A FULL SCREEN EDITOR IMPLEMENTED IN PASCAL

by

THEODORE JOHN SOCOLOFSKY

B.S., KANSAS STATE UNIVERSITY, 1980

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

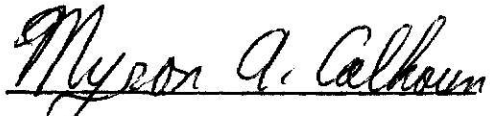
Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1981

Approved by:



Major Professor

ACKNOWLEDGEMENTS

I would like to thank Professor Ed Basham for revising the introduction, and Nassrin Tavakoli for paraphrasing parts of chapter 2 of "Pascal Microengine Computer Pascal Operator's Manual" (release A0 December, 1979) for sections of chapter 2 in this document. I thank Kim Janne for his valuable hints.

TABLE OF CONTENTS

SEDIT INTERNAL IMPLEMENTATION

1. INTRODUCTION

SEDIT (for S creen EDIT or) is a full screen editor. This editor provides a CRT terminal user an efficient means of editing information displayed on the screen. This editor, in contrast to most editors, provides the user a visual machine interactive means to move, correct, change and manipulate displayed text files. Accordingly, full screen editors are much more efficent and effective then other usual types of editors.

SEDIT is written in PASCAL language and implemented on Perkin-Elmer OS/32 MTM operating system. PASCAL was chosen because of its portability to other machines and operating systems. PASCAL is also a program language that is easy to maintain and modify. SEDIT is designed to operate on a variety of terminals, including both dumb and smart CRT terminals. This versitility is achieved through a modular design such that selected modules can be resident in smart terminals thus reliving the central processor of part of the work. Other modules can be modified such that SEDIT is blind to venders differences in CRT terminal characteristics.

The following paragraphs provid a detailed description of SEDIT.

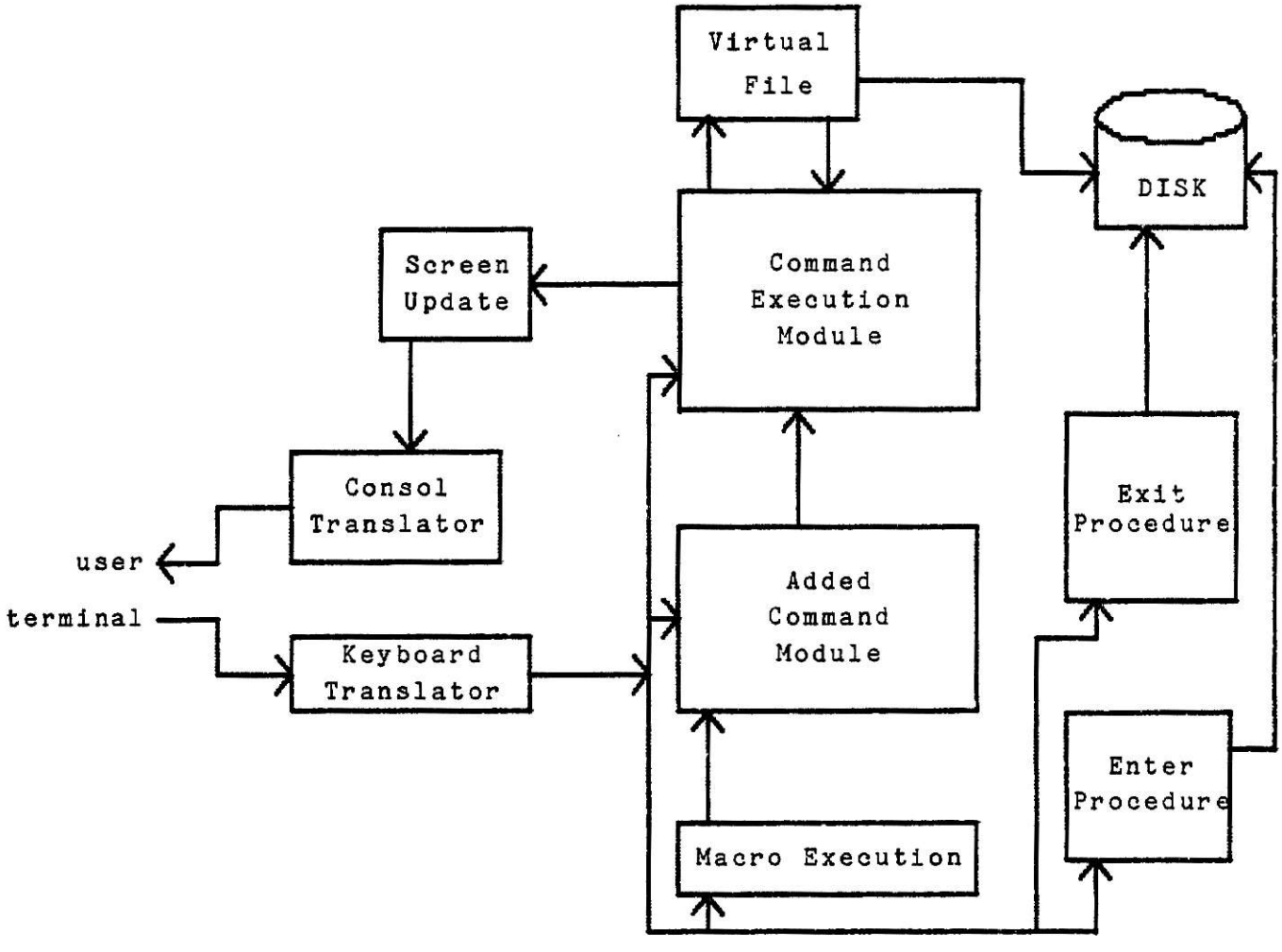
2. MODES OF SEDIT

SEDIT operates in two modes: Command and Modify mode. When in

command are ignored. The modify mode is used to change or correct an existing text file. The modify mode too has a set of valid acceptable inputs. The terminal operator can easly change from one mode to the other by means of the terminals escape key or control-C key.

3. ORGANIZATION OF SEDIT

SEDIT is organized around a disk file and a set of modules and procedures as shown at figure 1. A discussion of the pertenant modules follows:



ORGANIZATION OF SEDIT

3.1 Command Execution Module

The main part of SEDIT is the Command Execution Module. It is a set of procedures that are the primitive operations that can be performed on a file. The Command Execution Module is not concerned with file management or details of screen update. Its purpose is to execute the command. Figure 1 shows the relation of the Command Execution Module to the rest of SEDIT.

3.2 Added Command Definition Module

The use and scope of SEDIT can be changed by adding new commands. The definition of these new commands are, in effect, production rules of a grammar which translates the new commands (nonterminals) into the primitive commands (terminals) found in the Command Execution Module. Each production rule is expressed as a block of Pascal code found in the Added Command Definition Module. See Figure 1 for the relation of the Added Command Definition Module to SEDIT. The system programmer can thus modify SEDIT with the full power of Pascal. The programmer will have access to global variables in SEDIT and will not be concerned with file management or details of screen update. The programmer will only have to be aware of useful global variables and procedure names (the primitive commands) from the Command Execution Module.

By the selection of added commands, the system programmer can cause SEDIT to act like another editor. Thus SEDIT can be tailored to look like an editor that is familar to the terminal operator.

3.3 Macro Execution Module

SEDIT contains a powerful macro definition facility. At any time

during the execution of SEDIT the user can define and use commands called macros. These operations are processed by the Macro Execution Module (see Figure 1). The definitions include the name of the macro, may include repeat factors, and a series of other commands. These macro definitions further extend the flexibility of SEDIT and the ability to configure SEDIT to look like another editor. Once a Macro is defined, it is permenant for that file, and would be lost only upon EXIT of SEDIT. This is because of the capability of a Descripter File associated to each file. The Descripter Files are automatically created by SEDIT for each file that the user creates. More explanations are given in later sections. The name of the macro is not something like 'M1' but is whatever the user defines. The macro can be composed of any of the commands that are previously defined. It can contain nested parentheses and repeat factors. The macro can also contain variables to express a character string that is defined when the macro is executed.

3.4 Console And Keyboard Translators

SEDIT can be configured to different terminals with the Console And Keyboard Translators, (see Figure 1).

When SEDIT operates in full screen mode, ASCII code control characters are sent to the console to move the cursor on the screen. Because these control characters are not standard, SEDIT must account for the brand and kind of terminal being used. This is accomplished by means of a table of terminal characteristics. Then the proper file may be read to configure SEDIT output to the terminal. The table will have to be maintained by the system programmer. The file will contain

information about both input from the key board and necessary output to the console screen.

3.5 Virtual Files System

Another feature of SEDIT is the hidden file manipulation routines. See the Figure 1. The user can page through a file of any size both forward and backward without having to use any commands to read or write to disk. This manipulation of the file should look to the user as if all of the user's file were in memory. The use of memory for buffer space can be minimized and the manipulation of large files is not different from the manipulation of small files. This is accomplished most simply by the following procedure:

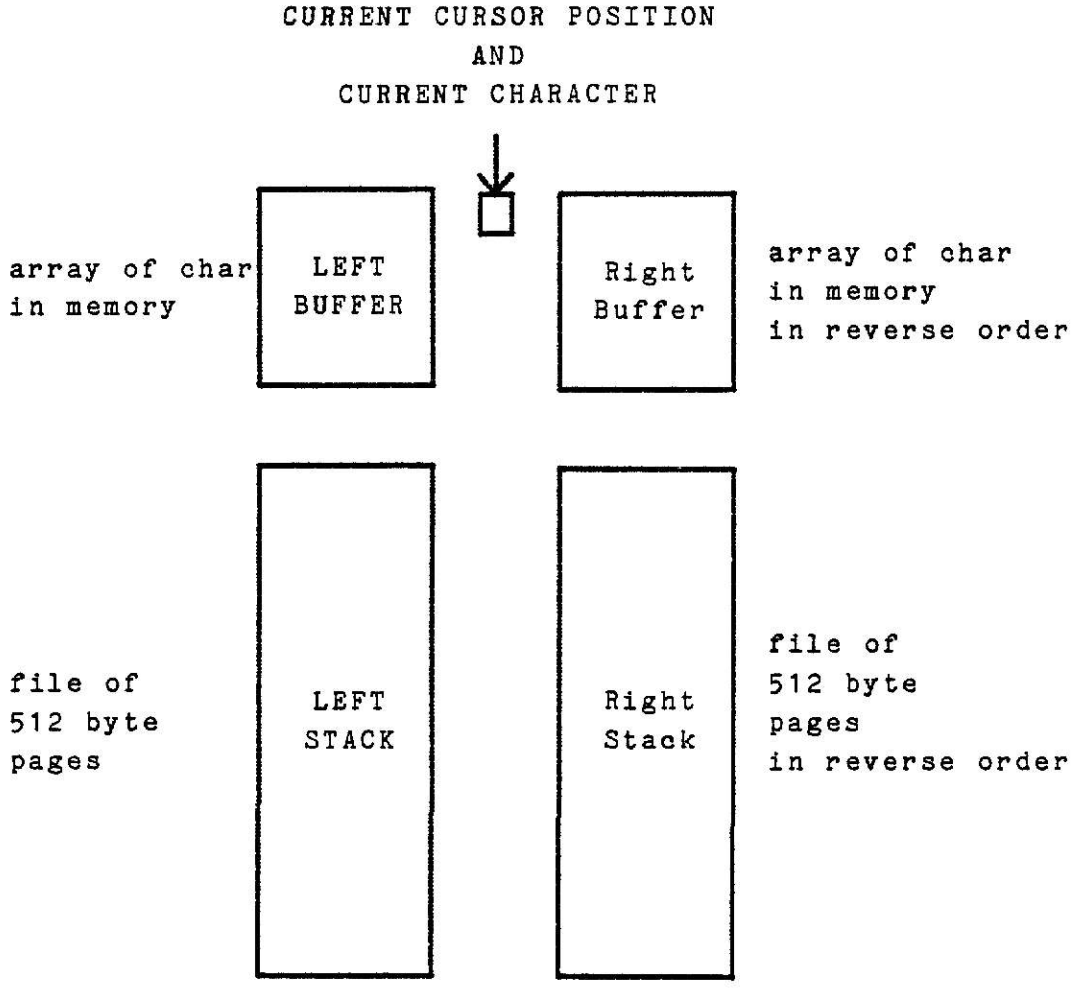
a) SEDIT first copies the file to be edited (if there is one) into a file called the Right File Stack. This file is written, a page at a time in reverse order. The last page of the user file is then the first page of the Right File Stack. See Figure 2.

b) Another file is allocated that is called the Left File Stack. This file is written a page at a time, but the pages are stored in first-to-last sequence.

c) There are also two buffers in memory called the Left Buffer Stack and Right Buffer Stack. Each buffer is a string of characters and the Right Buffer Stack contains the string in reverse order. The character at which the cursor is pointing is called the Current Character. The cursor position, or point of operation of commands, is always at the current character between the buffers.

d) When the user adds text, or moves the cursor down or to the left, characters are added to the Left Buffer Stack. When the Left Buffer

Stack is full, half of the buffer will be emptied by writing to the Left File Stack. This operation is hidden from the user. When the user is moving the cursor up or left, characters are taken from the Left Buffer Stack and placed into the Right Buffer Stack. When the Right Buffer Stack is full, it will be half emptied by writing to the Right File Stack. If at any time the Left or Right Buffer Stacks are emptied, they will be half filled by reading from the correct File Stack. This process will keep all disk I/O to a minimum and it will be done with efficient page oriented I/O.



RUN TIME EDIT BUFFERS

4. TERMINAL I/O

Output to the terminal is done with SVC1 calls that reside in the procedure SHIP_OUT which will output a line of known length. The output is done in Image mode.

Input is done with an SVC1 call in the Procedure INPUT. The call is Image mode Read. The echo back to the terminal is automatic and done by the programmable I/O device called a PASLA (Programmable Asynchronous Serial Line Adapter). This automatic echo is not desirable in this editor because it would prohibit the redefinition of characters. The function of echoing the character back is called ECHOPLEX. This function must be turned off when the editor is in use and turned on when done. The calls to control the echoplex should reside in the CSS file that evokes SEDIT. Then if SEDIT crashes (which is very unlikely) the execution of the CSS file will continue and the echoplex function will be enabled again.

The redefinition of the characters can occur in macro defination, consol translator or the keyboard translator.

5. FILE I/O

File I/O will be done with the prefix procedures OPEN, CLOSE, PUT, and GET. These procedures support page I/O.

6. FILE TYPES

SEDIT will support clean files. This means the files do not contain nonprinting characters, such as control characters, nor special characters such as paragraph indicators, that might be used for special

purposes in some editors. The files are the same as files currently supported with other programs such as Script, PPP, and the Pascal compilers.

7. WORK FILE

SEDIT will support a workfile. A workfile will store the default values of status variables. It will keep the name of the file the user was last working on. The user will be asked if he wants SEDIT to read in the file that he last edited. If the user replies with a yes, the file will be read and its descriptor file will be read. If there is not a descriptor file associated with the indicated file, the default values of the workfile will be used.

8. DESCRIPTOR FILE

With each file the user uses, there will be an associated file called the file descriptor. It will have the same name as the file but a different extension. The default file descriptor is called the workfile. Like the workfile, the descriptor contains information about the SEDIT status variables that determine the environment of SEDIT and also contains information that pertains to that file.

9. STATUS VARIBLES

The status varibles are the global varibles in SEDIT that describe the current status of the editor. The command execution module and added command definition module will have access to these variables. Some commands will directly modify the status variables and some indirectly.

A list of the Status Variables follows:

Current screen column cursor position : integer

Current screen row cursor position : integer

Current line cursor position : integer

Current column in line cursor position : integer

Current character at cursor : char

eofc : boolean , if eofc is true the cursor is pointing at the last character in the file, end of file char.

tofc : boolean , if tofc is true the cursor is pointing at the first character in the file, top of file char.

eofl : boolean , if eofl is true the cursor is in the last line in the file, end of file line.

tofl : boolean , if tofl is true the cursor is in the first line in the file, top of file line.

left margin : integer , column number

right margin : integer , column number.

automatic indentation : boolean , if true and in insert mode the next line will begin at the same column as the last one.

last line starting column : integer

script : boolean , if true the text will automatically be adjusted in the left and right margins.

token definition : boolean , if true the search argument applies to words, if false it applies to character strings.

current date : integer

last date : integer , last data the file was updated.

current search argument : char string

current change string : char string

paragraph margin : integer , column number.

tab sets : array of integer , column numbers.

name of current file : char string

starting anchor : interger , absolute position in the file.

ending anchor : integer , absolute position in the file.

current direction : char

line mode : boolean , if true line numbers will be shown.

USERS MANUAL

1. ENTERING SEDIT

If the user types SEDIT [filename][.ext], where filename and the extension are optional, the full screen editor will be run. If the user types the filename and not the extension, .PAS will be tried first. If the file is not found then .TXT will be used. If the file is not found the following menu will be presented to the user.

```
Input [filename][.ext] file not found.
PAS and TXT was searched for.
T)ype new file name.
c)reate new file.
```

The [filename][.ext] is where the name typed by the user is echoed. If 'T' is typed, SEDIT will ask the user to enter the new file name. If 'C' is typed, the current file name and extention becomes the new file. If the extension was typed by the user the above menu will appear without the second line and the extension will be echoed.

If the user does not type in the file name, SEDIT will automatically read the default descriptor file called the work file. The file name is SEDIT.DES, and a copy exists on each account. The work file and descriptor files contain values necessary to configure SEDIT. The name of the last file the user edited with SEDIT is stored in this work file.

SEDIT then gives the user the following menu.

The file last edited was
filename.ext
touch return to use this file,
else type in the new file name.

SEDIT responds with the same search as discribed above, where the user does not have to type in the extension.

SEDIT will then search for the proper descriptor file which has the name filename.DES. If the file is found, it is read and the environment of SEDIT is set the way it was when SEDIT was last used with this file. If the file is not found, the default values from SEDIT.DES will be used and the user will be informed of this action. Once the file to be edited is specified or found, the following prompt line will be displayed:

>SEDIT: I(nsert D(elete Q(uit R(place J(ump F(ind P(age A(djust ?(more

The '>' on the left side of the prompt line indicates the current status of direction. If the user types '?' an extension of the prompt line will appear as follows;

>SEDIT: X(change M(argin C(opy S(et H(ex E(xamine V(erify

2. EXITING THE EDITOR

The user may exit the editor by typing 'Q' for QUIT. On entering the quit mode, the following prompt will be displayed.

>Quit:

U(pdate - the current file and leave

E(xit - without updating

R(eturn - to the editor without updating

W(rite - to a new file

P(rint - to an output device

One of the five options must be selected by typing the appropriate letter. The options are described below:

U(pdate - The Editor will replace the old file with the file just modified and output the file's descriptor file.

E(xit - The Editor leaves without making any changes to the file nor to the descriptor file.

R(eturn - The Editor returns without updating, the cursor is returned to the exact place in the file it occupied when 'Q' was typed.

W(rite - With this option, a further prompt appears:

>Quit: 'name of output file', <return>

The modified file will be written to the indicated file name and the descriptor file will be output. If it is an existing file, the modified file will replace it. SEDIT, however, warns the user of the existing file

>File xxx.xxx already existed. do you want to replace the file?

"N" will let the user to enter a different file name

"Y" will replace the file

The command can be aborted by entering a <return> instead of the file name; return will be to the Editor.

After the file has been written to the disk the prompt line will be:

>Quit:

Writing . . . . . .

Your file is nnnn bytes long.

Do you want to E(xit from of R(eturn to the Editor?

Typing "E" exits from the Editor. Typing "R" returns the cursor to the same position in the file as when the "Q" was typed.

Although not discussed above, the user will also be given the option to output a file with line numbers, and the option to output a file to a printer.

P(rint - A prompt will appear and the user may type in the name of the output device to print to. The user can have the lines numbered if she/he wishes.

3. PRIMITIVE COMMANDS

These commands are used for cursor movement, or modifying text. REPEAT FACTORS are allowed by some of these commands to repeat the action of the command as many times as indicated by the immediately proceding number. For example, entering

2 <down-arrow>

will cause the <down-arrow> command to be repeated twice, moving the

cursor down two lines.

A slash (/) typed before the command indicates an infinite number of repeats.

Some commands are directional. If their direction is forward, they operate forward through the file; if backwards, they operate in the reverse.

3.1 MOVING COMMANDS

These commands move the cursor from one location to another, to position it for the next editing function.

The REPEAT FACTORS may be used with all of these commands. They include:

<ctrl>H, <left arrow>	moves cursor left
<ctrl>L, <right arrow>	moves curor right
<ctrl>K, <up arrow>	moves cursor up
<ctrl>J, <down arrow>	moves cursor down
"<"	changes the direction to backward
">"	changes the direction to forward
<space>	moves direction
<back-space>	moves left
<return>	moves to the beginning of the next line

Direction is always indicated by an arrow (> or <) in front of the prompt line. The direction is forward when the Editor is entered, but can be changed by typing the appropriate command whenever the "Edit:" prompt line is present.

Commands that do not have special function keys on the CRT keyboard are as follows:

3.1.1 JUMP

3.1.2 PAGE

3.1.3 FIND

3.1.1 JUMP

JUMP mode is entered by typing "J" for J(ump . When the JUMP mode is entered, the following prompt line will be displayed:

JUMP: B(eginning E(nd, <+n>, <-n>, <d>

"B" moves the cursor to the beginning of the file.

"E" moves the cursor to the end of the file

"<+n>" moves the cursor n lines forward

"<-n>" moves the cursor n lines backwards

"d" moves the cursor to line number d Note that n must be an integer, whereas d may be an integer or real number

3.1.2 PAGE

The PAGE command is executed in response to typing "P". PAGE moves the cursor one whole page up or down, depending on the direction of the arrow at the beginning of the prompt line. The cursor moves to the start of the top line. To move several pages at one time, the repeat factor may be used with this command.

3.1.3 FIND

FIND mode is entered by typing "F" . On entering FIND mode, one of the prompt lines below will appear:

>Find[n]: S(tring <target>=>

>Find[n]: W(ord <target>=>

FIND mode finds repeat factor [n] number of occurances of the <target> string. Text is scanned in the direction of the arrow on the prompt line. The target pattern can be found only if it appears in that section of the text.

In STRING mode, the Editor will look for any occurances of the target string, in WORD mode, it looks for an isolated occurance. Isolation means a string is surrounded by any combination of delimiters. To use the STRING mode, type "S" after the prompt line and before the target line, to use WORD, type "W". The default value found in the Environment may be overridden by typing "S" or "W".

The repeat factor can be used, and it must be typed before typing "F".

3.2 MODIFYING COMMANDS

The following operations are used to change the text of a file.

3.2.1 INSERT

3.2.2 DELETE

3.2.3 COPY

3.2.4 EXCHANGE

3.2.5 REPLACE

3.2.6 DISPLAY HEX

3.2.7 EXAMINE SPELLING

3.2.8 MARGIN

3.2.9 ADJUST

3.2.1 INSERT

INSERT mode is entered by typing "I" for I(nsert . The following prompt line will be displayed:

Insert: <bs> a char, <del> a line, <esc>, <ctrl>C

The user can simply start typing and the text will be inserted into the file at the point where the cursor is. The text to the right side of the cursor will shift to the right or to the next line.

"<bs> a char"	a character is deleted by backspacing
"<del> a line"	an entire line is deleted
"<esc>"	all inserted information will be deleted
"<ctrl>C"	all inserted information is put into the file, and put in the copy buffer. all previous information is the copy buffer is lost

3.2.2 DELETE

DELETE mode is entered by typing "D". The following prompt line will be displayed:

>Delete: <moving commands>, <esc>, <ctrl>C

When entering the delete mode, the cursor is positioned at the first character to be deleted. This position is called starting anchor. The user can move cursor around using the moving commands. The point at which the cursor is when the delete mode is exited is called the ending archor.

<esc>	exits edit mode without any changes to the file
<ctrl>C	exits edit mode, deletes the text between the starting and ending archors

3.2.3 COPY

COPY mode is entered by typing "C" for C(opy . When entering the COPY mode, the following prompt line appears:

>COPY: B(uffer F(ile <esc>

"B" copy text in the copy buffer into the file at the location of the cursor when "C" was typed.

On completion of the COPY, the cursor returns to a position immediately preceding the text that was copied, The use of the COPY does not change the contents of the copy buffer.

"F" copy text from another file.

The following prompt line will appear:

>COPY: ENTER THE FILE TO COPY FROM OR "END"

"file name"

>COPY: ENTER FIRST LINE OR "END"

"first line"

>COPY: ENTER LAST LINE OR "ALL"

"last line"

>COPY: ENTER THE FILE TO COPY FROM OR "END"

"end"

The "F" command allows the user to copy several files or different sections of different files.

The use of COPY does not change the contents of the file being copied.

3.2.4 EXCHANGE

EXCHANGE mode is entered by typing "X". On entering the EXCHANGE mode, the following prompt line appears:

EXCHANGE: TEXT <bs> a char, <esc>, <ctrl>C

EXCHANGE replaces one character in a file for each character of text entered. Backspacing one character will cause the original character in that position to reappear. Typing <esc> leaves the EXCHANGE mode without making any of the changes indicated since entering the mode. Typing <ctrl>C accept the changes as part of the file.

3.2.5 REPLACE

REPLACE mode is entered by typing "R". On entering the REPLACE mode, one of the following prompt lines will appear.

>Replace[n]: S(tring V(fy <target> <sub> =>

>Replace[n]: W(ord V(fy <target> <sub> =>

If repeat factor is used, it should be typed before "R". REPLACE mode finds the repeat factor [n] occurrance of the target string and replaces it with the substitute string.

The verify option V(fy allows examination of the target string (to the limit set by the repeat factor) to decide if it is to be replaced. Whenever REPLACE has found the target pattern in the file and verification has been requested, the following prompt line appears:

>Replace: <esc> aborts, 'r' replaces, ' ' doesn't

Typing an "R" will cause replacement, typing a space will cause a continuation of a search for the next occurance if the limit of the repeat factor has not yet been reached. The repeat factor counts the number of times an occurance is found, not the number of times "R" is entered. If a slash(/) is used as the repeat factor, every occurance of the target string will be replaced.

If the Editor cannot find the target string, the error message appears:

ERROR: Pattern not in the file. Press <space bar> to continue.

Example:

>REPLACE [2]: S(tring V(fy <target> <sub) => old-string/new-string/

>REPLACE [2]: S(tring V(fy <target> <sub> => W/old/new/

Note that the slash (/) is used here as a delimiter, you may use any other character as a delimiter besides "S", "W", and "V".

3.2.6 DISPLAY HEXDECIMAL

The command display hexadecimal is executed by the user typing 'H' for hex. The current line (which the cursor is in) is displayed. Followed by a line of the first digit of the hexidecimal number for each character. Followed by a line of the second digit of the hexidecimal number for each character.

An example follows.

```
  Contents of SEDIT
2246676672662544450
003F745E430F0035494A
```

3.2.7 EXAMIN SPELLING

SEDIT will contain a facility for the user to check spelling. When the user types 'E' for Examine Spelling the spelling of all the words from the current cursor position to the end of the current line will be checked. The repeat factor when used will cause more lines to be checked or the rest of the file. A word is defined as a character string delimited by spaces, where a sequence of hyphen-linefeed is ignored and a period is treated as a following delimitor. A word is said to be spelled correctly if it occurs in the dictionary. The dictionary is a file (DICTION.TXT) that SEDIT has access to that is just a list of words spelled correctly. If the word does not exist in the dictionary, or is spelled incorrectly the cursor is placed immediately

following that word, and the user is queried with the prompt line.

Spelling error. C)orrect spelling, <ctrl>C continue, <esc> escape

If the user types 'C', this indicates that the word is spelled correctly and it is placed in the dictionary. If the user types the <esc> key, he then exits from the spelling mode and returns to SEDIT mode where he can correct the spelling. If the user types the control 'C' key, the spelling mode is continued and the next word is checked.

3.2.8 MARGIN

The MARGIN command is executed by typing "M". This command does not appear on the prompt line, MARGIN is Environment dependent and cannot be executed except when FILLING is set to true and AUTO-INDENT is set to false. See SET command to set the Environment.

Three parameters are used by margin: right-, left- and paragraph-margin. MARGIN deals with only one paragraph at a time. It realigns the text to compress it as much as possibe without violating the three margins. To set the margin values, see SET mode.

For purposes of formatting, a paragraph is defined as the lines of text occuring between two blank lines. To MARGIN a paragraph, move the cursor anywhere in the paragraph and type "M".

Any given line of text can be protected from being MARGINed if the command character appears as the first non-blank character on the line. The MARGIN treats that line as though it were entirely blank.

3.2.9 ADJUST

ADJUST mode is entered by typing "A". On entering the ADJUST mode, the following prompt line appears:

>Adjust: L(eft R(ight C(enter <left,right,up,down-arrow> <ctrl>C

ADJUST mode adjusts indentation on a line-by-line basis. On any line, the right-arrow and left-arrow commands move the whole line one space to the right or left, respectively, each time the arrow is typed. Type <ctrl>C when indentation is adjusted.

To adjust a sequence of lines, adjust one, then use the up-arrow and down-arrow commands to adjust the line above or below, respectively. Repeat factors can be used before any of the arrows.

"L" and "R" are used to left- and right-justify lines to margins set in the Environment. "C" will center the line between the set margins. Typing an up- or down-arrow will justify or center the line above or below to the same specification.

4. SETTING FORMAT AND MACROS

SEDIT allows for formatting of text. A SET command is used to set the Environment for each of the formatting commands.

The macro definitions are entered with the set command.

SET mode is entered by typing "S". This command does not appear on the prompt line. On entering the SET mode, the following prompt line appears:

>Set: E(nvironment, M(acros, <esc>

4.1 FORMAT CONTROL

Through SET mode, SEDIT enables the user to set the Environment needs of the editing to be done. When "E" for E(nvironment is typed,

the following prompt appears:

```
>E(nvironment:  <options> <ctrl>C
     A(uto indent          true
     F(illing              false
     L(eft margin          0
     R(ight margin         79
     P(ara margin          5
     W(ord def             true
```

By typing the appropriate letter, any or all of the options may be changed. Each option is described below.

4.1.1 AUTO INDENT

AUTO ANDENT is set to true by typing "AT" and to false by typing "AF". Affects only the INSERT mode.

4.1.2 FILLING

FILLING is set to true by typing "FT" and to false by "FF". Affects the INSERT mode and allows the MARGIN command to function.

4.1.3 MARGINS

MARGINS are set by typing the appropriate letter ("L", "R" or "P") followed by a positive integer of less than 4 digits and a <space>. When Filling is true, the margins set here are the ones that affect INSERT and MARGIN and the center and justify commands in ADJUST.

4.1.4 WORD

WORD is set to true by typing "WT" and false by "WF". If true,

word is the default, if false string is the default. Affects FIND and REPLACE.

4.2 MACRO DEFINATIONS

Through SET mode, SEDIT enables the user to define the macros. When "M" is typed for M(acro, the macros will be listed on the screen and the user will be given the option to change them if he wishes.

5. MISCELLANEOUS COMMANDS

5.1 VERIFY

The VERIFY command is executed in response to typing "V". The text is verified by redisplaying the text on the screen. The Editor attempts to adjust the window so the cursor is at the center of the screen.

PROGRAM DOCUMENTATION

1. INTRODUCTION

This editor is not a copy of the UCSD Full Screen L2 editor. It is intended to have a subset of commands equal to the UCSD editor. It also works, whereas the Reverse File Paging function of the UCSD Full Screen editor does not work. The interval structure of the UCSD Full Screen editor is unknown. Any similarities between the SEDIT program and the UCSD program are strictly coincidental.

The actual program to date is a working subset of the ideal described previously. This subset includes the following commands:

movement commands (up, down, left, right direction control, page, return, jump to beginning, and jump to end); verify command.

The following modes exist:

Insert, Delete, Exchange.

An additional mode has been added and is called Fix. Fix mode will let the user remove nonprinting characters (except for line feeds and tab) from his file.

The excessive use of global variables is considered to be a poor trait of large programs. This attitude was taken very seriously in the construction of SEDIT. In the 2700 line program, there are 7 global variables. The former sections of this document list many more global variables than were ever implemented. This eliminates the necessity of

maintaining the value of these at all times, but makes it necessary to calculate the value when it is needed.

2. INPUT OUTPUT

A problem with SEDIT is the absence of appropritate IO interfaces. The Pascal routines Accept and Display are well- standardized single-character oriented IO. But the standardiza- tion is of the interface between Pascal and the operating system, not between Pascal and the user. On the 8/32 minicomputer under the MTM operating system with Robert Young's Pascal Driver the Accept and Display functions are buffered by the driver as line-oriented IO. This can be overcome with the use of SVC1 calls to the operating system with Image mode single character read and multiple character write. This method is very effective. In other operating systems Accept and Display might be adequate.

The prompts for IO to the console are disabled by the PROMPT command in the SEDIT.CSS file so that they will not interfere with the screen display.

The existing CRT driver will buffer input characters for the Pascal program only when the Pascal program is ready to Accept them. If the computer system is lightly used, SEDIT is almost always waiting for input. If SEDIT has a time-consuming task and is not asking the CRT driver for input, any characters the user types will be buffered as an MTM operating system command. Because the prompts are disabled, the user has no way of knowing when this state exists.

The desirable driver will buffer all input characters for the Pascal

program, and give the user access to the MTM operating system only when he hits break. The driver could use a circular ring buffer or double buffers to implement this feature (called the type ahead feature).

2. DATA STRUCTURES

A common problem with editor data structures is with insertion and deletion. If the user's text is in memory in an array of characters or an array of lines, the inevitable occurs: in- formation will be inserted or deleted. This requires that all of the contents of the array on one side of the operation be shifted. Pointers may be used to keep track of the string of text. Pointers require much manipulation and make a program very hard to read. In effect, they are messy. Shifting the contents of arrays periodically requires large amounts of time

Another alternative is to implement the user's file as a string of characters that is broken at the current point of action. Each half of the string is started in a stack. A string such as "eat that cat" with the current point of action at the "c" will exist with "eat that" in the left stack and "at" in the right stack. The character "space" will be at the top of the left stack and the "a" will be at the top of the right stack. The character "c" will be in the current character buffer of type CHAR. Deletions can be done by removing information from the top of either stack. Additions can be done by adding characters to the top of either stack. This method greatly increases the ease of insertion and deletion, but also increases the overhead for moving through the user's file.

One of the major good points of SEDIT is that it will work with files

of any length. The only limits are those imposed by the operating system. This is because of the data structures used to contain the user's file. An important point to realize is that the right stack must contain information in reverse order relative to the left stack.

To simplify the interface of the program with this data structure, only 4 procedure calls are used. The main program does not have access to any of the variables in these procedures other than the actual parameter CURRENT_CHAR . The procedures are:

Add(CURRENT_CHAR) which will add current character to the left stack.

Delete(CURRENT_CHAR) which will pop a character from the right stack and assign it to CURRENT_CHAR , destroying the original character.

MOVE.RIGHT(CURRENT_CHAR) which will push the CURRENT_CHAR onto the right stack and pop a character off of the left stack into CURRENT_CHAR .

MOVE.LEFT(CURRENT_CHAR) which will push the CURRENT_CHAR onto the left stack and pop a character off of the right stack into CURRENT_CHAR .

By using these 4 procedures the entire file can be examined and manipulated. The system is analogous to a Turring machine, where the CURRENT_CHAR is the position of the read head and the tape is the character string. When applied to cursor movement on the CRT screen, the current cursor position is analogous to the read head position. A

major source of confusion when thinking about this is the concept of moving in a certain direction. For example, when the string "1234" is in the data structure and the MOVE.LEFT (CURRENT_CHAR) procedure is executed, if the CURRENT_CHAR had been "3" it is now "4". But if the user has "1234" on the screeen with the current cursor position on "3" and executes a left arrow, the new CURRENT_CHAR is "2". The Move procedure implies move- ment of the character string past a certain point. The cursor movement commands imply movement of the point along the character string.

The user's file when in SEDIT is a character string delimited by the following characters: EM NL character string NL EM. When manipulating the file, the procedures in SEDIT detect the starting or ending of the file when CURRENT_CHAR = EM.

The left and right stacks exist in memory; they are arrays of characters. Their size is static. When the user fills a stack, it is half-emptied into a file. The file for the right stack is called the right file, for the left stack the left file. When a stack is emptied it is half-filled from a file. The stacks in memory called buffer stacks are actually decks, implemented as a circular ring buffer that can be popped and pushed from both ends. The emptying and filling of the buffer stacks is done 4 pages at a time by popping or pushing on the bottom of the stacks. Boolean parameters in the procedure calls terminate filling a buffer if the file is empty.

4. INITIALIZATION

The initialization of a file involves reading it into memory and

outputing it into the right file so that the pages in the right file stack are in reverse order. Next, the line numbers are added by moving the character left through the CURRENT_CHAR. When an NL character is detected, the line number characters are added. When the EM character is reached, the process is stopped, all of the file is shifted back to the right side of CURRENT_CHAR , and the screen is filled with the Verify procedure.

5. PROGRAM FILE NAMES

Line numbers are present ing the workin subset. The inhibiting of line number display has been provided for with the global Boolean variable SHOW-LINE.NUMBERS, although this has not been tested.

Line numbers are manipulated and created in a manner like Pedit. To make handling of line numbers simple, they are inserted into the runtime character string which is in the users file. The manipulation of the cursor by the user is limited to the text outside the line number field. This field exists after every character except the last one. Each field is 8 characters wide; four for the integer, one for the decimal point, two for decimal digits and one space.

The current version of SEDIT proves to be an effective means for editing a file. However, the time involved in initializing and terminating the use of a large file decreases the attractiveness to the user. On the existing minicomputer with hard disks the initialization of a 1000-line file may take 60 seconds; stepping from the beginning of the file to the end while editing may take 30 seconds. Of course, small steps are instantaneous.

The SEDIT program exists in 3 files: the PREFIX and SVC types are in SYSMAC.PAS , the data buffer management routines are in MACRO.PAS , the main program is in SEDIT1.PAS . These programs are lested in the appendix.

Class types are used for convenience; they are available with the PASCAL used in this computer science department. The classes can be omitted with the use of an editor and a few hours' time of a knowledgable programmer to gain portability to other operating systems.

APPENDIX

SEDIT Commands

Command	Operation
SEDIT [filename][.ext].....	entering SEDIT
Q.....	exiting SEDIT
<ctrl>H,<left arrow>.....	move cursor left*
<ctrl>L,<right arrow>.....	move cursor right*
<ctrl>K,<up arrow>.....	move cursor up*
<ctrl>J,<down arrow>.....	move cursor down*
<.....	change direction to backwards
>.....	change direction to forwards
<space>.....	move direction*
<back-space>.....	move against direction*
<return>.....	move to next line
A.....	adjust line in margins
C.....	copy from buffer or file
D.....	delete text
E.....	examine spelling
F.....	find a string or word*
H.....	display a line in hexidecimal*
I.....	insert text
J.....	jump to a differnt place in the text
M.....	margin adjustment of a paragraph

P..... page thru the text*

R..... replace a string with another*

S..... set format control and macro definations

V..... verify text, fill screen

X..... exchange text with what is typed

* These commands may be preceded with a repeat factor.

PROGRAM LISTINGS

Pascal Prefix, file name SYSMAC.PAS

```
1    "%MACRO FULL_PREFIX"
2
3
4    "$EJECT"
5    "PER BRINCH HANSEN                    *   AS MODIFIED FOR THE INTERDATA
6                                          *      8/32 UNDER OS/32-MT AT
7     INFORMATION SCIENCE                  *   DEPARTMENT OF COMPUTER SCIENCE
8     CALIFORNIA INSTITUTE OF TECHNOLOGY   *      KANSAS STATE UNIVERSITY
9                                          *
10    UTILITY PROGRAMS FOR                 *
11    THE SOLO SYSTEM                      *
12                                         *
13    18 MAY 1975                          *           1 DEC 1976"
14
15
16   "############
17   #  PREFIX  #
18   ############"
19
20
21   CONST NL = '(:10:)';   FF = '(:12:)';   CR = '(:13:)';   EM = '(:25:)';
22
23   CONST PAGELENGTH = 512;
24   TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF CHAR;
25
26   CONST LINELENGTH = 132;
27   TYPE LINE = ARRAY (.1..LINELENGTH.) OF CHAR;
28
29   CONST IDLENGTH = 12;
30   TYPE IDENTIFIER = ARRAY (.1..IDLENGTH.) OF CHAR;
31
32   TYPE FILE = 1..5;
33
34   TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);
35
36   TYPE FILEATTR = RECORD
37                     KIND: FILEKIND;
38                     ADDR: INTEGER;
39                     PROTECTED: BOOLEAN;
40                     NOTUSED: ARRAY (.1..5.) OF INTEGER
41                   END;
42
43   TYPE IODEVICE =
44     (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE, CARDDEVICE);
45
46   TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);
47
48   TYPE IOARG = (WRITEEOF, REWIND, UPSPACE, BACKSPACE);
```

```
49
50    TYPE IORESULT =
51      (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
52       ENDFILE, ENDMEDIUM, STARTMEDIUM);
53
54    TYPE IOPARAM = RECORD
55                     OPERATION: IOOPERATION;
56                     STATUS: IORESULT;
57                     ARG: IOARG
58                   END;
59
60    TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);
61
62    TYPE ARGTAG =
63      (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);
64
65    TYPE POINTER =  BOOLEAN;
66
67    TYPE ARGTYPE = RECORD
68                     CASE TAG: ARGTAG OF
69                       NILTYPE, BOOLTYPE: (BOOL: BOOLEAN);
70                       INTTYPE: (INT: INTEGER);
71                       IDTYPE: (ID: IDENTIFIER);
72                       PTRTYPE: (PTR: POINTER)
73                   END;
74
75    CONST MAXARG = 10;
76    TYPE ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;
77
78    TYPE ARGSEQ = (INP, OUT);
79
80    TYPE PROGRESULT =
81      (TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR, VARIANTERROR,
82       HEAPLIMIT, STACKLIMIT, CODELIMIT, TIMELIMIT, CALLERROR);
83
84    PROCEDURE READ(VAR C: CHAR);
85    PROCEDURE WRITE(C: CHAR);
86
87    PROCEDURE OPEN(F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);
88    PROCEDURE CLOSE(F: FILE);
89    PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
90    PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
91    FUNCTION LENGTH(F: FILE): INTEGER;
92
93    PROCEDURE MARK(VAR TOP: INTEGER);
94    PROCEDURE RELEASE(TOP: INTEGER);
95
96    PROCEDURE IDENTIFY(HEADER: LINE);
97    PROCEDURE ACCEPT(VAR C: CHAR);
98    PROCEDURE DISPLAY(C: CHAR);
99
100   PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE; VAR EOF: BOOLEAN);
101   PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);
102   PROCEDURE READLINE(VAR TEXT: UNIV LINE);
```

```
103    PROCEDURE WRITELINE(TEXT: UNIV LINE);
104    PROCEDURE READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
105    PROCEDURE WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
106
107    PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);
108
109    PROCEDURE IOTRANSFER
110      (DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: UNIV PAGE);
111
112    PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARAM: IOPARAM);
113
114    FUNCTION TASK: TASKKIND;
115
116    PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
117                  VAR LINE: INTEGER; VAR RESULT: PROGRESULT);
118
119    PROCEDURE RESET( LU : INTEGER);
120
121    PROCEDURE BREAKPT ( LN : INTEGER );
122
123    "%ENDMACRO FULL_PREFIX"
124
125
126    "%MACRO PREFIX"
127    CONST NL='(:10:)';  FF='(:12:)';  CR='(:13:)';  EM='(:25:)';
128    CONST PAGELENGTH=512;  LINELENGTH=132;
129    TYPE  PAGE=ARRAY [1..PAGELENGTH] OF CHAR;
130          LINE=ARRAY [1..LINELENGTH] OF CHAR;
131    PROCEDURE READ(VAR C: CHAR);
132    PROCEDURE WRITE(C: CHAR);
133
134    PROCEDURE OPEN(FILE:INTEGER; ID:UNIV LINE; VAR FOUND:BOOLEAN);
135    PROCEDURE CLOSE(FILE:INTEGER);
136    PROCEDURE GET(FILE:INTEGER; PAGE_NO:INTEGER; VAR BLOCK: UNIV PAGE);
137    PROCEDURE PUT(FILE:INTEGER; PAGE_NO:INTEGER; VAR BLOCK: UNIV PAGE);
138    FUNCTION LENGTH:INTEGER; "NOT IMPLEMENTED"
139
140    PROCEDURE MARK(VAR TOP: INTEGER);
141    PROCEDURE RELEASE(TOP: INTEGER);
142
143    PROCEDURE IDENTIFY;   "NOT IMPLEMENTED"
144    PROCEDURE ACCEPT(VAR C: CHAR);
145    PROCEDURE DISPLAY(C: CHAR);
146
147    "%ENDMACRO PREFIX"
148    "%MACRO SVC_TYPES"
149
150    "*************************************************************************"
151    "                                                                        "
152    "          OS/32MT3 SVC INTERFACE ROUTINE TYPES                          "
153    "                                                                        "
154    "*************************************************************************"
155
156
```

```
157    "MISCELLANEOUS DATA TYPES"
158
159    TYPE CHAR1 = PACKED ARRAY [1..1] OF CHAR;
160    TYPE CHAR3 = PACKED ARRAY [1..3] OF CHAR;
161    TYPE CHAR8 = PACKED ARRAY [1..8] OF CHAR;
162    TYPE CHAR4 = PACKED ARRAY [1..4] OF CHAR;
163    TYPE CHAR16 = ARRAY [1..16] OF CHAR;
164    TYPE CHAR28 = ARRAY [1..28] OF CHAR;
165
166    "SVC1 PARAMETER BLOCK"
167
168    TYPE SVC1_BLOCK = RECORD
169          SVC1_FUNC: BYTE;           "FUNCTION CODE"
170          SVC1_LU: BYTE;             "LOGICAL UNIT NUMBER"
171          SVC1_STAT: BYTE;           "DEV-INDEP STATUS"
172          SVC1_DEV_STAT: BYTE;       "DEV-DEPENDENT STATUS"
173          SVC1_BUFSTART: INTEGER;    "ADDRESS(BUFFER)"
174          SVC1_BUFEND: INTEGER;      "ADDRESS(BUFFER)+SIZE(BUFFER)-1"
175          SVC1_RANDOM_ADDR: INTEGER;  "RANDOM ADDRESS FOR DASD"
176          SVC1_XFER_LEN: INTEGER;    "TRANSFER LENGTH"
177          SVC1_RESERVED: INTEGER;    "RESERVED FOR ITAM USE"
178         END;
179
180    "SVC 1 FUNCTION CODES"
181
182    CONST SVC1_DATA_XFER = #00;       SVC1_COMMAND   = #80;
183          SVC1_READ      = #40;       SVC1_WRITE     = #20;
184          SVC1_TESTSET   = #60;       SVC1_TESTIO    = #00;
185          SVC1_ASCII     = #00;       SVC1_BINARY    = #10;
186          SVC1_PROCEED   = #00;       SVC1_WAIT      = #08;
187          SVC1_SEQL      = #00;       SVC1_RANDOM    = #04;
188          SVC1_CWAIT     = #00;       SVC1_UNC_PROC  = #02;
189          SVC1_FORMAT    = #00;       SVC1_IMAGE     = #01;
190
191          SVC1_REW       = #40;       SVC1_BSR       = #20;
192          SVC1_FSR       = #10;       SVC1_WFM       = #08;
193          SVC1_FSF       = #04;       SVC1_BSF       = #02;
194          SVC1_RESV_FN   = #01;
195
196    "SVC 1 DEVICE-INDEPENDENT STATUS CODES"
197
198    CONST SVC1_OK         = #00;       SVC1_ERROR      = #80;
199          SVC1_ILGFN      = #40;       SVC1_DU         = #20;
200          SVC1_EOM        = #10;       SVC1_EOF        = #08;
201          SVC1_UNRV       = #04;       SVC1_RECV       = #02;
202          SVC1_ILGLU      = #01;       SVC1_DEVBUSY    = #7F;
203
204
205    "FILE DESCRIPTOR FOR SVC7 REQUESTS"
206
207    TYPE FD_TYPE = PACKED RECORD
208          VOLN: CHAR4;               "VOLUME NAME"
209          FN: CHAR8;                 "FILE NAME"
210          EXTN: CHAR3;               "EXTENSION"
```

```
211          ACCT: CHAR;                "ACCOUNT NUMBER CODE"
212         END;
213
214     "SVC 7 PARAMETER BLOCK"
215
216     TYPE SVC7_BLOCK = RECORD
217          SVC7_CMD: BYTE;            "COMMAND"
218          SVC7_MOD: BYTE;            "MODIFIER/DEVICE TYPE"
219          SVC7_STAT: BYTE;           "STATUS"
220          SVC7_LU: BYTE;             "LOGICAL UNIT NUMBER"
221          SVC7_KEYS: SHORTINTEGER;   "READ/WRITE KEYS"
222          SVC7_RECLEN: SHORTINTEGER;"LOGICAL RECORD LENGTH"
223          SVC7_FD: FD_TYPE;          "FILE DESCRIPTOR"
224          SVC7_SIZE: INTEGER;        "FILE(/INDEX) SIZE"
225         END;
226
227     "SVC 7 COMMAND CODES"
228
229
230     CONST SVC7_ALLOC        = #80;      SVC7_ASSIGN     = #40;
231           SVC7_CHAP         = #20;      SVC7_RENAME     = #10;
232           SVC7_REPROT       = #08;      SVC7_CLOSE      = #04;
233           SVC7_DELETE       = #02;      SVC7_CHECKPT    = #01;
234           SVC7_FETCH_ATTR   = #00;
235
236     "SVC 7 MODIFIER CODES - ACCESS PRIVILEGES"
237
238     CONST SVC7_AP_SRO       = #00;      SVC7_AP_ERO     = #20;
239           SVC7_AP_SWO       = #40;      SVC7_AP_EWO     = #60;
240           SVC7_AP_SRW       = #80;      SVC7_AP_SREW    = #A0;
241           SVC7_AP_ERSW      = #C0;      SVC7_AP_ERW     = #E0;
242
243     "SVC 7 MODIFIER CODES - BUFFERING/FILE TYPE"
244
245     CONST SVC7_BUF_DEFAULT  = #00;      SVC7_BUF_PHYS   = #08;
246           SVC7_BUF_LOG      = #10;      SVC7_BUF_SVC15  = #18;
247           SVC7_FTYPE_CONTIG = #00;      SVC7_FTYPE_CHAIN= #01;
248           SVC7_FTYPE_INDEX  = #02;      SVC7_FTYPE_ITAM = #07;
249
250
251     "%ENDMACRO SVC_TYPES"
252
253     "%MACRO STANDARD_EXTERNS"
254     PROCEDURE BREAKPNT(LINENO:INTEGER);EXTERN;
255     PROCEDURE SVC1(VAR BLOCK:SVC1_BLOCK);EXTERN;
256     PROCEDURE SVC3(RETURN_CODE:INTEGER);EXTERN;
257     PROCEDURE SVC7(VAR BLOCK:SVC7_BLOCK);EXTERN;
258     "%ENDMACRO STANDARD_EXTERNS"
```

Class types, file name MACRO.PAS

```
1
2     "%MACRO CLASS_TYPES"
3     (************************************************************************
4      * SVC7 CALLS FOR ALLOCATE, RENAME ,AND ASSIGN.                         *
5      ************************************************************************)
6
7       ; TYPE SVC7_CALLS
8           = CLASS
9
10            ; VAR SVC7_B : SVC7_BLOCK
11
12            ; PROCEDURE SVC7 ( SVC7B : SVC7_BLOCK )
13              ; EXTERN
14
15            ; PROCEDURE ENTRY ALLOCATE_FILE
16                ( P_VOLN : CHAR4
17                ; P_FN : CHAR8
18                ; P_EXTN : CHAR3
19                ; VAR ALREADY_THERE : BOOLEAN
20                )
21
22              ; BEGIN
23                  WITH SVC7_B
24                  DO BEGIN
25                    SVC7_CMD := SVC7_ALLOC
26                  ; SVC7_MOD := SVC7_AP_SREW + SVC7_FTYPE_INDEX
27                  ; SVC7_STAT := 0
28                  ; SVC7_KEYS := 0
29                  ; SVC7_RECLEN := 512
30                  ; WITH SVC7_FD
31                    DO BEGIN
32                      VOLN := P_VOLN
33                    ; FN := P_FN
34                    ; EXTN := P_EXTN
35                    ; ACCT := 'P'
36                    END
37                  END
38                ; SVC7 ( SVC7_B )
39                ; IF SVC7_B . SVC7_STAT = 4
40                  THEN ALREADY_THERE := TRUE
41                END
42
43            ; PROCEDURE ENTRY RENAME_FILE
44                ( P_LU : BYTE ; P_FN : CHAR8 ; P_EXTN : CHAR3 )
45
46              ; BEGIN
47                  WITH SVC7_B
48                  DO BEGIN
49                    SVC7_CMD := SVC7_RENAME
50                  ; SVC7_LU := P_LU
51                  ; WITH SVC7_FD
52                    DO BEGIN
```

```
53                    FN := P_FN
54                  ; EXTN := P_EXTN
55                  ; ACCT := 'P'
56                  END
57                END
58              ; SVC7 ( SVC7_B )
59              END
60
61          ; PROCEDURE ENTRY ASSIGN_DEVICE
62              ( P_LU : BYTE ; P_VOLN : CHAR4 )
63
64            ; BEGIN
65                WITH SVC7_B
66                DO BEGIN
67                  SVC7_CMD := SVC7_ASSIGN
68                ; SVC7_LU := P_LU
69                ; WITH SVC7_FD
70                  DO BEGIN
71                    VOLN := P_VOLN
72                  ; FN := '        '
73                  ; EXTN := '   '
74                  END
75                END
76              ; SVC7 ( SVC7_B )
77              END
78
79          ; PROCEDURE ENTRY DELETE_FILE
80              ( P_VOLN : CHAR4 ; P_FN : CHAR8 ; P_EXTN : CHAR3 )
81
82            ; BEGIN
83                WITH SVC7_B
84                DO BEGIN
85                  SVC7_CMD := SVC7_DELETE
86    ;WITH SVC7_FD DO BEGIN
87                ; VOLN := P_VOLN
88                ; FN := P_FN
89                ; EXTN := P_EXTN
90    END
91                ; SVC7_KEYS := 0000
92                END
93              ; SVC7 ( SVC7_B )
94              END
95
96          ; BEGIN END
97
98    (*************************************************************************
99     * TERMINAL INPUT AND OUTPUT                                             *
100    *************************************************************************)
101
102     ; TYPE TERMINAL_IO
103         = CLASS
104
105           ; VAR SVC1_IN , SVC1_OUT : SVC1_BLOCK
106             ; C : CHAR
```

```
107               ; OUT_LINE : LINE
108               ; SVC1_OUTPUT_LENGTH : INTEGER
109
110             ; PROCEDURE SVC1 ( SVC1_OUT : SVC1_BLOCK )
111               ; EXTERN
112
113             ; PROCEDURE SET_SVC1_INPUT
114
115               ; BEGIN
116                   WITH SVC1_IN
117                   DO BEGIN
118                     SVC1_FUNC := SVC1_READ + SVC1_WAIT + SVC1_IMAGE
119                   ; SVC1_LU := 0
120                   ; SVC1_STAT := 0
121                   ; SVC1_DEV_STAT := 0
122                   ; SVC1_BUFSTART := ADDRESS ( C )
123                   ; SVC1_BUFEND := ADDRESS ( C ) + 1 - 1
124                   END
125                 END
126
127             ; PROCEDURE SET_SVC1_OUTPUT
128
129               ; BEGIN
130                   WITH SVC1_OUT
131                   DO BEGIN
132                     SVC1_FUNC := SVC1_WRITE + SVC1_WAIT + SVC1_IMAGE
133                   ; SVC1_LU := 0
134                   ; SVC1_STAT := 0
135                   ; SVC1_DEV_STAT := 0
136                   ; SVC1_BUFSTART := ADDRESS ( OUT_LINE )
137                   END
138                 ; SVC1_OUTPUT_LENGTH := 0
139                 END
140
141             ; PROCEDURE SHIP_OUT_
142
143               ; BEGIN
144                   SVC1_OUT . SVC1_BUFEND
145                   := SVC1_OUT . SVC1_BUFSTART + SVC1_OUTPUT_LENGTH - 1
146                 ; SVC1 ( SVC1_OUT )
147                 ; SVC1_OUTPUT_LENGTH := 0
148                 END
149
150             ; PROCEDURE ENTRY SHIP_OUT
151
152               ; BEGIN IF SVC1_OUTPUT_LENGTH <> 0 THEN SHIP_OUT_ END
153
154             ; PROCEDURE ENTRY INPUT ( VAR CURRENT_CHAR : CHAR )
155
156               ; BEGIN
157                   SVC1 ( SVC1_IN )
158                 ; CURRENT_CHAR := CHR ( ORD ( C ) MOD 128 )
159                 END
160
```

```
161           ; PROCEDURE OUTPUT_ ( CC : CHAR )
162
163             ; BEGIN
164                 SVC1_OUTPUT_LENGTH := SUCC ( SVC1_OUTPUT_LENGTH )
165               ; OUT_LINE [ SVC1_OUTPUT_LENGTH ] := CC
166               ; IF SVC1_OUTPUT_LENGTH = 79
167                 THEN SHIP_OUT_
168               END
169
170           ; PROCEDURE ENTRY OUTPUT ( CC : CHAR )
171
172             ; BEGIN OUTPUT_ ( CC ) END
173
174           ; PROCEDURE ENTRY OUTPUT_LINE ( ALINE : LINE )
175
176             ; VAR I : INTEGER
177
178             ; BEGIN
179                 I := 1
180               ; WHILE ALINE [ I ] <> '$'
181                 DO BEGIN
182                   OUTPUT_ ( ALINE [ I ] )
183                 ; I := I + 1
184                 END
185               ; SHIP_OUT_
186               END
187
188           ; BEGIN  (* INIT STATEMENT OF TERMINAL IO *)
189               SET_SVC1_OUTPUT
190             ; SET_SVC1_INPUT
191             END
192
193     (*************************************************************************
194      * BUFFER STACK CLASS, IN MEMORY BUFFERS FOR TEXT DATA                   *
195      *************************************************************************)
196
197       ; TYPE BUFFER_STACK
198           = CLASS
199
200             ; VAR BUFFER_ARRAY : ARRAY [ 1 .. BUFFER_SIZE ] OF CHAR
201               ; HEAD , TAIL , LENGTH , I : INTEGER
202
203             ; PROCEDURE ENTRY PUSH ( C : CHAR ; VAR FULL : BOOLEAN )
204
205               ; BEGIN
206                   BUFFER_ARRAY [ HEAD ] := C
207                 ; LENGTH := LENGTH + 1
208                 ; FULL := ( LENGTH = BUFFER_SIZE )
209                 ; HEAD := ( HEAD MOD BUFFER_SIZE ) + 1
210                 END
211
212             ; PROCEDURE ENTRY POP ( VAR C : CHAR ; VAR EMPTY : BOOLEAN )
213
214               ; BEGIN
```

```
215                 HEAD := ( HEAD + BUFFER_SIZE - 2 ) MOD BUFFER_SIZE + 1
216               ; LENGTH := LENGTH - 1
217               ; EMPTY := ( LENGTH = 0 )
218               ; C := BUFFER_ARRAY [ HEAD ]
219               END
220
221           ; PROCEDURE PUSH_ON_BOTTOM ( C : CHAR )
222
223             ; BEGIN
224                 BUFFER_ARRAY [ TAIL ] := C
225               ; LENGTH := LENGTH + 1
226               ; TAIL := ( TAIL + BUFFER_SIZE - 2 ) MOD BUFFER_SIZE + 1
227               END
228
229           ; PROCEDURE POP_OFF_BOTTOM
230               ( VAR C : CHAR ; VAR EMPTY : BOOLEAN )
231
232             ; BEGIN
233                 TAIL := ( TAIL MOD BUFFER_SIZE ) + 1
234               ; LENGTH := LENGTH - 1
235               ; C := BUFFER_ARRAY [ TAIL ]
236               ; EMPTY := ( LENGTH = 0 )
237               END
238
239           ; PROCEDURE ENTRY PUSH_PAGE ( P : PAGE ; REVERSE : BOOLEAN )
240
241             ; BEGIN
242                 IF REVERSE
243                 THEN FOR I := 1 TO 512 DO PUSH_ON_BOTTOM ( P [ I ] )
244                 ELSE FOR I := 512 DOWNTO 1 DO PUSH_ON_BOTTOM ( P [ I ] )
245               END
246
247           ; PROCEDURE ENTRY POP_PAGE
248               ( VAR P : PAGE ; REVERSE : BOOLEAN ; VAR EMPTY : BOOLEAN )
249
250             ; BEGIN
251                 IF REVERSE
252                 THEN FOR I := 512 DOWNTO 1
253                   DO POP_OFF_BOTTOM ( P [ I ] , EMPTY )
254                 ELSE BEGIN
255                   I := 0
256                 ; REPEAT I := I + 1
257                   ; POP_OFF_BOTTOM ( P [ I ] , EMPTY )
258                   UNTIL ( I = 512 ) OR EMPTY
259                 END
260               END
261
262           ; BEGIN TAIL := BUFFER_SIZE ; HEAD := 1 ; LENGTH := 0 END
263
264       (*************************************************************
265        * FILE STACK CLASS,  ON DISK FILE SPACE                     *
266        *************************************************************)
267
268         ; TYPE FILE_STACK
```

```
269         = CLASS ( FILE_NUMBER : FILE )
270
271           ; VAR PAGE_NUMBER : INTEGER
272
273           ; PROCEDURE ENTRY PUSH ( VAR PAGE_IN : PAGE )
274
275             ; BEGIN
276                 PUT ( FILE_NUMBER , PAGE_NUMBER , PAGE_IN )
277               ; PAGE_NUMBER := PAGE_NUMBER + 1
278               END
279
280           ; PROCEDURE ENTRY POP
281               ( VAR PAGE_OUT : PAGE ; VAR FILE_EMPTY : BOOLEAN )
282
283             ; BEGIN
284                 PAGE_NUMBER := PAGE_NUMBER - 1
285               ; GET ( FILE_NUMBER , PAGE_NUMBER , PAGE_OUT )
286               ; FILE_EMPTY := ( PAGE_NUMBER = 1 )
287               END
288
289           ; BEGIN PAGE_NUMBER := 1 END
290
291   (*************************************************************************
292    * LEFT AND RIGHT BUFFER STACK MANAGMENT                                 *
293    *************************************************************************)
294
295     ; TYPE BUFFER_MANAGEMENT
296         = CLASS ( IO : TERMINAL_IO )
297
298   (*
299    THE BUFFER_MANAGMENT CLASS IS GIVEN ACCESS TO THE
300    TERMINAL IO CLASS TO THAT IT CAN PUT DOTS AND COLINS
301    ON THE CONSOL SO THE USER KNOW THAT SOMETHING IS
302    HAPPENING.
303   *)
304
305           ; VAR LEFT , RIGHT : BUFFER_STACK
306             ; LF , RF : FILE_STACK
307             ; I , J : INTEGER
308             ; P : PAGE
309             ; RIGHT_EMPTY , RIGHT_FULL , LEFT_EMPTY , LEFT_FULL
310               : BOOLEAN
311
312           ; PROCEDURE ADD_LEFT ( C : CHAR )
313
314             ; VAR NOTUSED : BOOLEAN
315
316             ; BEGIN
317                 LEFT . PUSH ( C , LEFT_FULL )
318               ; IF LEFT_FULL
319                 THEN FOR I := 1 TO NO_PAGES_IN_BUFFER DIV 2
320                   DO BEGIN
321                     LEFT . POP_PAGE ( P , FALSE , NOTUSED )
322                   ; LF . PUSH ( P )
```

```
323                 END
324               END
325
326           ; PROCEDURE POP_RIGHT ( VAR CURRENT_CHAR : CHAR )
327
328             ; VAR EMPTY : BOOLEAN
329
330             ; BEGIN
331               ; RIGHT . POP ( CURRENT_CHAR , RIGHT_EMPTY )
332               ; IF RIGHT_EMPTY AND ( CURRENT_CHAR <> EM )
333                 THEN BEGIN
334                   I := 0
335                 ; REPEAT I := I + 1
336                   ; RF . POP ( P , EMPTY )
337                   ; RIGHT . PUSH_PAGE ( P , TRUE )
338                   UNTIL ( I = NO_PAGES_IN_BUFFER DIV 2 ) OR EMPTY
339                 END
340               END
341
342           ; PROCEDURE ENTRY MOVE_LEFT ( VAR CURRENT_CHAR : CHAR )
343
344             ; BEGIN
345                 ADD_LEFT ( CURRENT_CHAR )
346               ; POP_RIGHT ( CURRENT_CHAR )
347               END
348
349           ; PROCEDURE ADD_RIGHT ( C : CHAR )
350
351             ; VAR NOTUSED : BOOLEAN
352
353             ; BEGIN
354               ; RIGHT . PUSH ( C , RIGHT_FULL )
355               ; IF RIGHT_FULL
356                 THEN FOR I := 1 TO NO_PAGES_IN_BUFFER DIV 2
357                   DO BEGIN
358                     RIGHT . POP_PAGE ( P , TRUE , NOTUSED )
359                   ; RF . PUSH ( P )
360                   END
361               END
362
363           ; PROCEDURE POP_LEFT ( VAR CURRENT_CHAR : CHAR )
364
365             ; VAR EMPTY : BOOLEAN
366
367             ; BEGIN
368               ; LEFT . POP ( CURRENT_CHAR , LEFT_EMPTY )
369               ; IF LEFT_EMPTY AND ( CURRENT_CHAR <> EM )
370                 THEN BEGIN
371                   I := 0
372                 ; REPEAT I := I + 1
373                   ; LF . POP ( P , EMPTY )
374                   ; LEFT . PUSH_PAGE ( P , FALSE )
375                   UNTIL ( I = NO_PAGES_IN_BUFFER DIV 2 ) OR EMPTY
376                 END
```

```
377               END
378
379           ; PROCEDURE ENTRY MOVE_RIGHT ( VAR CURRENT_CHAR : CHAR )
380
381             ; BEGIN
382                 ADD_RIGHT ( CURRENT_CHAR )
383               ; POP_LEFT ( CURRENT_CHAR )
384               END
385
386           ; PROCEDURE ENTRY ADD ( C : CHAR )
387
388             ; BEGIN ADD_LEFT ( C ) END
389
390           ; PROCEDURE ENTRY DELETE ( VAR CURRENT_CHAR : CHAR )
391
392             ; BEGIN POP_RIGHT ( CURRENT_CHAR ) END
393
394           ; PROCEDURE ENTRY FIND_NO_OF_PAGES
395               ( FILE_NO : INTEGER ; VAR NO_OF_PAGES : INTEGER )
396
397             ; VAR END_OF_FILE : BOOLEAN
398               ; COUNT_OF_DOTS : INTEGER
399
400             ; BEGIN
401                 NO_OF_PAGES := 0
402               ; COUNT_OF_DOTS := 12
403               ; END_OF_FILE := FALSE
404               ; WHILE NOT ( END_OF_FILE )
405                 DO BEGIN
406                   NO_OF_PAGES := NO_OF_PAGES + 1
407                 ; GET ( FILE_NO , NO_OF_PAGES , P )
408                 ; IO . OUTPUT ( '.' )
409                 ; IO . SHIP_OUT
410                 ; COUNT_OF_DOTS := COUNT_OF_DOTS + 1
411                 ; IF COUNT_OF_DOTS > 80
412                   THEN BEGIN
413                     COUNT_OF_DOTS := 1
414                   ; IO . OUTPUT ( NL )
415                   ; IO . OUTPUT ( CR )
416                   ; IO . SHIP_OUT
417                   END
418               ; FOR I := 1 TO PAGELENGTH
419                   DO BEGIN
420                     IF ( P [ I ] = EM )
421                     THEN END_OF_FILE := TRUE
422                   END
423                 END
424               END
425
426           ; PROCEDURE ENTRY INIT_RIGHT_STACK
427               ( FILE_NO , LAST_PAGE_NO : INTEGER )
428
429             ; VAR COUNT_OF_COLINS : INTEGER
430
```

```
431             ; BEGIN
432                 COUNT_OF_COLINS := 1
433               ; IO . OUTPUT ( NL )
434               ; IO . OUTPUT ( CR )
435               ; IO . SHIP_OUT
436               ; FOR I := LAST_PAGE_NO DOWNTO 1
437                 DO BEGIN
438                   GET ( FILE_NO , I , P )
439                 ; COUNT_OF_COLINS := COUNT_OF_COLINS + 1
440                 ; IO . OUTPUT ( ':' )
441                 ; IO . SHIP_OUT
442                 ; IF COUNT_OF_COLINS > 80
443                   THEN BEGIN
444                     IO . OUTPUT ( NL )
445                   ; IO . OUTPUT ( CR )
446                   ; IO . SHIP_OUT
447                   ; COUNT_OF_COLINS := 1
448                   END
449                 ; RF . PUSH ( P )
450                 END
451               END
452
453           ; PROCEDURE ENTRY FILL_LEFT_STACK
454
455             ; VAR EMPTY : BOOLEAN
456
457             ; BEGIN
458                 REPEAT
459                   LEFT . POP_PAGE ( P , FALSE , EMPTY )
460                 ; LF . PUSH ( P )
461                 UNTIL EMPTY
462               END
463
464           ; BEGIN
465               INIT LEFT , RIGHT , LF ( 3 ) , RF ( 2 )
466             ; ADD_RIGHT ( NL )
467             END
468
469   (*************************************************************************
470    *   COPY BUFFER STACK MANAGMENT                                         *
471    *************************************************************************)
472
473     ; TYPE COPY_BUFFER_MANAGEMENT
474         = CLASS
475
476           ; VAR COPYB : BUFFER_STACK
477             ; COPYF : FILE_STACK
478             ; I : INTEGER
479             ; EMPTY , BUFFER_EMPTY , BUFFER_FULL , NOTUSED : BOOLEAN
480             ; P : PAGE
481
482           ; PROCEDURE ENTRY COPY_PUSH ( C : CHAR )
483
484             ; BEGIN
```

```
485                 COPYB . PUSH ( C , BUFFER_FULL )
486               ; IF BUFFER_FULL
487                 THEN FOR I := 1 TO NO_PAGES_IN_BUFFER DIV 2
488                   DO BEGIN
489                     COPYB . POP_PAGE ( P , FALSE , NOTUSED )
490                   ; COPYF . PUSH ( P )
491                   END
492               END
493
494           ; PROCEDURE ENTRY COPY_POP ( VAR C : CHAR )
495
496             ; BEGIN
497                 COPYB . POP ( C , BUFFER_EMPTY )
498               ; IF BUFFER_EMPTY AND ( C <> EM )
499                 THEN BEGIN
500                   I := 0
501                 ; REPEAT I := I + 1
502                   ; COPYF . POP ( P , EMPTY )
503                   ; COPYB . PUSH_PAGE ( P , FALSE )
504                   UNTIL ( I = NO_PAGES_IN_BUFFER DIV 2 ) OR EMPTY
505                 END
506               END
507
508           ; BEGIN
509               INIT COPYB , COPYF ( 4 )
510             ; COPYB . PUSH ( EM , NOTUSED )
511             ; COPYB . PUSH ( EM , NOTUSED )
512             END
513     "%ENDMACRO CLASS_TYPES"
```

Sedit main program, file name SEDIT1.PAS

```
1
2     (*#####################################################################
3      *                                                                   *
4      *      SSSS      EEEEEEEE    DDDDDDD      IIIIII     TTTTTTTT     *
5      *     SSSSSS     EEEEEEEE    DDDDDDDD     IIIIII     TTTTTTTT     *
6      *    SSS   SS    EE          DD    DD       II       T  TT  T     *
7      *    SSS         EE          DD    DD       II          TT        *
8      *     SSSSS      EEEEEE      DD    DD       II          TT        *
9      *        SSS     EEEEEE      DD    DD       II          TT        *
10     *     SS  SSS    EE          DD    DD       II          TT        *
11     *    SS   SSS    EE          DD    DD       II          TT        *
12     *    SSSSSSSS    EEEEEEEE    DDDDDDDD     IIIIII      TT        *
13     *     SSSSSS     EEEEEEEE    DDDDDDD      IIIIII     TTTT       *
14     *                                                                   *
15     *                                                                   *
16     *                                                                   *
17     *                                                                   *
18     * SEDIT VERSION 1                                                   *
19     * MADE BY THEODORE JOHN SOCOLOFSKY                                  *
20     * AS A PROJECT FOR CIS                                              *
21     * IN THE SUMMER OF 1981                                             *
22     #####################################################################*)
23    "%INCLUDE SVC_TYPES"
24    (*********************************************************************)
25    (*                                                                   *)
26    (*         OS/32MT3 SVC INTERFACE ROUTINE TYPES                      *)
27    (*                                                                   *)
28    (*********************************************************************)
29    "%INCLUDE FULL_PREFIX"
30    (*#########
31     # PREFIX #
32     #########*)
33      PROGRAM SEDIT
34        ( PARAM : LINE ) (*************************************)
35
36      ; TYPE FILE = 1 .. 5
37
38      ; CONST ETX = '(:3:)'
39        ; BEL = '(:7:)'
40        ; BS = '(:8:)'
41        ; HT = '(:9:)'
42        ; TAB = '(:9:)'
43        ; VT = '(:11:)'
44        ; CR = '(:13:)'
45        ; SUB = '(:26:)'
46        ; ESC = '(:27:)'
47        ; RS = '(:30:)'
48        ; DEL = '(:127:)'
49        ; PAD = '(:127:)'
50        ; SET_OF_DIGITS = [ '0' .. '9' ]
51        ; SET_OF_UPPERCASE_LETTERS = [ 'A' .. 'Z' ]
52        ; SET_OF_LOWERCASE_LETTERS = [ 'a' .. 'z' ]
```

```
53        ; SET_OF_PRINTING_SPECIAL_CHARACTERS
54          = [ ' ' .. '/' , ':' .. '@' , '[' .. '`' ]
55        ; SET_OF_VALID_INPUT_CHARACTERS
56          = SET_OF_DIGITS
57            + SET_OF_UPPERCASE_LETTERS
58            + SET_OF_LOWERCASE_LETTERS
59            + SET_OF_PRINTING_SPECIAL_CHARACTERS
60        ; SET_OF_INVALID_INPUT_CHARACTERS
61          = [ '(:0:)' .. '(:31:)' , '(:127:)' ]
62        ; NO_PAGES_IN_BUFFER = 8
63        ; BUFFER_SIZE = NO_PAGES_IN_BUFFER * PAGELENGTH
64
65    "%INCLUDE CLASS_TYPES"
66    (* MAIN ******************************************************)
67
68      ; VAR IO : TERMINAL_IO
69        ; B_M : BUFFER_MANAGEMENT
70        ; S7 : SVC7_CALLS
71        ; C_B_M : COPY_BUFFER_MANAGEMENT
72        ; SHOW_LINE_NUMBERS , SECOND_EDIT_PROMPT_ON : BOOLEAN
73        ; CURSOR_ROW , CURSOR_COL : INTEGER
74        ; SAVED_CURSOR_ROW , SAVED_CURSOR_COL : INTEGER
75        ; DIRECTION_CHAR : CHAR
76
77      ; PROCEDURE CLEAR_SCREEN
78
79        ; BEGIN
80            IO . OUTPUT ( SUB )
81          ; IO . OUTPUT ( PAD )
82          ; IO . OUTPUT ( PAD )
83          ; IO . SHIP_OUT
84          ; CURSOR_ROW := 1
85          ; CURSOR_COL := 1
86          END
87
88      ; PROCEDURE HOME_CURSOR
89
90        ; BEGIN
91            IO . OUTPUT ( RS )
92          ; IO . OUTPUT ( PAD )
93          ; IO . OUTPUT ( PAD )
94          ; IO . SHIP_OUT
95          ; CURSOR_ROW := 1
96          ; CURSOR_COL := 1
97          END
98
99      ; PROCEDURE MOVE_CURSOR ( ROW , COL : INTEGER )
100
101       ; VAR TROW , TCOL : INTEGER
102
103       ; BEGIN
104           TROW := ROW
105         ; TCOL := COL
106         ; IO . OUTPUT ( ESC )
```

```
107           ; IO . OUTPUT ( '=' )
108           ; IF ROW < 1
109             THEN TROW := 1
110           ; IF ROW > 24
111             THEN TROW := 24
112           ; IF COL < 1
113             THEN TCOL := 1
114           ; IF COL > 80
115             THEN TCOL := 80
116           ; IO . OUTPUT ( CHR ( TROW + 31 ) )
117           ; IO . OUTPUT ( CHR ( TCOL + 31 ) )
118           ; CURSOR_ROW := TROW
119           ; CURSOR_COL := TCOL
120           ; IO . SHIP_OUT
121           END
122
123       ; PROCEDURE SAVE_CURSOR_POSITION
124
125         ; BEGIN
126             SAVED_CURSOR_ROW := CURSOR_ROW
127           ; SAVED_CURSOR_COL := CURSOR_COL
128           END
129
130       ; PROCEDURE UNSAVE_CURSOR_POSITION
131
132         ; BEGIN
133             CURSOR_ROW := SAVED_CURSOR_ROW
134           ; CURSOR_COL := SAVED_CURSOR_COL
135           END
136
137       ; PROCEDURE MOVE_TO_END_OF_CURRENT_LINE
138           ( VAR CURRENT_CHAR : CHAR ; VAR LENGTH : INTEGER )
139
140         ; BEGIN
141             LENGTH := 0
142           ; WHILE CURRENT_CHAR <> NL
143             DO BEGIN
144               B_M . MOVE_LEFT ( CURRENT_CHAR )
145             ; LENGTH := LENGTH + 1
146             END
147           END
148
149       ; PROCEDURE MOVE_TO_END_OF_PREVIOUS_LINE
150           ( VAR CURRENT_CHAR : CHAR ; VAR LENGTH : INTEGER )
151
152         ; BEGIN
153             LENGTH := 0
154           ; WHILE CURRENT_CHAR <> NL
155             DO BEGIN
156               B_M . MOVE_RIGHT ( CURRENT_CHAR )
157             ; LENGTH := LENGTH + 1
158             END
159           END
160
```

```
161       ; PROCEDURE OUTPUT_REST_OF_LINE
162           ( VAR CURRENT_CHAR : CHAR ; VAR LENGTH : INTEGER )
163
164         ; VAR I , J : INTEGER
165
166         ; BEGIN
167             IF NOT SHOW_LINE_NUMBERS
168             THEN BEGIN
169               MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , I )
170             ; IF I > 9
171               THEN LENGTH := I
172               ELSE LENGTH := 9
173             ; J := 0
174             ; WHILE ( J < LENGTH ) AND ( CURRENT_CHAR <> EM )
175               DO BEGIN
176                 J := J + 1
177               ; B_M . MOVE_LEFT ( CURRENT_CHAR )
178               END
179             ; IF CURRENT_CHAR = EM
180               THEN B_M . MOVE_RIGHT ( CURRENT_CHAR )
181             END
182           ; LENGTH := 0
183           ; WHILE CURRENT_CHAR <> NL
184             DO BEGIN
185               IF CURSOR_COL + LENGTH <= 80
186               THEN IO . OUTPUT ( CURRENT_CHAR )
187             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
188             ; LENGTH := LENGTH + 1
189             END
190           ; IF CURSOR_COL + LENGTH <= 80
191             THEN CURSOR_COL := CURSOR_COL + LENGTH
192             ELSE BEGIN
193               CURSOR_COL := 80
194             ; IF CURSOR_COL + LENGTH > 80
195               THEN BEGIN
196                 MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
197               ; IO . OUTPUT ( '!' )
198               END
199             END
200           ; IF LENGTH <> 0
201             THEN IO . SHIP_OUT
202           END
203
204       ; PROCEDURE MOVE_TO_BEGINNING_OF_TEXT ( VAR CURRENT_CHAR : CHAR )
205
206         ; VAR I : INTEGER
207
208         ; BEGIN
209             REPEAT
210               B_M . MOVE_RIGHT ( CURRENT_CHAR )
211             UNTIL CURRENT_CHAR = EM
212           ; FOR I := 1 TO 10 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
213           END
214
```

```
215       ; PROCEDURE MOVE_TO_END_OF_TEXT ( VAR CURRENT_CHAR : CHAR )
216
217         ; BEGIN
218             REPEAT
219               B_M . MOVE_LEFT ( CURRENT_CHAR )
220             UNTIL CURRENT_CHAR = EM
221           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
222           END
223
224       ; PROCEDURE OUTPUT_REST_OF_SCREEN
225           ( VAR CURRENT_CHAR : CHAR
226           ; VAR LENGTH , NUMBER_OF_LINES : INTEGER
227           )
228
229         ; VAR TLENGTH : INTEGER
230
231         ; BEGIN
232             LENGTH := 0
233           ; NUMBER_OF_LINES := 0
234           ; WHILE ( CURRENT_CHAR <> EM ) AND ( CURSOR_ROW <= 24 )
235             DO BEGIN
236               MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
237             ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , TLENGTH )
238             ; LENGTH := LENGTH + TLENGTH + 1
239             ; NUMBER_OF_LINES := NUMBER_OF_LINES + 1
240             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
241             ; CURSOR_COL := 1
242             ; CURSOR_ROW := CURSOR_ROW + 1
243             END
244           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
245           ; LENGTH := LENGTH - 1
246           END
247
248       ; PROCEDURE VERIFY ( VAR CURRENT_CHAR : CHAR )
249
250         ; VAR I , LENGTH , TLENGTH , NUMBER_OF_LINES : INTEGER
251
252         ; BEGIN
253             NUMBER_OF_LINES := 0
254           ; LENGTH := 0
255           ; REPEAT MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , TLENGTH )
256             ; LENGTH := LENGTH - TLENGTH - 1
257             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
258             ; NUMBER_OF_LINES := NUMBER_OF_LINES + 1
259             UNTIL ( NUMBER_OF_LINES = 12 ) OR ( CURRENT_CHAR = EM )
260           ; NUMBER_OF_LINES := 1
261           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
262           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
263           ; LENGTH := LENGTH + 2
264           ; REPEAT MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , TLENGTH )
265             ; LENGTH := LENGTH + TLENGTH + 1
266             ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
267             ; NUMBER_OF_LINES := NUMBER_OF_LINES + 1
268             UNTIL ( NUMBER_OF_LINES = 24 ) OR ( CURRENT_CHAR = EM )
```

```
269           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
270           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
271           ; LENGTH := LENGTH - 2
272           ; CLEAR_SCREEN
273           ; MOVE_CURSOR ( 2 , 1 )
274           ; OUTPUT_REST_OF_SCREEN
275               ( CURRENT_CHAR , TLENGTH , NUMBER_OF_LINES )
276           ; FOR I := 1 TO TLENGTH - LENGTH
277             DO BEGIN
278               B_M . MOVE_RIGHT ( CURRENT_CHAR )
279             ; IF CURRENT_CHAR = NL
280               THEN NUMBER_OF_LINES := NUMBER_OF_LINES - 1
281             END
282           ; LENGTH := 0
283           ; IF CURRENT_CHAR = NL
284             THEN BEGIN B_M . MOVE_RIGHT ( CURRENT_CHAR ) ; LENGTH := 1 END
285           ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , TLENGTH )
286           ; LENGTH := LENGTH + TLENGTH
287           ; FOR I := 1 TO LENGTH DO B_M . MOVE_LEFT ( CURRENT_CHAR )
288           ; IF LENGTH > 80
289             THEN LENGTH := 80
290           ; NUMBER_OF_LINES := NUMBER_OF_LINES + 1
291           ; MOVE_CURSOR ( NUMBER_OF_LINES , LENGTH )
292           END
293
294       ; PROCEDURE MOVE_RIGHT_1_LINE
295           ( VAR CURRENT_CHAR : CHAR ; VAR J : INTEGER )
296
297         ; VAR L : INTEGER
298           ; END_WAS_FOUND : BOOLEAN
299
300         ; BEGIN
301             L := 0
302           ; IF CURRENT_CHAR = NL
303             THEN BEGIN
304               B_M . MOVE_RIGHT ( CURRENT_CHAR )
305             ; L := 1
306             ; IF CURRENT_CHAR = EM
307               THEN BEGIN B_M . MOVE_LEFT ( CURRENT_CHAR ) ; L := 0 END
308             END
309           ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
310           ; L := L + J
311           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
312           ; IF CURRENT_CHAR = EM
313             THEN BEGIN
314               B_M . MOVE_LEFT ( CURRENT_CHAR )
315             ; END_WAS_FOUND := TRUE
316             END
317             ELSE END_WAS_FOUND := FALSE
318           ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
319           ; J := 1
320           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
321           ; WHILE ( J < L ) AND ( CURRENT_CHAR <> NL )
322             DO BEGIN
```

```
323             J := J + 1
324           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
325             END
326         ; IF END_WAS_FOUND
327           THEN J := 0
328         END
329
330     ; PROCEDURE MOVE_LEFT_1_LINE
331         ( VAR CURRENT_CHAR : CHAR ; VAR I : INTEGER )
332
333       ; VAR L , J : INTEGER
334
335       ; BEGIN
336           L := 0
337         ; IF CURRENT_CHAR = NL
338           THEN BEGIN
339             B_M . MOVE_RIGHT ( CURRENT_CHAR )
340           ; L := 1
341           ; IF CURRENT_CHAR = EM
342             THEN BEGIN B_M . MOVE_LEFT ( CURRENT_CHAR ) ; L := 0 END
343           END
344         ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , I )
345         ; L := L + I
346         ; B_M . MOVE_LEFT ( CURRENT_CHAR )
347         ; MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , I )
348         ; B_M . MOVE_LEFT ( CURRENT_CHAR )
349         ; I := 1
350         ; IF CURRENT_CHAR = EM
351           THEN BEGIN
352             B_M . MOVE_RIGHT ( CURRENT_CHAR )
353           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
354           ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , I )
355           ; FOR J := 1 TO L DO B_M . MOVE_LEFT ( CURRENT_CHAR )
356           ; I := 0
357           END
358           ELSE WHILE ( CURRENT_CHAR <> NL ) AND ( I < L )
359             DO BEGIN
360               I := I + 1
361             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
362             END
363         END
364
365     ; PROCEDURE EDIT_PROMPT ( DIRECTION_CHAR : CHAR )
366
367       ; VAR SROW , SCOL : INTEGER
368
369       ; BEGIN
370           SROW := CURSOR_ROW
371         ; SCOL := CURSOR_COL
372         ; MOVE_CURSOR ( 1 , 1 )
373         ; IO . OUTPUT ( DIRECTION_CHAR )
374         ; IO . OUTPUT_LINE
375             ( 'SEDIT: I(nsert D(elete Q(uit P(age J(ump$' )
376         ; IO . OUTPUT_LINE
```

```
377             ( ' X(change F(ix V(erify                ? $' )
378           ; MOVE_CURSOR ( SROW , SCOL )
379           END
380
381       ; PROCEDURE UP_ARROW ( VAR CURRENT_CHAR : CHAR ; VAR TOF : BOOLEAN )
382
383         ; VAR I : INTEGER
384
385         ; BEGIN
386             CURSOR_ROW := CURSOR_ROW - 1
387           ; MOVE_RIGHT_1_LINE ( CURRENT_CHAR , I )
388           ; IF CURSOR_ROW = 1
389             THEN BEGIN
390               IF I <> 0
391               THEN BEGIN
392                 CURSOR_COL := I
393               ; VERIFY ( CURRENT_CHAR )
394               ; EDIT_PROMPT ( DIRECTION_CHAR )
395               END
396               ELSE MOVE_CURSOR ( 2 , CURSOR_COL )
397             END
398             ELSE IF I <> 0
399               THEN MOVE_CURSOR ( CURSOR_ROW , I )
400               ELSE MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
401           ; IF I = 0
402             THEN TOF := TRUE
403             ELSE TOF := FALSE
404           END
405
406       ; PROCEDURE DOWN_ARROW
407           ( VAR CURRENT_CHAR : CHAR ; VAR BOF : BOOLEAN )
408
409         ; VAR I , J : INTEGER
410
411         ; BEGIN
412             MOVE_LEFT_1_LINE ( CURRENT_CHAR , J )
413           ; IF J <> 0
414             THEN BEGIN
415               CURSOR_ROW := CURSOR_ROW + 1
416             ; IF CURSOR_ROW = 25
417               THEN BEGIN
418                 FOR I := 1 TO J - 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
419               ; IO . OUTPUT ( NL )
420               ; MOVE_CURSOR ( 24 , 1 )
421               ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , I )
422               ; EDIT_PROMPT ( DIRECTION_CHAR )
423               ; FOR I := 1 TO I - ( J - 1 )
424                 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
425               ; CURSOR_ROW := 24
426               END
427             ; MOVE_CURSOR ( CURSOR_ROW , J )
428             END
429           ; IF J = 0
430             THEN BOF := TRUE
```

```
431           ELSE BOF := FALSE
432         END
433
434       ; PROCEDURE LEFT_ARROW
435           ( VAR CURRENT_CHAR : CHAR ; VAR TOF : BOOLEAN )
436
437         ; VAR J , I : INTEGER
438
439         ; BEGIN
440             TOF := FALSE
441           ; IF CURSOR_COL = 80
442             THEN BEGIN
443             ; IF CURRENT_CHAR = NL
444               THEN BEGIN B_M . MOVE_RIGHT ( CURRENT_CHAR ) ; I := 1 END
445               ELSE I := 0
446             ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
447             ; J := J + I
448             ; FOR I := 1 TO J DO B_M . MOVE_LEFT ( CURRENT_CHAR )
449             ; IF ( J > 88 ) OR ( ( J > 80 ) AND SHOW_LINE_NUMBERS )
450               THEN CURSOR_COL := CURSOR_COL + 1
451             END
452           ; CURSOR_COL := CURSOR_COL - 1
453           ; IF ( CURSOR_COL < 1 )
454                OR ( SHOW_LINE_NUMBERS AND ( CURSOR_COL < 9 ) )
455             THEN BEGIN
456               IF CURRENT_CHAR = NL
457               THEN BEGIN B_M . MOVE_RIGHT ( CURRENT_CHAR ) ; I := 1 END
458               ELSE I := 0
459             ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
460             ; J := J + I
461             ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
462             ; IF CURRENT_CHAR <> EM
463               THEN BEGIN
464                 MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
465               ; B_M . MOVE_LEFT ( CURRENT_CHAR )
466               ; MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
467               ; CURSOR_ROW := CURSOR_ROW - 1
468               END
469               ELSE BEGIN
470                 FOR I := 1 TO J + 1 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
471               ; IF SHOW_LINE_NUMBERS
472                 THEN J := 8
473                 ELSE J := 0
474               ; TOF := TRUE
475               END
476             ; IF SHOW_LINE_NUMBERS
477               THEN CURSOR_COL := J + 1
478               ELSE CURSOR_COL := J - 7
479             END
480             ELSE B_M . MOVE_RIGHT ( CURRENT_CHAR )
481           ; IF CURSOR_COL > 80
482             THEN CURSOR_COL := 80
483           ; IF CURSOR_ROW = 1
484             THEN BEGIN
```

```
485             VERIFY ( CURRENT_CHAR )
486           ; EDIT_PROMPT ( DIRECTION_CHAR )
487           END
488           ELSE MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
489         END
490
491       ; PROCEDURE RIGHT_ARROW
492           ( VAR CURRENT_CHAR : CHAR ; VAR BOF : BOOLEAN )
493
494         ; VAR I , J : INTEGER
495
496         ; BEGIN
497             BOF := FALSE
498           ; CURSOR_COL := CURSOR_COL + 1
499           ; IF CURRENT_CHAR = NL
500             THEN BEGIN
501               IF SHOW_LINE_NUMBERS
502               THEN BEGIN
503                 B_M . MOVE_LEFT ( CURRENT_CHAR )
504               ; IF CURRENT_CHAR <> EM
505                 THEN BEGIN
506                   CURSOR_ROW := CURSOR_ROW + 1
507                 ; CURSOR_COL := 9
508                 ; IF CURSOR_ROW = 25
509                   THEN BEGIN
510                     IO . OUTPUT ( NL )
511                   ; MOVE_CURSOR ( 24 , 1 )
512                   ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , I )
513                   ; FOR J := 1 TO I - 8
514                     DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
515                   ; CURSOR_COL := 9
516                   ; EDIT_PROMPT ( DIRECTION_CHAR )
517                   END
518                   ELSE FOR J := 1 TO 8 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
519                 END
520                 ELSE BEGIN
521                   CURSOR_COL := CURSOR_COL - 1
522                 ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
523                 ; BOF := TRUE
524                 END
525               END
526               ELSE BEGIN
527                 B_M . MOVE_LEFT ( CURRENT_CHAR )
528               ; IF CURRENT_CHAR <> EM
529                 THEN BEGIN
530                   CURSOR_ROW := CURSOR_ROW + 1
531                 ; CURSOR_COL := 1
532                 ; IF CURSOR_ROW = 25
533                   THEN BEGIN
534                     IO . OUTPUT ( NL )
535                   ; MOVE_CURSOR ( 24 , 1 )
536                   ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , I )
537                   ; FOR J := 1 TO I DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
538                   ; CURSOR_COL := 1
```

```
539                 ; EDIT_PROMPT ( DIRECTION_CHAR )
540                 END
541               END
542               ELSE BEGIN
543                 CURSOR_COL := CURSOR_COL - 1
544               ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
545               END
546             END
547           END
548           ELSE B_M . MOVE_LEFT ( CURRENT_CHAR )
549         ; IF CURSOR_COL = 81
550           THEN CURSOR_COL := 80
551         ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
552         END
553
554     ; PROCEDURE RETURN ( VAR CURRENT_CHAR : CHAR )
555
556       ; VAR I , J , L : INTEGER
557         ; NEW_LINE : BOOLEAN
558
559       ; BEGIN
560           NEW_LINE := FALSE
561         ; MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , L )
562         ; IF SHOW_LINE_NUMBERS
563           THEN BEGIN
564             B_M . MOVE_LEFT ( CURRENT_CHAR )
565           ; IF CURRENT_CHAR <> EM
566             THEN BEGIN
567               CURSOR_ROW := CURSOR_ROW + 1
568             ; CURSOR_COL := 9
569             ; IF CURSOR_ROW = 25
570               THEN BEGIN
571                 IO . OUTPUT ( NL )
572               ; MOVE_CURSOR ( 24 , 1 )
573               ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , I )
574               ; FOR J := 1 TO I - 8 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
575               ; CURSOR_COL := 9
576               ; NEW_LINE := TRUE
577               END
578               ELSE FOR J := 1 TO 8 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
579             END
580             ELSE BEGIN
581               CURSOR_COL := CURSOR_COL + L
582             ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
583             END
584           END
585           ELSE BEGIN
586             B_M . MOVE_LEFT ( CURRENT_CHAR )
587           ; IF CURRENT_CHAR <> EM
588             THEN BEGIN
589               CURSOR_ROW := CURSOR_ROW + 1
590             ; CURSOR_COL := 1
591             ; IF CURSOR_ROW = 25
592               THEN BEGIN
```

```
593                 IO . OUTPUT ( NL )
594               ; MOVE_CURSOR ( 24 , 1 )
595               ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , I )
596               ; FOR J := 1 TO I DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
597               ; CURSOR_COL := 1
598               ; NEW_LINE := TRUE
599               END
600             END
601             ELSE BEGIN
602               CURSOR_COL := CURSOR_COL + L
603             ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
604             END
605           END
606         ; WHILE ( CURRENT_CHAR = ' ' ) AND ( CURSOR_COL < 80 )
607           DO BEGIN
608             CURSOR_COL := CURSOR_COL + 1
609           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
610           END
611         ; IF NEW_LINE
612           THEN EDIT_PROMPT ( DIRECTION_CHAR )
613           ELSE MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
614         END
615
616     ; PROCEDURE MOVE_PAGE
617         ( VAR CURRENT_CHAR : CHAR ; DIRECTION_CHAR : CHAR )
618
619       ; VAR I , J : INTEGER
620
621       ; BEGIN
622           IF DIRECTION_CHAR = '<'
623           THEN BEGIN
624             MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
625           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
626           ; IF CURRENT_CHAR <> EM
627             THEN BEGIN
628               FOR I := 1 TO J + 1 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
629             ; FOR I := 1 TO 23 DO MOVE_RIGHT_1_LINE ( CURRENT_CHAR , J )
630             ; CURSOR_COL := J
631             ; VERIFY ( CURRENT_CHAR )
632             ; EDIT_PROMPT ( DIRECTION_CHAR )
633             END
634             ELSE FOR I := 1 TO J + 1 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
635           END
636         ; IF DIRECTION_CHAR = '>'
637           THEN BEGIN
638             MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
639           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
640           ; IF CURRENT_CHAR <> EM
641             THEN BEGIN
642               FOR I := 1 TO J + 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
643             ; FOR I := 1 TO 23 DO MOVE_LEFT_1_LINE ( CURRENT_CHAR , J )
644             ; CURSOR_COL := J
645             ; VERIFY ( CURRENT_CHAR )
646             ; EDIT_PROMPT ( DIRECTION_CHAR )
```

```
647             END
648             ELSE FOR I := 1 TO J + 1
649               DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
650           END
651         END
652
653       ; PROCEDURE ADD_LINE_NUMBERS ( VAR CURRENT_CHAR : CHAR )
654
655         ; VAR LINE_NUMBER , NUMBER , J , I : INTEGER
656           ; INTEGER_STRING : ARRAY [ 1 .. 4 ] OF CHAR
657
658         ; BEGIN
659       "CHECK FOR BUFFER CONTAINING EM NL EM
660       IF IT DOES ADD A NL CHARACTOR. THEN MOVE BACK TO LEFT EM."
661           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
662           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
663           ; IF CURRENT_CHAR = EM
664             THEN BEGIN
665               B_M . ADD ( NL )
666             ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
667             END
668           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
669           ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
670           ; IO . OUTPUT ( NL )
671           ; IO . OUTPUT ( CR )
672           ; IO . SHIP_OUT
673           ; LINE_NUMBER := 0
674           ; REPEAT WHILE CURRENT_CHAR <> NL
675               DO B_M . MOVE_LEFT ( CURRENT_CHAR )
676             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
677             ; IF CURRENT_CHAR <> EM
678               THEN BEGIN
679                 LINE_NUMBER := ( LINE_NUMBER + 1 ) MOD 10000
680               ; NUMBER := LINE_NUMBER
681               ; IF NUMBER MOD 10 = 0
682                 THEN BEGIN
683                   IO . OUTPUT ( '<' )
684                 ; IO . SHIP_OUT
685                 ; IF NUMBER MOD 800 = 0
686                   THEN BEGIN
687                     IO . OUTPUT ( NL )
688                   ; IO . OUTPUT ( CR )
689                   ; IO . SHIP_OUT
690                   END
691                 END
692               ; FOR I := 4 DOWNTO 1
693                 DO BEGIN
694                   INTEGER_STRING [ I ] := CHR ( NUMBER MOD 10 + 48 )
695                 ; NUMBER := NUMBER DIV 10
696                 END
697               ; I := 1
698               ; WHILE ( INTEGER_STRING [ I ] = '0' ) AND ( I < 4 )
699                 DO BEGIN
700                   B_M . ADD ( ' ' )
```

```
701               ; I := I + 1
702               END
703             ; FOR J := I TO 4 DO B_M . ADD ( INTEGER_STRING [ J ] )
704             ; FOR J := 1 TO 4 DO B_M . ADD ( ' ' )
705             END
706           UNTIL CURRENT_CHAR = EM
707         ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
708                                                 "MOVE BACK TO BEGINNING."
709         ; LINE_NUMBER := 1
710         ; IO . OUTPUT ( NL )
711         ; IO . OUTPUT ( CR )
712         ; IO . SHIP_OUT
713         ; REPEAT B_M . MOVE_RIGHT ( CURRENT_CHAR )
714           ; IF CURRENT_CHAR = NL
715             THEN BEGIN
716               LINE_NUMBER := LINE_NUMBER + 1
717             ; IF LINE_NUMBER MOD 10 = 0
718               THEN BEGIN
719                 IO . OUTPUT ( '>' )
720               ; IO . SHIP_OUT
721               ; IF LINE_NUMBER MOD 800 = 0
722                 THEN BEGIN
723                   IO . OUTPUT ( NL )
724                 ; IO . OUTPUT ( CR )
725                 ; IO . SHIP_OUT
726                 END
727               END
728             END
729           UNTIL CURRENT_CHAR = EM
730         ; FOR I := 1 TO 10 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
731         END
732
733     ; PROCEDURE INSERT_PROMPT
734
735       ; VAR SROW , SCOL : INTEGER
736
737       ; BEGIN
738           SROW := CURSOR_ROW
739         ; SCOL := CURSOR_COL
740         ; MOVE_CURSOR ( 1 , 1 )
741         ; IO . OUTPUT_LINE
742             ( 'INSERT: [left arrow] a char, [del] a li$' )
743         ; IO . OUTPUT_LINE
744             ( 'ne, [esc] to abort, [ctrl<c>] to exit   $' )
745         ; MOVE_CURSOR ( SROW , SCOL )
746         END
747
748     ; PROCEDURE OUTPUT_FRONT_OF_LINE ( VAR CURRENT_CHAR : CHAR )
749
750       ; VAR I , L : INTEGER
751
752       ; BEGIN
753           IF CURRENT_CHAR = NL
754           THEN BEGIN B_M . MOVE_RIGHT ( CURRENT_CHAR ) ; I := 1 END
```

```
755           ELSE I := 0
756         ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , L )
757         ; L := L + I
758         ; B_M . MOVE_LEFT ( CURRENT_CHAR )
759         ; MOVE_CURSOR ( CURSOR_ROW , 1 )
760         ; FOR I := 1 TO L - 1
761           DO BEGIN
762             IO . OUTPUT ( CURRENT_CHAR )
763           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
764           END
765         ; CURSOR_COL := L
766         ; IO . SHIP_OUT
767         END
768
769       ; PROCEDURE SPREAD_END_OF_LINE ( VAR CURRENT_CHAR : CHAR )
770
771         ; VAR I , J , L , NUMBER_OF_SPACES : INTEGER
772
773         ; BEGIN
774             MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
775           ; FOR I := 1 TO J DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
776           ; NUMBER_OF_SPACES := 80 - J - CURSOR_COL + 1
777           ; IF NUMBER_OF_SPACES > 1
778             THEN BEGIN
779               SAVE_CURSOR_POSITION
780             ; IF CURRENT_CHAR <> NL
781               THEN BEGIN
782                 FOR I := 1 TO NUMBER_OF_SPACES DO IO . OUTPUT ( ' ' )
783               ; FOR I := 1 TO J
784                 DO BEGIN
785                   IO . OUTPUT ( CURRENT_CHAR )
786                 ; B_M . MOVE_LEFT ( CURRENT_CHAR )
787                 END
788               ; FOR I := 1 TO J DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
789               END
790             ; UNSAVE_CURSOR_POSITION
791             ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
792             END
793           END
794
795       ; PROCEDURE SPREAD_CURRENT_LINE
796           ( VAR CURRENT_CHAR : CHAR ; VAR NUMBER_OF_SPACES : INTEGER )
797
798         ; VAR I , L , J : INTEGER
799
800         ; BEGIN
801             MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
802           ; FOR I := 1 TO J DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
803           ; NUMBER_OF_SPACES := 80 - J - CURSOR_COL + 1
804           ; IF NUMBER_OF_SPACES > 1
805             THEN BEGIN
806               SAVE_CURSOR_POSITION
807             ; OUTPUT_FRONT_OF_LINE ( CURRENT_CHAR )
808             ; IF CURRENT_CHAR <> NL
```

```
809             THEN BEGIN
810               FOR I := 1 TO NUMBER_OF_SPACES DO IO . OUTPUT ( ' ' )
811             ; FOR I := 1 TO J
812               DO BEGIN
813                 IO . OUTPUT ( CURRENT_CHAR )
814               ; B_M . MOVE_LEFT ( CURRENT_CHAR )
815               END
816             ; FOR I := 1 TO J DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
817             END
818           ; UNSAVE_CURSOR_POSITION
819           ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
820           END
821       END
822
823     ; PROCEDURE SPLIT_CURRENT_LINE ( VAR CURRENT_CHAR : CHAR )
824
825       ; VAR I , J , L , K : INTEGER
826
827       ; BEGIN
828           OUTPUT_FRONT_OF_LINE ( CURRENT_CHAR )
829         ; SAVE_CURSOR_POSITION
830         ; FOR I := CURSOR_COL TO 80 DO IO . OUTPUT ( ' ' )
831         ; WHILE CURSOR_ROW < 23
832           DO BEGIN
833             CURSOR_ROW := CURSOR_ROW + 1
834           ; MOVE_CURSOR ( CURSOR_ROW , 1 )
835           ; FOR I := 1 TO 80 DO IO . OUTPUT ( ' ' )
836           ;
837           END
838         ; MOVE_CURSOR ( CURSOR_ROW + 1 , 1 )
839         ; MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
840         ; FOR I := 1 TO J DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
841         ; IF J > 81
842           THEN K := 81
843           ELSE K := J
844         ; FOR I := 1 TO 80 - K DO IO . OUTPUT ( ' ' )
845         ; FOR I := 1 TO K
846           DO BEGIN
847             IO . OUTPUT ( CURRENT_CHAR )
848           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
849           END
850         ; FOR I := 1 TO K DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
851         ; UNSAVE_CURSOR_POSITION
852         ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
853         END
854
855     ; PROCEDURE INSERT_A_NL_AND_LINE_NO ( VAR CURRENT_CHAR : CHAR )
856
857       ; VAR I , J : INTEGER
858         ; INTEGER_STRING : ARRAY [ 1 .. 4 ] OF CHAR
859         ; DECIMAL_STRING : ARRAY [ 1 .. 2 ] OF CHAR
860         ; INT , DEC : INTEGER
861
862       ; BEGIN
```

```
863             IF CURRENT_CHAR = NL
864             THEN BEGIN B_M . MOVE_RIGHT ( CURRENT_CHAR ) ; I := 1 END
865             ELSE I := 0
866           ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
867           ; J := I + J
868           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
869           ; INT := 0
870           ; FOR I := 1 TO 4
871             DO BEGIN
872               INTEGER_STRING [ I ] := CURRENT_CHAR
873             ; IF CURRENT_CHAR <> ' '
874               THEN INT := INT * 10 + ORD ( CURRENT_CHAR ) - 48
875             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
876             END
877           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
878           ; DEC := 0
879           ; FOR I := 1 TO 2
880             DO BEGIN
881               IF CURRENT_CHAR <> ' '
882               THEN DEC := DEC * 10 + ORD ( CURRENT_CHAR ) - 48
883             ; B_M . MOVE_LEFT ( CURRENT_CHAR )
884             END
885           ; FOR I := 1 TO J - 8 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
886           ; MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
887           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
888           ; IF CURRENT_CHAR = EM
889             THEN BEGIN
890             ; FOR I := 1 TO J + 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
891             ; B_M . ADD ( NL )
892             ; C_B_M . COPY_PUSH ( NL )
893             ; INT := INT + 1
894             ; FOR I := 4 DOWNTO 1
895               DO BEGIN
896                 INTEGER_STRING [ I ] := CHR ( INT MOD 10 + 48 )
897               ; INT := INT DIV 10
898               END
899             ; I := 1
900             ; WHILE ( INTEGER_STRING [ I ] = '0' ) AND ( I < 4 )
901               DO BEGIN
902                 B_M . ADD ( ' ' )
903               ; C_B_M . COPY_PUSH ( ' ' )
904               ; I := I + 1
905               END
906             ; FOR J := I TO 4
907               DO BEGIN
908                 B_M . ADD ( INTEGER_STRING [ J ] )
909               ; C_B_M . COPY_PUSH ( INTEGER_STRING [ J ] )
910               END
911             ; FOR J := 1 TO 4
912               DO BEGIN
913                 B_M . ADD ( ' ' )
914               ; C_B_M . COPY_PUSH ( ' ' )
915               END
916             END
```

```
917             ELSE BEGIN
918             ; FOR I := 1 TO J + 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
919             ; B_M . ADD ( NL )
920             ; C_B_M . COPY_PUSH ( NL )
921             ; FOR I := 1 TO 4
922               DO BEGIN
923                 B_M . ADD ( INTEGER_STRING [ I ] )
924               ; C_B_M . COPY_PUSH ( INTEGER_STRING [ I ] )
925               END
926             ; DEC := DEC + 1
927             ; B_M . ADD ( '.' )
928             ; C_B_M . COPY_PUSH ( '.' )
929             ; FOR I := 2 DOWNTO 1
930               DO BEGIN
931                 DECIMAL_STRING [ I ] := CHR ( DEC MOD 10 + 48 )
932               ; DEC := DEC DIV 10
933               END
934             ; FOR I := 1 TO 2
935               DO BEGIN
936               ; B_M . ADD ( DECIMAL_STRING [ I ] )
937               ; C_B_M . COPY_PUSH ( DECIMAL_STRING [ I ] )
938               END
939             ; B_M . ADD ( ' ' )
940             ; C_B_M . COPY_PUSH ( ' ' )
941             END
942           END
943
944       ; PROCEDURE OUTPUT_CURRENT_LINE ( VAR CURRENT_CHAR : CHAR )
945
946         ; VAR L , J , I : INTEGER
947
948         ; BEGIN
949             IF CURRENT_CHAR = NL
950             THEN BEGIN L := 1 ; B_M . MOVE_RIGHT ( CURRENT_CHAR ) END
951             ELSE L := 0
952           ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
953           ; L := L + J - 1
954           ; B_M . MOVE_LEFT ( CURRENT_CHAR )
955           ; MOVE_CURSOR ( CURSOR_ROW , 1 )
956           ; OUTPUT_REST_OF_LINE ( CURRENT_CHAR , J )
957           ; FOR I := J TO 80 DO IO . OUTPUT ( ' ' )
958           ; IF SHOW_LINE_NUMBERS
959             THEN BEGIN
960               CURSOR_COL := L + 1
961             ; FOR I := J DOWNTO L + 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
962             END
963             ELSE BEGIN
964               CURSOR_COL := L - 7
965             ; FOR I := J + 8 DOWNTO L DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
966             END
967           ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
968           END
969
970       ; PROCEDURE INSERT_MODE ( VAR CURRENT_CHAR : CHAR )
```

```
971
972       ; VAR C , CC : CHAR
973         ; I , J , L , SAVED_COPY_COUNT , COPY_COUNT , NUMBER_OF_SPACES
974           : INTEGER
975         ; SPREAD , SPLIT : BOOLEAN
976
977       ; BEGIN
978           INSERT_PROMPT
979         ; IO . INPUT ( C )
980         ; IF NOT ( C IN [ ETX , ESC ] )
981           THEN BEGIN
982             REPEAT C_B_M . COPY_POP ( CC ) UNTIL CC = EM
983           ; C_B_M . COPY_PUSH ( EM )
984           ; COPY_COUNT := 0
985           ; SPLIT := FALSE
986           ; SPREAD := TRUE
987           ; SPREAD_CURRENT_LINE ( CURRENT_CHAR , NUMBER_OF_SPACES )
988           ; IF NUMBER_OF_SPACES < 2
989             THEN BEGIN
990               SPLIT_CURRENT_LINE ( CURRENT_CHAR )
991             ; SPLIT := TRUE
992             ; SPREAD := FALSE
993             END
994           ; REPEAT IF C = CR
995               THEN BEGIN
996                 IF SPREAD
997                 THEN BEGIN
998                   FOR I := CURSOR_COL TO 80 DO IO . OUTPUT ( ' ' )
999                 ; IF ( CURSOR_ROW = 24 ) OR ( CURSOR_ROW = 23 )
1000                  THEN BEGIN
1001                    IO . OUTPUT ( NL )
1002                  ; IF SHOW_LINE_NUMBERS
1003                    THEN CURSOR_COL := 9
1004                    ELSE CURSOR_COL := 1
1005                  ; CURSOR_ROW := 24
1006                  ; INSERT_PROMPT
1007                  ; COPY_COUNT := COPY_COUNT + 9
1008                  ; INSERT_A_NL_AND_LINE_NO ( CURRENT_CHAR )
1009                  ; SPREAD_CURRENT_LINE
1010                      ( CURRENT_CHAR , NUMBER_OF_SPACES )
1011                  END
1012                  ELSE BEGIN
1013                    CURSOR_ROW := CURSOR_ROW + 1
1014                  ; COPY_COUNT := COPY_COUNT + 9
1015                  ; INSERT_A_NL_AND_LINE_NO ( CURRENT_CHAR )
1016                  ; SPREAD := FALSE
1017                  ; SPLIT := TRUE
1018                  ; SPLIT_CURRENT_LINE ( CURRENT_CHAR )
1019                  END
1020                END
1021                ELSE IF SPLIT
1022                  THEN BEGIN
1023                    IF CURSOR_ROW < 23
1024                    THEN BEGIN
```

```
1025                      CURSOR_ROW := CURSOR_ROW + 1
1026                    ; COPY_COUNT := COPY_COUNT + 9
1027                    ; INSERT_A_NL_AND_LINE_NO ( CURRENT_CHAR )
1028                    ; OUTPUT_FRONT_OF_LINE ( CURRENT_CHAR )
1029                    END
1030                    ELSE BEGIN
1031                    ; COPY_COUNT := COPY_COUNT + 9
1032                    ; INSERT_A_NL_AND_LINE_NO ( CURRENT_CHAR )
1033                    ; MOVE_CURSOR ( 24 , 1 )
1034                    ; IF CURRENT_CHAR <> NL
1035                      THEN FOR I := 1 TO 80 DO IO . OUTPUT ( ' ' )
1036                    ; FOR I := 1 TO 19 DO IO . OUTPUT ( NL )
1037                    ; MOVE_CURSOR ( 24 , 9 )
1038                    ; SPREAD_END_OF_LINE ( CURRENT_CHAR )
1039                    ; INSERT_PROMPT
1040                    ; CURSOR_ROW := 5
1041                    ; CURSOR_COL := 1
1042                    ; OUTPUT_FRONT_OF_LINE ( CURRENT_CHAR )
1043                    END
1044                  END
1045                END
1046                ELSE IF C IN SET_OF_VALID_INPUT_CHARACTERS
1047                  THEN BEGIN
1048                    B_M . ADD ( C )
1049                  ; C_B_M . COPY_PUSH ( C )
1050                  ; COPY_COUNT := COPY_COUNT + 1
1051                  ; CURSOR_COL := CURSOR_COL + 1
1052                  ; IO . OUTPUT ( C )
1053                  ; IO . SHIP_OUT
1054                  ; IF SPREAD
1055                    THEN BEGIN
1056                      NUMBER_OF_SPACES := NUMBER_OF_SPACES - 1
1057                    ; IF NUMBER_OF_SPACES < 2
1058                      THEN BEGIN
1059                        IF CURSOR_ROW = 24
1060                        THEN BEGIN
1061                          IO . OUTPUT ( NL )
1062                        ; CURSOR_ROW := CURSOR_ROW - 1
1063                        ; INSERT_PROMPT
1064                        END
1065                      ; SPLIT_CURRENT_LINE ( CURRENT_CHAR )
1066                      ; SPLIT := TRUE
1067                      ; SPREAD := FALSE
1068                      END
1069                    END
1070                  END
1071              ; IF ( COPY_COUNT > 0 )
1072                   AND ( ( ( ( CURSOR_COL = 1 )
1073                             OR ( SHOW_LINE_NUMBERS AND ( CURSOR_COL = 9 )
1074                                )
1075                           )
1076                           AND ( C = BS )
1077                         )
1078                         OR ( C = DEL )
```

```
1079                    )
1080                  THEN BEGIN
1081                    L := CURSOR_COL
1082                  ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1083                  ; WHILE ( COPY_COUNT > 0 ) AND ( CURRENT_CHAR <> NL )
1084                    DO BEGIN
1085                      B_M . DELETE ( CURRENT_CHAR )
1086                    ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1087                    ; COPY_COUNT := COPY_COUNT - 1
1088                    ; C_B_M . COPY_POP ( CC )
1089                    ; CURSOR_COL := CURSOR_COL - 1
1090                    END
1091                  ; IF ( COPY_COUNT = 0 ) AND ( CURRENT_CHAR <> NL )
1092                    THEN BEGIN
1093                    ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1094                    ; IF SPREAD
1095                      THEN BEGIN
1096                        SPREAD_CURRENT_LINE
1097                          ( CURRENT_CHAR , NUMBER_OF_SPACES )
1098                      ; IF CURRENT_CHAR = NL
1099                        THEN FOR I := CURSOR_COL TO 80
1100                          DO IO . OUTPUT ( ' ' )
1101                      END
1102                      ELSE BEGIN
1103                        MOVE_CURSOR ( CURSOR_ROW , 9 )
1104                      ; FOR I := 9 TO 80 DO IO . OUTPUT ( ' ' )
1105                      ; OUTPUT_FRONT_OF_LINE ( CURRENT_CHAR )
1106                      END
1107                    END
1108                    ELSE BEGIN
1109                      MOVE_CURSOR ( CURSOR_ROW , 1 )
1110                    ; FOR I := 1 TO L DO IO . OUTPUT ( ' ' )
1111                    ; B_M . DELETE ( CURRENT_CHAR )
1112                    ; COPY_COUNT := COPY_COUNT - 1
1113                    ; C_B_M . COPY_POP ( CC )
1114                    ; IF CURRENT_CHAR = NL
1115                      THEN BEGIN
1116                        I := 1
1117                      ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1118                      END
1119                      ELSE I := 0
1120                    ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
1121                    ; J := J + I
1122                    ; FOR I := 1 TO J DO B_M . MOVE_LEFT ( CURRENT_CHAR )
1123                    ; CURSOR_COL := J
1124                    ; CURSOR_ROW := CURSOR_ROW - 1
1125                    ; IF CURSOR_ROW = 1
1126                      THEN BEGIN
1127                        CURSOR_ROW := 2
1128                      ; OUTPUT_FRONT_OF_LINE ( CURRENT_CHAR )
1129                      END
1130                      ELSE MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
1131                    END
1132                  END
```

```
1133              ELSE IF ( C = BS ) AND ( COPY_COUNT > 0 )
1134                THEN BEGIN
1135                  COPY_COUNT := COPY_COUNT - 1
1136                ; C_B_M . COPY_POP ( CC )
1137                ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1138                ; B_M . DELETE ( CURRENT_CHAR )
1139                ; IF CURSOR_COL < 81
1140                  THEN BEGIN
1141                    IO . OUTPUT ( BS )
1142                  ; IO . OUTPUT ( ' ' )
1143                  ; IO . OUTPUT ( BS )
1144                  END
1145                  ELSE IO . OUTPUT ( ' ' )
1146                ; CURSOR_COL := CURSOR_COL - 1
1147                ; IO . SHIP_OUT
1148                END
1149            ; IO . INPUT ( C )
1150            UNTIL C IN [ ESC , ETX ]
1151          ; IF C = ESC
1152            THEN "DELETE COPY_COUNT OF CHAR BACKWORDS." BEGIN
1153              IF COPY_COUNT > 0
1154              THEN BEGIN
1155                FOR I := COPY_COUNT DOWNTO 1
1156                DO BEGIN
1157                  B_M . MOVE_RIGHT ( CURRENT_CHAR )
1158                ; IF CURRENT_CHAR = NL
1159                  THEN SPLIT := TRUE
1160                ; B_M . DELETE ( CURRENT_CHAR )
1161                END
1162              END
1163            END
1164          ; IF SPLIT
1165            THEN VERIFY ( CURRENT_CHAR )
1166            ELSE OUTPUT_CURRENT_LINE ( CURRENT_CHAR )
1167          END
1168        END
1169
1170      ; PROCEDURE DIRECTION ( VAR COMMAND_CHAR , DIRECTION_CHAR : CHAR )
1171
1172        ; BEGIN
1173            IF COMMAND_CHAR IN [ '<' , ',' ]
1174            THEN BEGIN
1175              DIRECTION_CHAR := '<'
1176            ; SAVE_CURSOR_POSITION
1177            ; MOVE_CURSOR ( 1 , 1 )
1178            ; IO . OUTPUT ( '<' )
1179            ; UNSAVE_CURSOR_POSITION
1180            ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
1181            END
1182          ; IF COMMAND_CHAR IN [ '>' , '.' ]
1183            THEN BEGIN
1184              DIRECTION_CHAR := '>'
1185            ; SAVE_CURSOR_POSITION
1186            ; MOVE_CURSOR ( 1 , 1 )
```

```
1187            ; IO . OUTPUT ( '>' )
1188            ; UNSAVE_CURSOR_POSITION
1189            ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
1190            END
1191          END
1192
1193      ; PROCEDURE DELETE_PROMPT ( DIRECTION_CHAR : CHAR )
1194
1195        ; VAR SROW , SCOL : INTEGER
1196
1197        ; BEGIN
1198            SROW := CURSOR_ROW
1199          ; SCOL := CURSOR_COL
1200          ; MOVE_CURSOR ( 1 , 1 )
1201          ; IO . OUTPUT ( DIRECTION_CHAR )
1202          ; IO . OUTPUT_LINE
1203              ( 'DELETE: [moving commands], [esc] to abor$' )
1204          ; IO . OUTPUT_LINE
1205              ( 't, [ctrl<c>] to exit                    $' )
1206          ; MOVE_CURSOR ( SROW , SCOL )
1207          END
1208
1209      ; PROCEDURE BLANK_RIGHT
1210          ( VAR CURRENT_CHAR : CHAR
1211          ; VAR J : INTEGER
1212          ; VAR BOF : BOOLEAN
1213          )
1214
1215        ; VAR I : INTEGER
1216
1217        ; BEGIN
1218            IF CURRENT_CHAR = NL
1219            THEN BEGIN
1220              RIGHT_ARROW ( CURRENT_CHAR , BOF )
1221            ; IF NOT BOF
1222              THEN BEGIN
1223                J := J + 9
1224              ; C_B_M . COPY_PUSH ( NL )
1225              ; IF SHOW_LINE_NUMBERS
1226                THEN BEGIN
1227                  MOVE_CURSOR ( CURSOR_ROW , 1 )
1228                ; IO . OUTPUT_LINE ( '        $' )
1229                ; CURSOR_COL := 9
1230                END
1231              ; FOR I := 8 DOWNTO 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
1232              ; FOR I := 1 TO 8
1233                DO BEGIN
1234                  C_B_M . COPY_PUSH ( CURRENT_CHAR )
1235                ; CURRENT_CHAR := ' '
1236                ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1237                END
1238              END
1239            END
1240            ELSE BEGIN
```

```
1241              J := J + 1
1242            ; C_B_M . COPY_PUSH ( CURRENT_CHAR )
1243            ; CURRENT_CHAR := ' '
1244            ; IO . OUTPUT ( ' ' )
1245            ; RIGHT_ARROW ( CURRENT_CHAR , BOF )
1246            END
1247          END
1248
1249      ; PROCEDURE BLANK_LEFT
1250          ( VAR CURRENT_CHAR : CHAR
1251          ; VAR J : INTEGER
1252          ; VAR TOF : BOOLEAN
1253          )
1254
1255        ; VAR I : INTEGER
1256
1257        ; BEGIN
1258            IF ( SHOW_LINE_NUMBERS AND ( CURSOR_COL = 9 ) )
1259               OR ( CURSOR_COL = 1 )
1260            THEN BEGIN
1261              LEFT_ARROW ( CURRENT_CHAR , TOF )
1262            ; IF NOT TOF
1263              THEN BEGIN
1264                J := J - 9
1265              ; IF SHOW_LINE_NUMBERS
1266                THEN BEGIN
1267                  SAVE_CURSOR_POSITION
1268                ; CURSOR_ROW := CURSOR_ROW + 1
1269                ; MOVE_CURSOR ( CURSOR_ROW , 1 )
1270                ; IO . OUTPUT_LINE ( '        $' )
1271                ; UNSAVE_CURSOR_POSITION
1272                ; MOVE_CURSOR ( CURSOR_ROW , CURSOR_COL )
1273                END
1274              ; FOR I := 0 TO 8 DO B_M . MOVE_LEFT ( CURRENT_CHAR )
1275              ; FOR I := 8 DOWNTO 1
1276                DO BEGIN
1277                  B_M . MOVE_RIGHT ( CURRENT_CHAR )
1278                ; C_B_M . COPY_PUSH ( CURRENT_CHAR )
1279                ; CURRENT_CHAR := ' '
1280                END
1281              ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1282              ; C_B_M . COPY_PUSH ( NL )
1283              END
1284            END
1285            ELSE BEGIN
1286              J := J - 1
1287            ; LEFT_ARROW ( CURRENT_CHAR , TOF )
1288            ; C_B_M . COPY_PUSH ( CURRENT_CHAR )
1289            ; IO . OUTPUT ( ' ' )
1290            ; IO . OUTPUT ( BS )
1291            ; IO . SHIP_OUT
1292            ; CURRENT_CHAR := ' '
1293            END
1294          END
```

```
1295
1296      ; PROCEDURE RECOVER_LEFT
1297          ( VAR CURRENT_CHAR : CHAR ; VAR J : INTEGER )
1298
1299        ; VAR I : INTEGER
1300          ; NOTUSED : BOOLEAN
1301
1302        ; BEGIN
1303            IF ( SHOW_LINE_NUMBERS AND ( CURSOR_COL = 9 ) )
1304               OR ( CURSOR_COL = 1 )
1305            THEN BEGIN
1306              J := J - 9
1307            ; FOR I := 8 DOWNTO 1
1308              DO BEGIN
1309                B_M . MOVE_RIGHT ( CURRENT_CHAR )
1310              ; C_B_M . COPY_POP ( CURRENT_CHAR )
1311              ; IF SHOW_LINE_NUMBERS
1312                THEN BEGIN
1313                  IO . OUTPUT ( BS )
1314                ; IO . OUTPUT ( CURRENT_CHAR )
1315                ; IO . OUTPUT ( BS )
1316                ; IO . SHIP_OUT
1317                END
1318              END
1319            ; LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1320            ; C_B_M . COPY_POP ( CURRENT_CHAR )
1321            END
1322            ELSE BEGIN
1323              J := J - 1
1324            ; LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1325            ; C_B_M . COPY_POP ( CURRENT_CHAR )
1326            ; IF CURSOR_COL <> 80
1327              THEN BEGIN
1328                IO . OUTPUT ( CURRENT_CHAR )
1329              ; IO . OUTPUT ( BS )
1330              END
1331              ELSE IO . OUTPUT ( '!' )
1332            ; IO . SHIP_OUT
1333            END
1334          END
1335
1336      ; PROCEDURE RECOVER_RIGHT
1337          ( VAR CURRENT_CHAR : CHAR ; VAR J : INTEGER )
1338
1339        ; VAR I : INTEGER
1340          ; NOTUSED : BOOLEAN
1341
1342        ; BEGIN
1343            IF CURRENT_CHAR = NL
1344            THEN BEGIN
1345              J := J + 9
1346            ; C_B_M . COPY_POP ( CURRENT_CHAR )
1347            ; RIGHT_ARROW ( CURRENT_CHAR , NOTUSED )
1348            ; FOR I := 8 DOWNTO 1 DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
```

```
1349            ; IF SHOW_LINE_NUMBERS
1350              THEN MOVE_CURSOR ( CURSOR_ROW , 1 )
1351            ; FOR I := 1 TO 8
1352              DO BEGIN
1353                C_B_M . COPY_POP ( CURRENT_CHAR )
1354              ; IF SHOW_LINE_NUMBERS
1355                THEN IO . OUTPUT ( CURRENT_CHAR )
1356              ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1357              END
1358            ; CURSOR_COL := 9
1359            ; IO . SHIP_OUT
1360            END
1361            ELSE BEGIN
1362              J := J + 1
1363            ; C_B_M . COPY_POP ( CURRENT_CHAR )
1364            ; IO . OUTPUT ( CURRENT_CHAR )
1365            ; RIGHT_ARROW ( CURRENT_CHAR , NOTUSED )
1366            END
1367          END
1368
1369        ; PROCEDURE DELETE_MODE_DOWN
1370            ( VAR CURRENT_CHAR : CHAR
1371            ; VAR COPY_COUNT , P_TARGET_COL : INTEGER
1372            )
1373
1374          ; VAR BOF : BOOLEAN
1375            ; TARGET_COL : INTEGER
1376
1377          ; BEGIN
1378            ; TARGET_COL := P_TARGET_COL
1379            ; IF COPY_COUNT > - 1
1380              THEN BEGIN
1381                BOF := FALSE
1382              ; WHILE ( CURRENT_CHAR <> NL ) AND NOT BOF
1383                DO BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1384              ; IF NOT BOF
1385                THEN BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1386              ; WHILE NOT BOF
1387                      AND ( CURRENT_CHAR <> NL )
1388                      AND ( CURSOR_COL < TARGET_COL )
1389                DO BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1390              END
1391              ELSE BEGIN
1392              ; WHILE ( CURRENT_CHAR <> NL ) AND ( COPY_COUNT <> 0 )
1393                DO RECOVER_RIGHT ( CURRENT_CHAR , COPY_COUNT )
1394              ; IF COPY_COUNT = 0
1395                THEN BEGIN
1396                  WHILE ( CURRENT_CHAR <> NL ) AND NOT BOF
1397                  DO BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1398                ; IF NOT BOF
1399                  THEN BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1400                ; WHILE NOT BOF
1401                        AND ( CURRENT_CHAR <> NL )
1402                        AND ( CURSOR_COL < TARGET_COL )
```

```
1403              DO BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1404            END
1405            ELSE BEGIN
1406              RECOVER_RIGHT ( CURRENT_CHAR , COPY_COUNT )
1407            ; WHILE ( COPY_COUNT < 0 )
1408                    AND ( CURRENT_CHAR <> NL )
1409                    AND ( CURSOR_COL < TARGET_COL )
1410              DO RECOVER_RIGHT ( CURRENT_CHAR , COPY_COUNT )
1411            ; WHILE ( CURRENT_CHAR <> NL )
1412                    AND ( CURSOR_COL < TARGET_COL )
1413              DO BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , BOF )
1414            END
1415          END
1416        END
1417
1418    ; PROCEDURE DELETE_MODE_RETURN
1419        ( VAR CURRENT_CHAR : CHAR ; VAR COPY_COUNT : INTEGER )
1420
1421      ; VAR COL : INTEGER
1422
1423      ; BEGIN
1424          IF SHOW_LINE_NUMBERS
1425          THEN BEGIN
1426            COL := 9
1427          ; DELETE_MODE_DOWN ( CURRENT_CHAR , COPY_COUNT , COL )
1428          END
1429          ELSE BEGIN
1430            COL := 1
1431          ; DELETE_MODE_DOWN ( CURRENT_CHAR , COPY_COUNT , COL )
1432          END
1433        END
1434
1435    ; PROCEDURE DELETE_MODE_UP
1436        ( VAR CURRENT_CHAR : CHAR ; VAR COPY_COUNT : INTEGER )
1437
1438      ; VAR TOF : BOOLEAN
1439        ; TARGET_COL : INTEGER
1440
1441      ; BEGIN
1442          TARGET_COL := CURSOR_COL
1443        ; IF COPY_COUNT < 1
1444          THEN BEGIN
1445            TOF := FALSE
1446          ; WHILE ( CURRENT_CHAR <> NL ) AND NOT TOF
1447            DO BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1448          ; IF NOT TOF
1449            THEN BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1450          ; WHILE NOT TOF AND ( CURSOR_COL > TARGET_COL )
1451            DO BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1452          END
1453          ELSE BEGIN
1454            WHILE ( CURRENT_CHAR <> NL ) AND ( COPY_COUNT <> 0 )
1455            DO RECOVER_LEFT ( CURRENT_CHAR , COPY_COUNT )
1456          ; IF COPY_COUNT = 0
```

```
1457              THEN BEGIN
1458                WHILE ( CURRENT_CHAR <> NL ) AND NOT TOF
1459                DO BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1460              ; IF NOT TOF
1461                THEN BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1462              ; WHILE CURSOR_COL > TARGET_COL
1463                DO BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1464              END
1465              ELSE BEGIN
1466                RECOVER_LEFT ( CURRENT_CHAR , COPY_COUNT )
1467              ; WHILE ( COPY_COUNT > 0 )
1468                      AND ( CURRENT_CHAR <> NL )
1469                      AND ( CURSOR_COL > TARGET_COL )
1470                DO RECOVER_LEFT ( CURRENT_CHAR , COPY_COUNT )
1471              ; WHILE CURSOR_COL > TARGET_COL
1472                DO BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , TOF )
1473              END
1474            END
1475          END
1476
1477      ; PROCEDURE DELETE_MODE ( VAR CURRENT_CHAR : CHAR )
1478
1479        ; VAR C , CC : CHAR
1480          ; NL_FOUND : BOOLEAN
1481          ; COPY_COUNT , J , I , L , SAVED_COPY_COUNT : INTEGER
1482          ; NOTUSED : BOOLEAN
1483
1484        ; BEGIN
1485            DELETE_PROMPT ( DIRECTION_CHAR )
1486          ; IO . INPUT ( C )
1487          ; IF NOT ( C IN [ ESC , ETX ] )
1488            THEN BEGIN
1489              REPEAT C_B_M . COPY_POP ( CC ) UNTIL CC = EM
1490            ; C_B_M . COPY_PUSH ( CC )
1491            ; COPY_COUNT := 0
1492            ; REPEAT IF ( C = BS )
1493                        OR ( ( C = ' ' ) AND ( DIRECTION_CHAR = '<' ) )
1494                THEN IF COPY_COUNT < 1
1495                  THEN BLANK_LEFT ( CURRENT_CHAR , COPY_COUNT , NOTUSED )
1496                  ELSE RECOVER_LEFT ( CURRENT_CHAR , COPY_COUNT )
1497              ; IF ( C = FF )
1498                   OR ( ( C = ' ' ) AND ( DIRECTION_CHAR = '>' ) )
1499                THEN IF COPY_COUNT > - 1
1500                  THEN BLANK_RIGHT ( CURRENT_CHAR , COPY_COUNT , NOTUSED )
1501                  ELSE RECOVER_RIGHT ( CURRENT_CHAR , COPY_COUNT )
1502              ; IF C = NL
1503                THEN DELETE_MODE_DOWN
1504                    ( CURRENT_CHAR , COPY_COUNT , CURSOR_COL )
1505              ; IF C = VT
1506                THEN DELETE_MODE_UP ( CURRENT_CHAR , COPY_COUNT )
1507              ; IF C = CR
1508                THEN DELETE_MODE_RETURN ( CURRENT_CHAR , COPY_COUNT )
1509              ; IF C IN [ 'V' , 'v' ]
1510                THEN BEGIN
```

```
1511                VERIFY ( CURRENT_CHAR )
1512              ; DELETE_PROMPT ( DIRECTION_CHAR )
1513                END
1514            ; IF C IN [ '>' , '<' , '.' , ',' ]
1515              THEN DIRECTION ( C , DIRECTION_CHAR )
1516            ; IO . INPUT ( C )
1517            UNTIL C IN [ ESC , ETX ]
1518          ; IF C = ESC
1519            THEN BEGIN
1520              SAVED_COPY_COUNT := COPY_COUNT
1521            ; WHILE COPY_COUNT < 0
1522              DO RECOVER_RIGHT ( CURRENT_CHAR , COPY_COUNT )
1523            ; WHILE COPY_COUNT > 0
1524              DO RECOVER_LEFT ( CURRENT_CHAR , COPY_COUNT )
1525            ; IF SAVED_COPY_COUNT < 0
1526              THEN BEGIN
1527                FOR I := - 1 DOWNTO SAVED_COPY_COUNT
1528                DO BEGIN
1529                  B_M . MOVE_RIGHT ( CURRENT_CHAR )
1530                ; C_B_M . COPY_PUSH ( CURRENT_CHAR )
1531                END
1532              ; FOR I := - 1 DOWNTO SAVED_COPY_COUNT
1533                DO B_M . MOVE_LEFT ( CURRENT_CHAR )
1534              END
1535            ; IF SAVED_COPY_COUNT > 0
1536              THEN BEGIN
1537                FOR I := 1 TO SAVED_COPY_COUNT
1538                DO BEGIN
1539                  C_B_M . COPY_PUSH ( CURRENT_CHAR )
1540                ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1541                END
1542              ; FOR I := 1 TO SAVED_COPY_COUNT
1543                DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
1544              END
1545            END
1546          ; IF C = ETX
1547            THEN BEGIN
1548              NL_FOUND := FALSE
1549            ; IF COPY_COUNT < 0
1550              THEN FOR I := - 1 DOWNTO COPY_COUNT
1551                DO BEGIN
1552                  IF CURRENT_CHAR = NL
1553                  THEN NL_FOUND := TRUE
1554                ; B_M . DELETE ( CURRENT_CHAR )
1555                END
1556            ; IF COPY_COUNT > 0
1557              THEN FOR I := 1 TO COPY_COUNT
1558                DO BEGIN
1559                  B_M . MOVE_RIGHT ( CURRENT_CHAR )
1560                ; IF CURRENT_CHAR = NL
1561                  THEN NL_FOUND := TRUE
1562                ; B_M . DELETE ( CURRENT_CHAR )
1563                END
1564            ; IF NL_FOUND
```

```
1565              THEN VERIFY ( CURRENT_CHAR )
1566              ELSE OUTPUT_CURRENT_LINE ( CURRENT_CHAR )
1567            END
1568          END
1569        ; EDIT_PROMPT ( DIRECTION_CHAR )
1570        END
1571
1572    ; PROCEDURE EXCHANGE_PROMPT
1573
1574      ; VAR SROW , SCOL : INTEGER
1575
1576      ; BEGIN
1577          SROW := CURSOR_ROW
1578        ; SCOL := CURSOR_COL
1579        ; MOVE_CURSOR ( 1 , 1 )
1580        ; IO . OUTPUT_LINE
1581            ( 'EXCHANGE: to return characters [left arr$' )
1582        ; IO . OUTPUT_LINE
1583            ( 'ow], to exit [ctrl<c>], to abort [esc]  $' )
1584        ; MOVE_CURSOR ( SROW , SCOL )
1585        END
1586
1587    ; PROCEDURE EXCHANGE_MODE ( VAR CURRENT_CHAR : CHAR )
1588
1589      ; VAR C : CHAR
1590        ; COPY_COUNT , I : INTEGER
1591        ; NOTUSED : BOOLEAN
1592
1593      ; BEGIN
1594          COPY_COUNT := 0
1595        ; EXCHANGE_PROMPT
1596        ; IO . INPUT ( C )
1597        ; WHILE NOT ( C IN [ ESC , ETX ] )
1598          DO BEGIN
1599            IF ( C IN SET_OF_VALID_INPUT_CHARACTERS )
1600               AND ( CURRENT_CHAR <> NL )
1601            THEN BEGIN
1602              C_B_M . COPY_PUSH ( CURRENT_CHAR )
1603            ; COPY_COUNT := COPY_COUNT + 1
1604            ; CURRENT_CHAR := C
1605            ; IO . OUTPUT ( CURRENT_CHAR )
1606            ; RIGHT_ARROW ( CURRENT_CHAR , NOTUSED )
1607            END
1608          ; IF ( C = FF ) AND ( CURRENT_CHAR <> NL )
1609            THEN BEGIN
1610              C_B_M . COPY_PUSH ( CURRENT_CHAR )
1611            ; COPY_COUNT := COPY_COUNT + 1
1612            ; RIGHT_ARROW ( CURRENT_CHAR , NOTUSED )
1613            END
1614          ; IF C = BS
1615            THEN BEGIN
1616              IF ( COPY_COUNT = 0 )
1617              THEN LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1618            ; IF COPY_COUNT > 0
```

```
1619                THEN BEGIN
1620                  LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1621                ; C_B_M . COPY_POP ( CURRENT_CHAR )
1622                ; COPY_COUNT := COPY_COUNT - 1
1623                ; IO . OUTPUT ( CURRENT_CHAR )
1624                ; IO . OUTPUT ( BS )
1625                ; IO . SHIP_OUT
1626                END
1627              END
1628            ; IO . INPUT ( C )
1629            END
1630          ; IF C = ETX
1631            THEN FOR I := 1 TO COPY_COUNT DO C_B_M . COPY_POP ( C )
1632          ; IF C = ESC
1633            THEN BEGIN
1634              FOR I := 1 TO COPY_COUNT
1635              DO BEGIN
1636                LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1637              ; C_B_M . COPY_POP ( CURRENT_CHAR )
1638              ; IO . OUTPUT ( CURRENT_CHAR )
1639              END
1640            ; IO . SHIP_OUT
1641            END
1642          END
1643
1644      ; PROCEDURE QUIT_MODE
1645          ( VAR DONE : BOOLEAN ; VAR CURRENT_CHAR : CHAR )
1646
1647        ; VAR I , COUNT : INTEGER
1648
1649        ; BEGIN
1650            DONE := TRUE
1651          ; MOVE_CURSOR ( 24 , 1 )
1652          ; IO . OUTPUT ( NL )
1653          ; IO . SHIP_OUT
1654          ; IO . OUTPUT_LINE ( 'QUITTING$' )
1655          ; WHILE CURRENT_CHAR <> EM DO B_M . MOVE_RIGHT ( CURRENT_CHAR )
1656          ; FOR I := 1 TO 10 DO B_M . DELETE ( CURRENT_CHAR )
1657          ; COUNT := 90
1658          ; IO . OUTPUT ( '''' )
1659          ; IO . SHIP_OUT
1660          ; REPEAT WHILE CURRENT_CHAR <> NL
1661              DO B_M . MOVE_LEFT ( CURRENT_CHAR )
1662            ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1663            ; IF CURRENT_CHAR <> EM
1664              THEN BEGIN
1665                FOR I := 1 TO 8 DO B_M . DELETE ( CURRENT_CHAR )
1666              ; COUNT := COUNT + 1
1667              ; IF COUNT MOD 10 = 0
1668                THEN BEGIN
1669                  IO . OUTPUT ( '''' )
1670                ; IO . SHIP_OUT
1671                ; IF COUNT MOD 800 = 0
1672                  THEN BEGIN
```

```
1673                  IO . OUTPUT ( NL )
1674                ; IO . OUTPUT ( CR )
1675                ; IO . SHIP_OUT
1676                END
1677              END
1678            END
1679          UNTIL CURRENT_CHAR = EM
1680        ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1681        ; B_M . FILL_LEFT_STACK
1682        ; IO . OUTPUT ( NL )
1683        ; IO . OUTPUT ( CR )
1684        ; IO . SHIP_OUT
1685        END
1686
1687    ; PROCEDURE SPACE ( VAR CURRENT_CHAR , DIRECTION_CHAR : CHAR )
1688
1689      ; VAR NOTUSED : BOOLEAN
1690
1691      ; BEGIN
1692          IF DIRECTION_CHAR = '<'
1693          THEN LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1694        ; IF DIRECTION_CHAR = '>'
1695          THEN RIGHT_ARROW ( CURRENT_CHAR , NOTUSED )
1696        END
1697
1698    ; PROCEDURE JUMP_PROMPT
1699
1700      ; VAR SROW , SCOL : INTEGER
1701
1702      ; BEGIN
1703          SROW := CURSOR_ROW
1704        ; SCOL := CURSOR_COL
1705        ; MOVE_CURSOR ( 1 , 1 )
1706        ; IO . OUTPUT_LINE
1707            (
1708  'JUMP: E(nd B(eginning L(ine number,  to exit [space,ctrl<c>,or esc]$'
1709            )
1710        ; IO . OUTPUT_LINE ( '            $' )
1711        ; MOVE_CURSOR ( SROW , SCOL )
1712        END
1713
1714    ; PROCEDURE JUMP_MODE ( VAR CURRENT_CHAR : CHAR )
1715
1716      ; VAR C : CHAR
1717
1718      ; BEGIN
1719          JUMP_PROMPT
1720        ; REPEAT IO . INPUT ( C )
1721          ; IF C IN [ 'E' , 'e' ]
1722            THEN BEGIN
1723              B_M . MOVE_LEFT ( CURRENT_CHAR )
1724            ; IF CURRENT_CHAR = EM
1725              THEN B_M . MOVE_RIGHT ( CURRENT_CHAR )
1726              ELSE BEGIN
```

```
1727                MOVE_TO_END_OF_TEXT ( CURRENT_CHAR )
1728              ; VERIFY ( CURRENT_CHAR )
1729              END
1730            END
1731          ; IF C IN [ 'B' , 'b' ]
1732            THEN BEGIN
1733              B_M . MOVE_RIGHT ( CURRENT_CHAR )
1734            ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1735            ; IF CURRENT_CHAR = EM
1736              THEN BEGIN
1737                B_M . MOVE_LEFT ( CURRENT_CHAR )
1738              ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1739              END
1740              ELSE BEGIN
1741                MOVE_TO_BEGINNING_OF_TEXT ( CURRENT_CHAR )
1742              ; VERIFY ( CURRENT_CHAR )
1743              END
1744            END
1745          UNTIL ( C IN [ ' ' , ESC , ETX , CR , 'E' , 'e' , 'B' , 'b' ]
1746                )
1747        END
1748
1749      ; PROCEDURE FIX_PROMPT
1750
1751        ; BEGIN
1752          ; CLEAR_SCREEN
1753          ; IO . OUTPUT_LINE
1754              ( 'FIX: Fix mode will let you rid your file of$' )
1755          ; IO . OUTPUT_LINE ( ' unwanted characters.(:10:)(:13:)$' )
1756          ; IO . OUTPUT_LINE
1757              ( '(:10:)Type a  G  or  g  to start the search$' )
1758          ; IO . OUTPUT_LINE
1759              ( ' from the current cursor position.(:10:)(:13:)$' )
1760          ; IO . OUTPUT_LINE
1761              (
1762       '(:10:)All nonprinting characters except linefeed and(:10:)(:13:)$'
1763              )
1764          ; IO . OUTPUT_LINE
1765              ( '(:10:)tab characters will be deleteted.(:10:)(:13:)$' )
1766          ; IO . OUTPUT_LINE
1767              ( '(:10:)All trailing blanks will be deleted.(:10:)(:13:)$'
1768              )
1769          ; IO . OUTPUT_LINE ( '(:10:)Type anything else to continue.$' )
1770          END
1771
1772      ; PROCEDURE FIX_MODE ( VAR CURRENT_CHAR : CHAR )
1773
1774        ; VAR C : CHAR
1775          ; J , I : INTEGER
1776
1777        ; BEGIN
1778            FIX_PROMPT
1779          ; IO . INPUT ( C )
1780          ; IF C IN [ 'G' , 'g' ]
```

```
1781            THEN BEGIN
1782            ; IO . OUTPUT_LINE ( ' GO.(:10:)(:8:)(:8:)(:8:)WORKING.$' )
1783            ; REPEAT IF CURRENT_CHAR = NL
1784                THEN BEGIN
1785                  B_M . MOVE_RIGHT ( CURRENT_CHAR )
1786                ; MOVE_TO_END_OF_PREVIOUS_LINE ( CURRENT_CHAR , J )
1787                ; FOR I := 1 TO J DO B_M . MOVE_LEFT ( CURRENT_CHAR )
1788                ; WHILE ( CURRENT_CHAR = ' ' ) AND ( J > 8 )
1789                  DO BEGIN
1790                    B_M . DELETE ( CURRENT_CHAR )
1791                  ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1792                  ; J := J - 1
1793                  END
1794                ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1795                ; B_M . MOVE_LEFT ( CURRENT_CHAR )
1796                END
1797                ELSE IF CURRENT_CHAR
1798                        IN SET_OF_VALID_INPUT_CHARACTERS + [ TAB ]
1799                  THEN B_M . MOVE_LEFT ( CURRENT_CHAR )
1800                  ELSE BEGIN
1801                    B_M . DELETE ( CURRENT_CHAR )
1802                  ; IO . OUTPUT_LINE
1803                      ( '(:10:)(:13:)BAD CHARACTER FOUND!$' )
1804                  END
1805              UNTIL CURRENT_CHAR = EM
1806            ; B_M . MOVE_RIGHT ( CURRENT_CHAR )
1807            ; IO . OUTPUT_LINE ( '(:10:)(:10:)(:13:)DONE$' )
1808            ; IO . OUTPUT_LINE
1809                ( '(:10:)(:13:)Type any key to continue.$' )
1810            ; IO . INPUT ( C )
1811            END
1812          END
1813
1814        ; PROCEDURE SECOND_EDIT_PROMPT ( DIRECTION_CHAR : CHAR )
1815
1816          ; VAR SROW , SCOL : INTEGER
1817
1818          ; BEGIN
1819              IF SECOND_EDIT_PROMPT_ON
1820              THEN BEGIN
1821                EDIT_PROMPT ( DIRECTION_CHAR )
1822              ; SECOND_EDIT_PROMPT_ON := FALSE
1823              END
1824              ELSE BEGIN
1825                SECOND_EDIT_PROMPT_ON := TRUE
1826              ; SROW := CURSOR_ROW
1827              ; SCOL := CURSOR_COL
1828              ; MOVE_CURSOR ( 1 , 1 )
1829              ; IO . OUTPUT ( DIRECTION_CHAR )
1830              ; IO . OUTPUT_LINE
1831                  ( 'SEDIT:   < > [movement commands]        $' )
1832              ; IO . OUTPUT_LINE
1833                  ( '                                       $' )
1834              ; MOVE_CURSOR ( SROW , SCOL )
```

```
1835            END
1836          END
1837
1838      ; PROCEDURE EXECUTE_COMMAND
1839          ( VAR CURRENT_CHAR , COMMAND_CHAR , DIRECTION_CHAR : CHAR
1840          ; VAR DONE : BOOLEAN
1841          )
1842
1843        ; VAR NOTUSED : BOOLEAN
1844          ; J , N : INTEGER
1845
1846        ; BEGIN
1847            CASE COMMAND_CHAR
1848            OF
1849              BS : LEFT_ARROW ( CURRENT_CHAR , NOTUSED )
1850            ; FF : RIGHT_ARROW ( CURRENT_CHAR , NOTUSED )
1851            ; VT : UP_ARROW ( CURRENT_CHAR , NOTUSED )
1852            ; NL : DOWN_ARROW ( CURRENT_CHAR , NOTUSED )
1853            ; CR : RETURN ( CURRENT_CHAR )
1854            ; 'I' , 'i'
1855              : BEGIN
1856                INSERT_MODE ( CURRENT_CHAR )
1857              ; EDIT_PROMPT ( DIRECTION_CHAR )
1858              END
1859            ; 'D' , 'd' : DELETE_MODE ( CURRENT_CHAR )
1860            ; 'X' , 'x'
1861              : BEGIN
1862                EXCHANGE_MODE ( CURRENT_CHAR )
1863              ; EDIT_PROMPT ( DIRECTION_CHAR )
1864              END
1865            ; 'Q' , 'q' : QUIT_MODE ( DONE , CURRENT_CHAR )
1866            ; 'V' , 'v'
1867              : BEGIN
1868                VERIFY ( CURRENT_CHAR )
1869              ; EDIT_PROMPT ( DIRECTION_CHAR )
1870              END
1871            ; 'P' , 'p' : MOVE_PAGE ( CURRENT_CHAR , DIRECTION_CHAR )
1872            ; '<' , '>' , ',' , '.'
1873              : DIRECTION ( COMMAND_CHAR , DIRECTION_CHAR )
1874            ; ' ' : SPACE ( CURRENT_CHAR , DIRECTION_CHAR )
1875            ; 'J' , 'j'
1876              : BEGIN
1877                JUMP_MODE ( CURRENT_CHAR )
1878              ; EDIT_PROMPT ( DIRECTION_CHAR )
1879              END
1880            ; 'F' , 'f'
1881              : BEGIN
1882                FIX_MODE ( CURRENT_CHAR )
1883              ; VERIFY ( CURRENT_CHAR )
1884              ; EDIT_PROMPT ( DIRECTION_CHAR )
1885              END
1886            ; '/' , '?' : SECOND_EDIT_PROMPT ( DIRECTION_CHAR )
1887            ; '1' : OUTPUT_REST_OF_LINE ( CURRENT_CHAR , J )
1888            ; '2' : MOVE_TO_END_OF_CURRENT_LINE ( CURRENT_CHAR , J )
```

```
1889            ; '3' : SPREAD_CURRENT_LINE ( CURRENT_CHAR , J )
1890            ; '4' : SPLIT_CURRENT_LINE ( CURRENT_CHAR )
1891            ; '5' : MOVE_TO_END_OF_TEXT ( CURRENT_CHAR )
1892            ; '6' : OUTPUT_REST_OF_SCREEN ( CURRENT_CHAR , J , N )
1893            ; '7' : MOVE_RIGHT_1_LINE ( CURRENT_CHAR , J )
1894            ; '8' : MOVE_LEFT_1_LINE ( CURRENT_CHAR , J )
1895            ; '9' : OUTPUT_CURRENT_LINE ( CURRENT_CHAR )
1896            ; ELSE : BEGIN END
1897            END
1898          END
1899
1900      ; PROCEDURE INITIALIZE_WITH_FILE
1901          ( FILE_LOGICAL_UNIT_NUMBER : INTEGER ; VAR CURRENT_CHAR : CHAR )
1902
1903        ; VAR NO_OF_PAGES : INTEGER
1904
1905        ; BEGIN
1906          ; IO . OUTPUT_LINE ( 'INITIALIZING$' )
1907          ; B_M . FIND_NO_OF_PAGES
1908              ( FILE_LOGICAL_UNIT_NUMBER , NO_OF_PAGES )
1909          ; B_M . INIT_RIGHT_STACK
1910              ( FILE_LOGICAL_UNIT_NUMBER , NO_OF_PAGES )
1911          ; CURRENT_CHAR := EM
1912          ; ADD_LINE_NUMBERS ( CURRENT_CHAR )
1913          END
1914
1915      ; PROCEDURE GO
1916
1917        ; VAR CURRENT_CHAR , COMMAND_CHAR : CHAR
1918          ; DONE : BOOLEAN
1919
1920        ; BEGIN
1921            CLEAR_SCREEN
1922          ; IO . OUTPUT_LINE ( 'SEDIT VERSION 1(:10:)(:10:)(:13:)$' )
1923          ; INITIALIZE_WITH_FILE ( 1 , CURRENT_CHAR )
1924          ; VERIFY ( CURRENT_CHAR )
1925          ; DIRECTION_CHAR := '>'
1926          ; EDIT_PROMPT ( DIRECTION_CHAR )
1927          ; DONE := FALSE
1928          ; REPEAT
1929    "ACCEPT_COMMAND"
1930              IO . INPUT ( COMMAND_CHAR )
1931            ; EXECUTE_COMMAND
1932                ( CURRENT_CHAR , COMMAND_CHAR , DIRECTION_CHAR , DONE )
1933            UNTIL DONE
1934          END
1935
1936      ; BEGIN
1937          INIT IO , B_M ( IO ) , S7 , C_B_M
1938        ; SHOW_LINE_NUMBERS := TRUE
1939        ; SECOND_EDIT_PROMPT_ON := FALSE
1940        ; GO
1941        END
1942    .
```

A FULL SCREEN EDITOR IMPLEMENTED IN PASCAL

by

THEODORE JOHN SOCOLOFSKY

B.S., KANSAS STATE UNIVERSITY, 1980

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1981

ABSTRACT

A Full Screen Editor has been built. This editor provides a CRT terminal user an efficient means of editing information displayed on the screen. This editor, in contrast to most editors, provides the user a direct visual image of the contents of a text file, and a means to move, correct, change and manipulate displayed text files. Accordingly, full screen editors are much more efficent and effective then other usual types of editors. The editor was written in PASCAL language which was choosen because of its portability to other machines and operating systems.