*********************************************************
*********************************************************
        *
```

```
vep     GOSBVL NCH      No "#"?
        GOC    errex1   Error out "Syntax Error".
        D1=D1+ 2        Step over "#".
        GOSBVL NUMCK    Parse channel number or error.
        GOSBVL COMCK+   Check and output comma or
        GONC   syntxe   error out if not found.
        GOSBVL NUMCK    Parse 1st argument or error.
        GOSBVL COMCK+   Check and output comma or
        GONC   syntxe   error out if not found
        GOVLNG FIXP     Parse 2nd argument or error.
        *
errex1  ST=1   4
syntxe  GOVLNG SYNTXe
        *
**********************************************************
**********************************************************
**
** Name: ved
**
** Category: Decompile routines.
**
** File: VELEXS.DOC
**
** Purpose: Decompile VELIST statement.
**
** Entry:
**      D1      @  Token stream, past the keyword token.
**      D0      @  output buffer, past keyword and blank.
**      A       =  Next token.
**      C       =  Next token.
**      D(A)    @  AVMEME.
**
** Exit:
**      Through FIXDC.
**
** Calls:
**      OUTBYT, FIXDC.
**
** Uses:
**  Inclusive: A-C, D1, D0, R0-R2, S0, S3, S8, S10, S11.
**
** Stack levels: 6
**
** History:
**
**   Date    Programmer              Modification
** --------  ----------  --------------------------------
** 07/09/86  S. Tendon   Designed and coded.
**
**********************************************************
**********************************************************
        *
ved     LCASC  '#'
        GOSBVL OUTBYT
        A=DAT1 B
        GOVLNG FIXDC
        *
**********************************************************
**********************************************************
**
```

```
** Name: velist
**
** Category: System Command.
**
** File: VELEXS.DOC
**
** Purpose:
**      This command is similar to PLIST, but with a
**      different syntax. Executes faster than PLIST.
**
** Syntax:
**      VELIST #<channel#>,<start record#>,<end record#>
**
**      NOTE: <start record#> and <end record#> may be
**      variable expressions.
**
** Entry: D0 @  past VELIST token.
**
** Exit: Through NXSTM or error exit.
**
** Calls:
**      GETCH#, GETARG, POSFIL, BSERR, FRCRDr, RCDSKP+,
**      PRPSND
**
** Uses:
**  Inclusive: A, B, C, D, R0-R3, D0, D1, sEOF, sBADRC
**             STMTD1, OUTBS, S-R1-1 and everything used
**             by expression execute.
**
** Stack levels: 6
**
** NOTE:
**      The GETARG routine is an unsupported entry point.
**      Code adapted from IDS vol. I, p. 17-21, 17-24/25.
**
**
** History:
**
**   Date    Programmer             Modification
** -------- ----------  ---------------------------------
** 07/09/86  S. Tendon  Adapted from indicated source;
**                      Changed GOSUB RCDSKP with CD0EX
**                      in the list routine (just before
**                      the 've50' label).
**
*********************************************************
*********************************************************
        *
        REL(5) ved     Offset to decompile routine.
        REL(5) vep     Offset to parse routine.
velist  GOSBVL GETCH#  Get channel# into CHN#SV, or exit.
        *************************************************
        * Entry: D0 @ channel#.
        * Exit: A(B) channel# in binary, D0 past channel#
        *   token, CHN#SV = channel#.
        *   Error exit if channel# >255 or <=0.
        * Uses: All CPU registers, status, scratch RAM
        *   except STMTR0, STMTR1. (EXPR is called).
        *************************************************
        GOSBVL GETARG  Get limiting record numbers.
        ***************************************************
        * UNSUPPORTED ENTRY POINT...
        * Entry: D0 @ tCOMMA or tEOL before arguments.
        * Exit: R2(A)=1st arg or 0, R3(A)=2nd arg or 0.
        * Uses: Everything execpt STMTR0 & STMR1.
        ***************************************************
        P=      0       Make sure P = 0 for next POSFIL.
        C=R2            Recall 1st parameter.
        A=R3            Recall 2nd parameter.
        ?C<=A   A       1st parameter <= 2nd parameter?
        GOYES   ve10    Yes: continue velist.
        *
        LC(4)   eIVARG  No: "Invalid Arg"
        GOTO    bserr   Error Exit.
        *
ve10    R1=C            Save 1st parameter for POSFIL.
        D1=(5) CHN#SV   D1 @ channel save byte.
        A=DAT1 B        Recall channel number.
        GOSUB   POSFIL  Find first line to list.
        GONC    ve30    Found? Yes: go on.
        ?C#0    A       No: do you have an error code?
        GOYES   bserr   Yes: exit with error.
        *
ve20    LC(4)   eEOFIL  Default error "End of File".
        GONC    bserr   Exit with default error, (BET).
        *
ve30    D=D-1   S       Check Device Code: File in Main?
        ***************************************************
        * Device Code:
        * 0 - Mainframe.
        * 1 - IRAM.
        * 2 - ROM.
        * 8 - HPIL.
        ***************************************************
        GOC     ve40    Yes: ok, continue.
        D=D-1   S       No: File in IRAM?
        GOC     ve40    Yes: ok, continue.
        LC(4)   eFACCS  No: "Invalid Access".
bserr   GOVLNG BSERR    Error exit.
        *
ve40    ?ST=1   sBADRC  Refuse to list bad record(s), and
        GOYES   ve20    error exit with "End of File".
        ***************************************************
        * Situation:
        * D1 @ start of first line.
        * D0 @ start of next line.
        * R0 = Record number of last record in file.
        * R1 = First record number.
        * R2 = First record number.
        * R3 = Last record number.
        * D(A) @ EOF.
        * STMTD1 @ FIB
        ***************************************************
        D1=D1+ 4        D1 @ data of 1st record to list.
        AD1EX           A(A) @ data of 1st record to list.
        AR3EX           A(A) = end rec# R3(A) @ SOD 1st REC
        *
        LC(2)  (PRINTt)*16+#F
        D1=(5) MLFFLG   Set up MLFFLG and STMTR0 for
        DAT1=C B        a PRINT class statement.
        *
        D1=(5) STMTD1
        C=DAT1 A        C(A) @ FIB entry.
        D1=C            D1 @ FIB entry.
        D1=D1+ (oFBEGb) D1 @ file begin address in FIB.
        C=DAT1 A        C(A) @ file begin address.
        D1=C            D1 @ file start.
        D1=D1+ 16       D1 @ start of data in file.
        D1=D1+ 16       .
        D1=D1+ 5        .
        CD1EX           C @ SOD of file.
        GOSUB  FRCRDr   Find end of last record to list.
        **************************************************
        * If you can find the last record then all records
        * up to (including) the last one are ok.
        **************************************************
        CD1EX           C @ EOF, EOD or last record...
        D1=C
        GOC    ve50     ...if error occured in FRCRDr.
        **************************************************
        * Carry set here means that the previous call to
        * FRCRDr produced an error. In this case C doesn't
        * point to the start of the desired record, but to
        * EOF or EOD, or to start of last record in file
        * if sBADRC is set. So this pointer will be passed
        * on to S-R1-1, i.e. the pointer beyond which
        * VELIST will stop.
        **************************************************
        CD0EX           C @ record past last record to list
        * NOTE: Original code was GOSUB RCDSKP. CD0EX gets
        * the same result, but executes faster.
        **************************************************
ve50    D0=(5) S-R1-1   Save pointer to EOF...
        DAT0=C A        .
        C=D    A        .
        RSTK=C          ...on Return Stack.
        *
        C=R3            C @ SOD of first record.
veloop  D1=(5) OUTBS
        DAT1=C A        OUTBS @ SOD in current record.
        D1=C            D1 @ SOD in current record.
        C=RSTK          C(A) @ EOF.
        D=C    A        D(A) @ EOF.
        GOSUB  RCDSK+   Skip current record.
        R0=C            R0 @ next record.
        C=D    A        C(A) @ EOF.
        RSTK=C          Save pointer @ EOF on Return Stack.
        GOSBVL PRPSND   Print line.
        **************************************************
        * Entry: P = 0, HEXMODE, B(A) = # of bytes in
        *   buffer, OUTBS @ start of buffer, R0 @ past EOL
        *   S-R1-1 @ EOF.
        * Exit: P = 0, buffer sent to display, C(W) = R0.
        * Uses: A, B, C, D, D1, D0, R1, R2
        **************************************************
        D1=C            D1 @ next record.
        D1=D1+ 4        Skip length header of record.
        CD1EX           C @ SOD of next record.
        GONC   veloop   (BET)
        *
```

```
*******************************************************
*******************************************************
**
** Name: POSFIL, POSTXT - Position Memory Text File to
**      Record n.
**
** Category: FILUTIL.
**
** File: VELEXS.DOC
**
** Purpose:
**      Position memory text file to given record. File is
**      indicated by channel number (POSFIL), or by file
**      header (POSTXT).
**
** Entry:
**      A(B)   =  Channel number (POSFIL only).
**      A(A)   =  File header address (POSTXT only).
**      R1(A)  =  Desired line number (first line = Ø).
**      P      =  Ø.
**
** Exit:
**      HARD ERROR EXIT if Channel# not open:
**             ("File Not Found").
**      ELSE:
**
**      sBADRC =  Set if D1 is positioned at a bad record.
**      R1     =  Entry condition.
**      P      =  Ø.
**
**      CARRY CLEAR: Desired record found.
**      D1     @  Abs address of start of line.
**      DØ     @  Abs address of start of next line.
**      RØ     =  Record number of last record in file.
**      B(S)   =  File protection nib from FIB.
**      D(A)   =  Abs address of EOF.
**      D(S)   =  Device code of file (POSFIL only).
**      STMTD1 =  FIB address (POSFIL only).
**
**      CARRY SET: Desired record not found.
**      sEOF   =  Set if D1 is positioned at EOF as
**                  defined by file chain.
**      C(A)   =  Error code:
**                  File is not in memory (POSFIL only).
**                  File is private.
**                  File is not TEXT file.
**                  Channel number not found.
**                  Premature EOF ("End of File").
**             =  Ø if requested line is not in file. D1
**                  is positioned at EOD or EOF. D1, D
**                  and R1 exit conditions are valid.
**
** Calls:
**      FILSK+, FRCRDr
**
** Uses:
**  Inclusive: A, B, C, D, RØ, DØ, D1, sEOF, sBADRC
**             STMTD1 (POSFIL only).
**
** Stack levels: 3
```

```
**
** NOTE:
**      Extracted from IDS vol. I p. 17-31/33.
**
** Algorithm:
**      Locate file FIB,
**             return error if channel# not found.
**      Verify that file is in memory.
**      Fetch file header.
**      Verify that file type is TEXT.
**      Verify that file is not private.
**      Compute file start, EOF.
**      Call FRCRDNr to locate record.
**      Set up exit conditions.
**
** History:
**
**   Date    Programmer             Modification
** -------- ----------  ---------------------------------
** 09/16/83 F. Hall     Designed and coded.
** 07/07/86 S. Tendon   Improved comments.
**
*********************************************************
*********************************************************
        *
POSFIL  GOSBVL FIBADR  Find FIB address (or error out).
        *************************************************
        * Entry: A(B) = Channel#.
        * Exit: D1, A, STMTD1 = FIB entry address.
        * Uses: A, B, C, D1, R0.
        *************************************************
        D1=D1+ oPROTb  D1 @ protection nib in FIB.
        A=DAT1 S       Read protection nib into A(S)...
        B=A    S       ... and into B(S).
        GOSBVL ?PRFIL  Private file?
        *************************************************
        * Entry: P = 0, A(S) = protection nib.
        * Exit: P = 0.
        *   CARRY SET:
        *   C(3-0) = File protection error code (eFPROT).
        * Uses: C(S), C(3-0).
        *************************************************
        RTNC           Yes: error out "File Protect".
        D1=D1+ (oDEVCb)-(oPROTb) D1 @ device code in FIB.
        C=DAT1 S       Read device code into C(S)...
        D=C    S       ... and into D(S).
        *              C(S) = 8 HEX if external file.
        C=C+C  S       External file?
        GOC    POSF40  Yes: Error out "Invalid Access".
        D1=D1+ (oFBEGb)-(oDEVCb) D1 @ file start address.
        A=DAT1 A       Read file start address into A(A).
POSTXT  D1=A           D1 @ file header.
        D1=D1+ oFTYPh  D1 @ file type in header.
        C=0    A       Make sure C(4) will be 0 for test.
        C=DAT1 4       Read file type into C(3-0).
        C=C-1  A       Check file type...
        ?C#0   A       ...not TEXT file?
        GOYES  POSF60  Error out "Invalid File Type".
        D1=D1+ (oFLAGh)-(oFTYPh) D1 @ file protection nib.
        GOSBVL FILSK+  File skip.
        ************************************************
        * Entry: P = 0, A(A) @ File header start.
        * Exit: P = 0, C(A) @ next file in chain (or to 00
        *   byte), A(A) = Length in file's length field,
        *   D1 @ file length field, carry clear.
        * Uses: A(A), C(A), D1.
        ************************************************
        D=C    A        D @ EOF.
        D1=D1+ 5        D1 @ Start of data (SOD) of file.
        CD1EX           C @ SOD, D1 @ EOF.
        A=R1            Recall desired record#.
        GOSUB  FRCRDr   Position to desired record.
        RTNNC           Return if record found.
        *
        * Continue here if error occured in FRCRDr.
        *
        C=0    A        Set up C(A) for error code.
        ?ST=0  sBADRC   If problen was EOF or EOD
        RTNYES          return.
        LC(2)  eEOFIL   "End of File".
        RTNSC
        *
POSF40  LC(4)  eFACCS   "Invalid Access".
        RTNSC
        *
POSF60  LC(4)  eFTYPE   "Invalid File Type".
        RTNSC
        *
********************************************************
********************************************************
**
** Name: FRCRDn, FRCRDr - Find given TEXT record.
**
** Category: FILUTIL
**
** File: VELEXS.DOC
**
** Purpose: Given TEXT file line #n (n>0), or record #n
**      (n>=0), locate that record.
**
** Entry:
**      A(A)    =  Desired line number or record number.
**      C(A)    @  Start of data in file.
**      D(A)    @  EOF according to file chain.
**      P       =  0.
**
** Exit:
**      R0      =  Current record number or FFFFF if no
**                   records in file; EOD mark is not
**                   counted as a record.
**      R1      =  Desired record number (>=0).
**      B(A)    =  Number of bytes of data in line
**                   according to line length header or
**                    FFFFF if incomplete header in corrupt
**                    record.
**      sBADRC =  Set if current record extends beyond EOF
**                   This indicates file is corrupt, can
**                   occure for two reasons: I) Only 1 byte
**                   left in file (line header requires 2
**                   bytes); II) Line header present but
**                     record length extends beyond EOF.
**      P      =  0.
**
**      CARRY CLEAR: Desired record found.
**      D1     @  Start of desired record.
**      D0     @  Start of next record.
**
**      CARRY SET: Desired record not found.
**      D1     @  EOF or EOD mark, or start of last record
**                 in file if sBADRC set.
**
** Calls:
**      PRSREC
**
** Uses:
**  Inclusive: A, B(A), C, R0, R1, D0, D1, sEOF.
**
** Stack levels: 2
**
** NOTE:
**      Extracted from IDS vol. I p. 17-33/35.
**
** Detail:
**
** Algorithm:
**      Current record# = -1.
**      Save current record address.
**      Clear sEOF, sBADRC
** 1.0  Parse record header,
**             return "Not Found" if no record.
**      Increment current record#.
**      If current record# = desired record#,
**             return "Found".
**      If sBADRC is clear
**             loop to 1.0
**      Else
**             return "Not found".
**
** History:
**
**   Date    Programmer              Modification
** --------  ----------  ---------------------------------
** 09/14/83  S.W.        Wrote routine.
** 07/07/86  S. Tendon   Improved comments.
**
*********************************************************
*********************************************************
        *
FRCRDn  A=A-1   A       Convert line# to record#.
FRCRDr  R1=A            Save desired record#.
        A=0     W       Current record# = -1
        A=A-1   A       .
        R0=A            Save current record#.
        ST=0    sEOF    Clear status.
        ST=0    sBADRC  .
FRCR10  GOSUB   PRSREC  Parse record header.
        RTNC            Return if no such record.
        *************************************************
        * NOTE: you return with the same exit conditions
        * you get from PRSREC when carry is set.
        ***************************************************
        D0=C            D0 @ start of next line.
        A=R0            Recall current record#.
        A=A+1  A        Increment current record#.
        R0=A            Save incremented current record#.
        C=R1            Recall desired record#.
        ?A=C   A        Are we at desired record#.
        GOYES  rtncc    Yes: return "Found".
        ?ST=1  sBADRC  Bad record?
        RTNYES          Yes: return "Not Found".
        ***************************************************
        * NOTE: Here you return with carry and sBADRc set.
        ***************************************************
        CD0EX           C(A) @ start of next line.
        GONC   FRCR10  Loop again (BET).
        *
***********************************************************
***********************************************************
**
** Name: PRSREC - Parse TEXT record header.
**
** Category: FILUTIL.
**
** File: VELEXS.DOC
**
** Purpose:
**      Examine the line length header of a TEXT file
**      record to determine line length for normal record,
**      or presence of End-Of-Data (EOD) mark, or presence
**      of End-Of-File (EOF), or absence of comlete line
**      header (corrupt file).
**
** Entry:
**      C(A)   @  Start address of record.
**      D(A)   @  EOF from file chain.
**      P      =  0.
**
** Exit:
**      D1     @  Start of record.
**      D(A)   @  EOF from file chain.
**      P      =0.
**
**      CARRY CLEAR: record exists.
**      B(A)   =  Number of bytes of data in record.
**      C(A)   @  Start of next record.
**      sBADRC =  Set if line goes beyond EOF
**                    else unchanged.
**
**      CARRY SET: Record not present,
**                    (@ EOF, EOD or no header).
**      B(A)   =  0 if @ EOF or EOD.
**             =  -1 if no line length header present.
**      sBADRC =  Set if no header present, else
unchanged.
**      sEOF   =  Set if EOF, else unchanged.
**
**
** Calls:
**      SWPBYT
**
```

```
** Uses:
**  Inclusive: A, B(A), C, D1, sEOF, sBADRC
**
** Stack levels: 1
**
** NOTE:
**      Extracted from IDS vol. I p. 17-35/36.
**
** Algorithm:
**      # bytes = 0.
**      If current position = EOF
**             Set sEOF.
**             Return "Not Found".
**      If line header is incomplete
**             Set sBADRC.
**             # bytes = -1.
**             Return "Not Found".
**      If line header = EOD mark (FFFF)
**             Return "Not Found".
**      Compute # bytes in line.
**      Compute start of next line.
**      If start of next line > EOF
**             Set sBADRC.
**      Return "Found".
**
** History:
**
**   Date    Programmer             Modification
** --------  ----------  ---------------------------------
** 09/19/83  FH          Adapted from code by SW.
** 07/07/86  S. Tendon   Improved comments.
**
*********************************************************
*********************************************************
        *
PRSREC  B=0     A       # bytes = 0.
        D1=C            D1 @ start of record.
        ?C>=D   A       At EOF?
        GOYES   PRSR10  Yes: return "Not Found".
        D1=D1+ 4        D1 @ start of data.
        CD1EX           D1 @ start of record, C(A) @ SOD.
        ?C>D    A       Line header missing?
        GOYES   PRSR20  Yes: return "Bad Record Not Found".
        A=DAT1 4        Read line header.
        GOSBVL  SWPBYT  Compute B = # bytes of data.
        *************************************************
        * Entry: A(3-0) bytes to be reversed.
        * Exit: A(3-0) reversed bytes.
        * Uses: A(A), C(A)
        *************************************************
        P=      3       .
        B=A     WP      .
        C=B     A       Test for EOD, compute # bytes.
        *               B(3-0) = FFFF hex if EOD.
        B=B+1   WP      Set carry if EOD.
        P=      0
        RTNC            Return "Not Found" if EOD.
        BCEX    A       B = #bytes, C = #bytes + 1.
        CSRB            Round to even #bytes (LIF stndrd).
        C=C+1   A       Compute total #nibs in record.
        C=C+C  A       . #bytes + 2 for header.
        C=C+C  A       . #nibs + 4 for header.
        AD1EX          A @ start of record.
        D1=A           .
        C=A+C  A       C = start of next record.
        ?C<=D  A       NOT corrupt record?
        GOYES  rtncc   Yes: return "Found".
        ST=1   sBADRC  Set "Bad Record".
rtncc   RTNCC          return "Found".
        *
PRSR10  ST=1   sEOF    Set EOF flag.
        RTNSC          Return "Not found."
        *
PRSR20  B=B-1  A       Set #bytes = -1.
        ST=1   sBADRC  Set "Bad Record".
        RTNSC          Return "Not found".
        *
**********************************************************
**********************************************************
**
** Name: RCDSKP, RCDSK+ - Record Skip.
**
** Category: FILUTIL
**
** File: VELEXS.DOC
**
** Purpose:
**      Skip over a TEXT fil record.
**
** Entry:
**      D1     @  Record start (prior to 2 byte length
**                  field).
**      D(A)   @  EOF from file chain.
**      P      =  0.
**
** Exit:
**      P      =  0.
**
**      CARRY CLEAR:
**      D1     @  Current recort SOD.
**      B(A)   =  Number of bytes of data in record.
**      C(A)   @  next record.
**      sEOF   =  Clear.
**      sBADRC =  Set if current record extends beyond EOF
**
**      CARRY SET: No record to skip.
**      D1     =  Entry conditione, either @ EOF, EOD
**                  (FFFF), or @ incomplete line header.
**      sEOF   =  Set if D1 @ EOF.
**      sBADRC =  Set if header incomlete, else clear.
**
** Calls:
**      PRSREC
**
** Uses:
**  Inclusive: A, B(A), C(A), E1
**
** Stack levels: 2
**
** History:
```

```
**
**   Date    Programmer               Modification
** --------  ----------  ---------------------------------
** 09/14/83  SW          Wrote routine.
** 09/19/83  FH          Adapted for FILSZR.
**
**********************************************************
**********************************************************
        *
RCDSK+  D1=D1- 4        D1 @ Start of record.
RCDSKP  CD1EX           C @ Start of record.
        ST=0   sEOF     Present status.
        ST=0   sBADRC   .
        GOSUB  PRSREC   Parse record.
        RTNC            Retrun if no record to skip.
        D1=D1+ 4        D1 @ SOD.
        RTNCC           Return OK.
        *
        END
```