6800/6809 EDITOR

V 1.1

USER'S MANUAL

SOFTWARE DYNAMICS

2111 W. Crescent, Suite G, Anaheim, CA 92801

E D I T

U S E R ' S

M A N U A L

EDIT V1.1

5th Printing

NOTICE

This manual describes EDIT Version 1.1. Software Dynamics has carefully checked the information given in this manual, and it is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Software Dynamics reserves the right to change the specifications without notice.

EDIT V1.1 is an improved version of EDIT V1.Ø, and has been optimized for use under SDOS. These optimizations prevent use of EDIT V1.1 in non-SDOS environments.

TABLE OF CONTENTS

EDITOR FEATURES

EDIT is a general purpose context editor, used to enter and correct ASCII text files.

Significant features include:

-- Two input files, each re-selectable without leaving EDIT

-- One output file, re-selectable without leaving EDIT

-- Ability to merge or split files

-- Multiple commands allowed on command lines

-- Commands for both line-oriented and character-oriented editing

-- Easy-to-use search, change, and remove commands

-- Automatic typing of a line after a search, change, or remove

-- Value registers

-- Easy insertion and deletion of text

-- Display commands

-- Command input can be taken from a file, allowing canned editing procedures (macros)

-- Iteration commands, including testing and branching

-- Arithmetic, logicals, and relationals (for iteration control)

-- Command success/fail testing (for iteration control)

INTRODUCTORY CONCEPTS

EDIT is a general purpose context editor. "Context" is the part of a word, sentence, or paragraph that occurs just before and/or after a specified character, word, or phrase. "Editing" is the preparation, arrangement, revision, or correction of a document.

The editor is used to change (or create) a text file (a file is generally defined to be any collection of data, usually on a disk; for our purposes, only files containing textual information are important). With the exception of deleting ASCII nulls, EDIT does not restrict the kind of text data that may be edited (local operating system conventions may place additional restrictions, however).

Besides holding text, files may be used to hold frequently used command sequences, or for moving and/or duplicating blocks of text from the text buffer.

The editor treats the text to be processed as a stream of characters, which can be read and edited in chunks. Once a chunk is processed, it cannot be changed or examined without re-processing the entire text file again.

EDIT uses the computer's memory as a "workspace" to perform the required editing. The workspace is used for two purposes: storing text while editing and as a place to put editing commands typed by the user. The portion of the workspace used for holding text while it is being edited is called the "text buffer" and the portion used for holding commands being typed in is called the "command buffer". Although the workspace has a fixed size determined by the particular system configuration, the text buffer and command buffer may dynamically grow and shrink during the editing process.

The editor fills its text buffer on a page by page basis. A "page" is that portion of a text file up to and including a form character (^L). Therefore form characters are generally used to make the amount of text transferred into the text buffer at any one time more manageable. Forms should generally be placed to make the number of characters in any one page less than half the size of the workspace. Multiple pages may be present in the text buffer as long as enough memory is available to hold them.

The editing process operates as follows:

1.) The user invokes the editor.
2.) A file containing text is selected to be modified (Source file) (ER or EB command).
3.) A new file is created which will eventually contain the modified version of the file selected in Step 2 (Destination file) (EW command).
4.) The first page of text from the source file is copied into the text buffer (1A command).
5.) The user interacts with the editor to modify the contents of the text buffer until he is satisfied with the results (Insert, Change, Type, Search commands).
6.) The modified page is written to the destination file (P command).
7.) The next source file page is read into the text buffer, modified, etc.
8.) Having changed all parts of the text that needed changing, the user exits the Editor (EXIT command).

The user tells the editor what to do next by typing commands.

Many EDIT operations are invoked by a single character command mnemonic: "T" means Type one line; "I" means Insert, etc. Some commands allow or require values before them to specify a count of how many times the command should be done: "5T" means Type five lines; "2D" means Delete two characters, etc. If a value is not supplied, it is generally assumed to be "1". A "-" (minus) sign preceding a command causes the value to be assumed to be -1. Some commands require a string argument (arbitrary sequence of characters) followed by a delimiter (end-of-string indicator): "SHELLO\" means Search for the occurrence of "HELLO" (\ is the delimiter); "CABC\XYZ\" means Change "ABC" to "XYZ", 3FHI\ means Find the third occurrence of "HI".

Some of the editor commands are spelled with two characters. There are two reasons for this:

1.) Many editor commands are very powerful and can destroy portions of a file if used carelessly or accidentally. In addition to the two character format, "dangerous" commands also require a value (usually a single digit) preceding them. This 3 character format helps ensure that the user really intended to use the command.

2.) It is difficult to choose unique single character command names that are mnemonic given only 26 alphabetical characters.

When the editor is ready to accept command input, it will signal ready with the prompt character "*". Commands that are typed into the editor are placed into the command buffer and "saved-up" until the activation character (carriage-return) is received. If the command requires more input, the editor will prompt for more input before executing the command. At this point, the editor will execute (perform) all the specified operations. When the editor finishes the required operations, it will again signal ready by displaying the prompt.

When typing in a command, uppercase or lowercase commands may be used interchangeably. String arguments are used exactly as typed. Consequently, "Shello\" is not equivalent to "SHELLO\", while "sHELLO\" is the same as "SHELLO\".

Line input editing (correcting typing mistakes while entering a command string) is subject to local system conventions. Usually, a Delete (or Rubout) character can be typed to erase an erroneous character. The SDOS virtual terminal driver will allow editing within a typed-in line before EDIT sees the line; refer to the SDOS manual for more details.

One or more EDIT commands may be typed followed by a carriage-return. This series of commands is referred to as a command string. The command string is not restricted to one line of input as it may include carriage-returns in string arguments. The last character in a command string is a carriage-return which is not part of a string argument; this carriage return is called the activation character. A carriage-return appearing in a string argument is considered to be part of the string and is not an activation character. The editor saves up the command string in the command buffer until the activation character is typed. The commands are then executed left to right until no more commands remain.

When the editor discovers an error, such as incorrect command syntax, or a command cannot be performed, it will print an error code which can be found in the error summary.

Editor commands that manipulate and display text do their work on text in the text buffer. Since there is usually a lot of text in the buffer, some mechanism is needed to focus the editor's attention on a particular part of the text buffer. This focusing is done via a special pointer called the "CP" (character pointer). CP always points to a particular character in the text buffer, which is where the actual editing is performed. The CP and the text buffer together are very like a pencil and some paper with printing on it; before one can change the printing on the paper, one must first place the pencil on the word to be changed. Commands exist to allow the user to move the CP around in the text buffer in several different ways. The location of CP is generally termed "the context"; thus the name of the editor's style.

For the purpose of explanations involving the CP, the text buffer could be considered a linear arrangement of all of the characters currently in the text buffer. For example, suppose the text buffer contained the following text and the CP was positioned on the "F":

```
        ABC
        DEF
        GHI
        JKL
```

This could be represented linearly as follows:

Beginning of text buffer

```
        A
        B
        C
        <CR>
        D
        E                    (BACKWARD, OR PREVIOUS TO CP)
        F       <-- CP
        <CR>
        G                    (FORWARD FROM CP)
        H
        I
        <CR>
        J
        K
        L
        <CR>
```

End of text buffer

In this case, "ABC" and "DE" are considered to be "before" the CP while "GHI" and "JKL" are considered to be after the CP. We say that the CP "points" to the "F" character. Also, CP is considered to be "between" "DE" and "GHI". Note that the end of a line has an explicit marker representing it in the buffer; this marker is called a "carriage-return" <CR> character. The CP may point to a <CR> character.

THE DISPLAY SCREEN VERSUS THE TEXT BUFFER

Many who are new to the idea of computers and editing may tend to think in terms of the familiar typewriter concepts:

> "I can't see it, therefore, it is not there."
>
> "That line that just rolled off the screen disappeared for good."
>
> "The top line (or bottom line) of the screen is the most interesting line; since I work there, the editor must work there."

Often, what is displayed on the screen is confused with what is in the text buffer. Actually, what is on the screen is only a copy of the part of the text buffer that the editor was asked to show the user, like a "window". The computer can be asked to display (using the "T" command) as little as one line or as much as the entire text buffer -- but only the last screen full of displayed text is available to be viewed. This is like a small window through which only a small portion of the text buffer can be seen at any one time. Even though lines will disappear off the top of the screen while scrolling, the lines never disappear from the text buffer until the computer is explicitly instructed to make them do so.

Note that the editor's text buffer may grow to contain more text than can fit on the computer's display device all at one time. For example, imagine the text buffer currently contains 5ØØ lines of text. If

```
B1ØØT
```

is typed into the computer, the editor will move the CP to the beginning of the buffer and type the first 1ØØ lines, most of which will scroll off the top of the screen, leaving only the last 23 (depending upon the size of the display) lines in full view. Note that the CP is still at the beginning of the buffer even though the line containing the CP was scrolled off the top of the screen, and all 5ØØ lines of text still remain in the text buffer.

COMMONLY USED COMMANDS

Probably the most frequently used commands are:

A	Append from input file
B	Begin (move CP to beginning of buffer)
C	Change one string to another string
D	Delete character
EB	Edit a file and make a backup copy of the old one
ER	Edit Read (select input file)
EW	Edit Write (select output file)
EXIT	Copy current buffer and the remainder of the input file (if any) to the output file and exit the editor.
I	Insert
K	Kill (delete) lines
L	Move CP in units of lines
M	Move CP in units of characters
P	Punch current buffer to output file
R	Remove a string
S	Search for a string
T	Type (display) line(s)
U	Delete Until (up to) a matched string
Z	Move CP to the end of the buffer
<CR>	Move CP to beginning of next line. Equivalent to 1L1T

These commands will handle all of the needs of the casual user and will be described further in this section. A complete reference for all the commands appears in the detailed command description section.

Of these commands, B(egin), C(hange) and T(ype) are sufficient to make virtually all changes to text in the buffer, and are worth remembering for the novice user.

A Append

This command is used to bring a portion of the input file into the editor's text buffer where editing may take place. The editor will read text from the input file until a control-L (page mark) is encountered, end of input file is encountered, or the text buffer becomes full. For example:

```
A
```

will append the next page of the input file to the end of the text buffer. An End of File message will be printed when the last page of the input file is appended; this is not an error.

B Begin

This command is used to move the CP to the beginning of the text buffer. The form is simply:

```
B
```

C Change

The C command is used to change the occurrence of one string to another string. For example,

```
CHELLO\GOODBYE\
```

will search for and then change the next (starting from CP) occurrence of "HELLO" to "GOODBYE" and leave the CP pointing to the first character after the replacement string. Note that strings (in general) may contain a carriage-return (CR) character.

As with all of the editor commands, the editing operation is performed relative to the CP. This means that if there were two occurrences of "HELLO", and the CP was positioned before the first "HELLO", the first "HELLO" would be changed. If the CP was positioned after the first "HELLO", the second "HELLO" would be changed.

To make the editor search backwards from the CP, simply type a minus (-) before the change command. For example:

```
-CHELLO\GOODBYE\
```

In this case, if the CP were positioned after the first "HELLO", the editor would search backwards from the CP for the match, discovering the first "HELLO", and change it to "GOODBYE".

If the CP were at the beginning of the text buffer, we could tell the editor to change the second occurrence of "HELLO" to "GOODBYE" by typing:

```
        2CHELLO\GOODBYE\
```

This will force the editor to change the second occurrence of "HELLO" and leave the first occurrence alone.

If Change indicates that it cannot find the desired string then perhaps that string is before CP and a backward ("-") change will find it.

Automatic display of the current line (the one containing the CP) will happen if a Change, Delete, Remove, or Search command is the last command in the command string so the user can easily verify that the changes actually made are the desired ones.

D   Delete Character

This command is used to remove the character(s) pointed to by CP. This command must have a value. The command:

```
        5D
```

would delete five characters starting with the CP and leave the CP pointing to the next character after the deleted sequence.

EB   Edit with Backup

This command selects a file to be edited. It is intended for use when one wishes to make changes to a file which already exists. It tells the editor to keep a backup copy of the original file, so if a disaster occurs, recovery of the original text is easy. It acts similar to a combination of ER and EW.

```
        EBMYFILE.TXT\
```

tells the editor to get input (ER) from MYFILE.TXT for editing, and to output (EW) edited text to a temporary file. When an EXIT is performed, MYFILE.TXT is made into a backup file by renaming it to MYFILE.BAK, and the resulting temporary output file is then renamed MYFILE.TXT. If EB is used, ER and EW should not be used.

ER Edit Read

This command is used to select an input file for the editor. It should be used with EW when one wishes to make a similar copy of a file with some changes and leave the original file intact (ER and EW together).

```
        ERPAYROLL.BAS\
```

would instruct the editor to open the file named "PAYROLL.BAS" for editor input; this is required before commands to fetch text from the input file can be issued. This command is not required if the editor is being used solely to create a new file (see EW). Since the file name given to EDIT is passed to SDOS, it is subject to the rules and conventions of SDOS.

A null file name will cause the editor to close the input file. At this point, another input file may be selected:

```
        ER\ERINVENTORY.BAS\
```

EW Edit Write

This command is used to create an output file. It should be used when one wants to create a file for the first time (EW by itself), or when one wishes to make a similar copy of a file with some changes and leave the original file intact (ER and EW together). If an output file is already open, it will be closed and the new file will be selected. The editor uses a "CREATE" call to the operating system which may cause any file by the same name to be deleted before the new file is created. An example of the EW command is:

```
        EWPROG1.DOC\
```

Use ER and EW as follows to make a modified version of JOESCONTRACT for SAM:

```
        ERJOESCONTRACT.TXT\EWSAMSCONTRACT.TXT\
```

EXIT

This command is used to terminate an editing session. When EXIT is typed, the current text buffer is copied to the output file and the remainder of the input file is also copied to the output file. If there is an input file selected or there is text in the text buffer, there must be an output file selected.

If no output is desired, use "ER\" to close the input file (only if it is open) and use "EZ" to clear the text buffer (required only if the text buffer is not empty):

```
        ER\EZEXIT
```

I Insert

This command is used to insert text immediately BEFORE the CP. For example, let's say that the CP was positioned on the first "E" of the example "THIS IS A EXAMPLE". The command

```
IGOOD \
```

would insert the characters "GOOD " after "THIS IS A " and before "EXAMPLE" producing "THIS IS A GOOD EXAMPLE".

To insert many lines of text, simply type "I" followed by the desired text which is then followed by the delimiter "\":

```
ILine 1
Line 2
Line 3
.
.
.
Line n
\
```

K Kill Line

Kill is used to delete ("kill") lines of text near CP.

The argument specifies how many lines to "kill", starting from CP. If the argument is negative, lines are deleted before CP. ØK kills the part of the line to the left of CP. If CP is at the beginning of the line,

```
ØK
```

will do nothing.

```
1K
```

will remove one line, and

```
-2K
```

will remove the two previous lines.

L Lines (Move CP over lines)

This command is used to move the CP around in the text buffer in units of lines. For example,

```
    L
```

moves the CP forward to the beginning of the next line.

```
    ØL
```

moves the CP to the beginning of the current line.

```
    5L
```

moves the CP forward five lines.

```
    -5L
```

moves the CP back to the beginning of the fifth previous line.

M Move (CP over characters)

This command is used to move the CP forward or backward starting from the CP in units of characters.

```
    M
```

moves the CP forward 1 character.

```
    -5M
```

moves the CP backward 5 characters.

P Punch (text buffer to output file)

This command is used to move the contents of the text buffer to the output file (thus making room for the next page of text from the input file). This command requires a value. If the value is Ø, then the current text buffer is moved to the output file, and the buffer is left empty. If the value (n) is greater than Ø, then n-1 pages (counting the current text buffer as 1 page) are copied from the input to the output file, and the nth page is read into the text buffer.

```
    1P
```

moves the current text buffer to the output file, and fetches the next text buffer from the input file.

R   Remove a string

This command searches for the specified string and removes it from the text buffer. This command is exactly equivalent to the change command with a null replacement string. If the CP preceded the phrase "THIS IS A GOOD EXAMPLE", then

```
        RTHIS IS \
```

would leave "A GOOD EXAMPLE". The CP is left pointing to the first character past the removed string.

S   Search for a string

This command is used to search for the occurrence of a string in the text buffer and set the CP pointing to the first character after the matched occurrence. This command is very important in a context editor, because it is valuable in placing CP in the desired location for editing. By searching for a unique occurrence of some phrase, sequence of characters, special marker in the text, etc., the CP can be located very near the area of text to be edited.

Assume that a particular word was misspelled in several places and needed to be corrected. We can locate the misspelled word by using the Search command. For example, if the CP were at the top of the buffer, and we knew that AMOUNT was misspelled AMMOUNT somewhere in the buffer, the following command would find the mispelled version:

```
        SAMMOUNT\
```

Assume that a paragraph needed to be inserted in a section of a document labeled "Questions and Answers". The Search command could be used to locate the section by searching for "Questions and Answers", and by applying the Search command a few more times, the desired context can be located. If necessary, subsequent application of the "L" and "M" commands will pinpoint the exact context. Note that the "C" command includes a Search, which is very convenient for finding and changing a string.

T Type (display lines)

This command is used to make the editor print out lines of text (in the buffer near CP) on the console. This is particularly useful for examining changes just made to the text, displaying the current context, or for finding out exactly where the CP is. Note that this is a command that asks the editor to do some typing instead of the user.

```
T
```

will type from the CP inclusive, up to and including the next carriage-return.

```
-5T5T
```

will type out ten lines; the five lines before the CP and the five lines after the CP.

```
ØT
```

will type from the beginning of the line up to, but not including, the CP.

U Until (delete up to matched string)

This command is useful for deleting text from the current position of the CP up to, but not including, the matched string. For example,

```
1U<tab>\
```

will delete every character from the CP inclusive, up to but not including the first <tab> character. Warning: this can delete large chunks of text if you give the wrong string as an argument; use it carefully!

Z   Move CP to the end of the text buffer

Z is useful for moving the CP to the end of the text buffer to view the end of the text buffer,

```
       Z-23T
```

or to insert text at the end of the text buffer:

```
       ZILine 1
       Line 2
       Line 3
       .
       .
       .
       Line n
       \
```

<CR>   Moves CP forward 1 line and displays that line.

The <CR> command is exactly equivalent to "1L1T". It is handy for stepping through the text buffer one line at a time.

COMMONLY USED COMMAND SEQUENCES

Some examples of common types of command sequences are given below:

SEQUENCE	EXPLANATION
CNUTZ\NUTS\	Change "NUTZ" to "NUTS"
ØTT	Type the current line
ØLT	Position CP to the beginning of the Line and Type it
BS<CR> LABEL\-TT	Find the first occurence of "<Carriage-Return>Label" and type the previous line and the current line
23T	Type a screen-full of context
23L23T	Move CP forward 23 Lines then Type a screen-full of context
-1ØT1ØT	Type the previous 1Ø lines and the next 1Ø lines
2C.\,\	Change the second period to a comma (and type the line if Auto-Type is on)
ILOTSA<CR> WORDS\-2TT	Insertion. Inserts the text LOTSA<CR>WORDS at CP, and then types 3 lines around CP. Note that the entire inserted string is inserted before the CP.

Probably the most useful sequence of editing commands is "ØTT" which means type from the beginning of the line up to the CP (ØT part) and then type the remainder of the line from the CP (T part). The net effect is to type the current line, no matter where the CP is within the line, and without moving the CP. Since this command is so commonly used, EDIT provides for an automatic "ØTT" after every change, delete, search, remove or move CP command. "ØEA" will disable the auto-type feature. "1EA" will re-enable it.

Once the user is familiar with the commonly used commands, he should investigate the empty string search (see "S" command in Detailed Command descriptions section) and the X command, as they prove to be very useful.

TWO SIMPLE EDITING EXAMPLES

In the first example, a new file is created in the Editor. Note the "\" character appearing in the left margin. This is the editor's signal that it is expecting more of a string argument. Note also that a TAB character has been consistently used when moving from the left margin to the first word in a line; this is why the rlØ\ command below does not affect the spacing. Note the "12V" command in Example 1; this is the most common method of putting a "Form" character in the buffer. The ^L is printed by EDIT when it outputs a form.

Example 1:

```
.EDIT
EDIT V1.1s    Copyright (C) 1979 Software Dynamics
*ewprimes.bas\
*I1Ø   **** PRIME NUMBER CHASER ****
\      REM PRINTS OUT FIRST 1ØØ PRIME NUMBERS
\      DIM X, PRIMES (1ØØ), CANDIDATE/3/
\      DIM PRIMESELECTOR, NPRIMES/1/
\      PRINT "Prime Finder"
\      PRIMES(NPRIMES) = 2
\      X = PRIMES(PRIMESELECTOR)
\\
*-LT
       X = PRIMES(PRIMESELECTOR)
*i1ØØ FOR I = 1 TO NPRIMES
\\
*z-23t
1Ø     **** PRIME NUMBER CHASER ****
       REM PRINTS OUT FIRST 1ØØ PRIME NUMBERS
       DIM X, PRIMES (1ØØ), CANDIDATE/3/
       DIM PRIMESELECTOR, NPRIMES/1/
       PRINT "Prime Finder"
       PRIMES(NPRIMES) = 2
1ØØ    FOR I = 1 TO NPRIMES
       X = PRIMES(PRIMESELECTOR)
*br1Ø\miREM\
       REM**** PRIME NUMBER CHASER ****
*Øt
       REM*i \
       REM **** PRIME NUMBER CHASER ****
*zi    IF INT(CANDIDATE/X) * X = CANDIDATE THEN 18Ø
\      NEXT PRIMESELECTOR
\      ! FOUND A NEW PRIME
\      NPRIMES = NPRIMES + 1
\      PRIMES(NPRIMES) = CANDIDATE
\      CANDIDATE = CANDIDATE + 2
\      IF NPRIMES <> 1ØØ THEN 1ØØ
\      FOR PRIMESELECTOR = 1 TO 1ØØ
\      PRINT PRIMES(PRIMESELECTOR)
\      NEXT PRIMESELECTOR
\      PRINT ' All Done!'
```

```
\200  STOP
\     END
\\
*12V
^L*-23T
      REM **** PRIME NUMBER CHASER ****
      REM PRINTS OUT FIRST 100 PRIME NUMBERS
      DIM X, PRIMES (100), CANDIDATE/3/
      DIM PRIMESELECTOR, NPRIMES/1/
      PRINT "Prime Finder"
      PRIMES(NPRIMES) = 2
100   FOR I = 1 TO NPRIMES
      X = PRIMES(PRIMESELECTOR)
      IF INT(CANDIDATE/X) * X = CANDIDATE THEN 180
      NEXT PRIMESELECTOR
      ! FOUND A NEW PRIME
      NPRIMES = NPRIMES + 1
      PRIMES(NPRIMES) = CANDIDATE
      CANDIDATE = CANDIDATE + 2
      IF NPRIMES <> 100 THEN 100
      FOR PRIMESELECTOR = 1 TO 100
      PRINT PRIMES(PRIMESELECTOR)
      NEXT PRIMESELECTOR
      PRINT ' All Done!'
200   STOP
      END
^L*bsDATE + 2\0LT
      CANDIDATE = CANDIDATE + 2
*i180\
180   CANDIDATE = CANDIDATE + 2
*exit
```

In this example, an existing file (produced in Example 1) is edited. Lines which contain only an "*" are prompts printed by EDIT, with <CR> (which acts as 1L1T) entered by operator. "-EA" was done to show CP location.

Example 2:

```
.edit
EDIT V1.1s    Copyright (C) 1979 Software Dynamics
*ebprimes.bas\
*-ea
*1ab23t
      REM **** PRIME NUMBER CHASER ****
      REM PRINTS OUT FIRST 100 PRIME NUMBERS
      DIM X, PRIMES (100), CANDIDATE/3/
      DIM PRIMESELECTOR, NPRIMES/1/
      PRINT "Prime Finder"
      PRIMES(NPRIMES) = 2
100   FOR I = 1 TO NPRIMES
      X = PRIMES(PRIMESELECTOR)
      IF INT(CANDIDATE/X) * X = CANDIDATE THEN 180
      NEXT PRIMESELECTOR
      ! FOUND A NEW PRIME
      NPRIMES = NPRIMES + 1
      PRIMES(NPRIMES) = CANDIDATE
180   CANDIDATE = CANDIDATE + 2
      IF NPRIMES <> 100 THEN 100
      FOR PRIMESELECTOR = 1 TO 100
      PRINT PRIMES(PRIMESELECTOR)
      NEXT PRIMESELECTOR
      PRINT ' All Done!'
200   STOP
      END
^L*
      REM PRINTS OUT FIRST 100 PRIME NUMBERS
*
      DIM X, PRIMES (100), CANDIDATE/3/
*
      DIM PRIMESELECTOR, NPRIMES/1/
*
      PRINT "Prime Finder"
*
      PRIMES(NPRIMES) = 2
*
100   FOR I = 1 TO NPRIMES
*cI\PRIMESELECTOR\
100   FOR PRIMESELECTOR = 1 TO NPRIMES

                       ^

*
      X = PRIMES(PRIMESELECTOR)
*
      IF INT(CANDIDATE/X) * X = CANDIDATE THEN 180
*i    Q = INT(CANDIDATE/X)
```

```
\      IF Q < X THEN 13Ø
\\-2tt
       Q = INT(CANDIDATE/X)
       IF Q < X THEN 13Ø
       IF INT(CANDIDATE/X) * X = CANDIDATE THEN 18Ø
*sIF \lu \
       IF  * X = CANDIDATE THEN 18Ø

          ^

*Øt
       IF *iQ\
       IF Q * X = CANDIDATE THEN 18Ø

           ^

*
       NEXT PRIMESELECTOR
*
       ! FOUND A NEW PRIME
*i13Ø\
13Ø    ! FOUND A NEW PRIME

   ^

*z-23t
       REM **** PRIME NUMBER CHASER ****
       REM PRINTS OUT FIRST 1ØØ PRIME NUMBERS
       DIM X, PRIMES (1ØØ), CANDIDATE/3/
       DIM PRIMESELECTOR, NPRIMES/1/
       PRINT "Prime Finder"
       PRIMES(NPRIMES) = 2
1ØØ    FOR PRIMESELECTOR = 1 TO NPRIMES
       X = PRIMES(PRIMESELECTOR)
       Q = INT(CANDIDATE/X)
       IF Q < X THEN 13Ø
       IF Q * X = CANDIDATE THEN 18Ø
       NEXT PRIMESELECTOR
13Ø    ! FOUND A NEW PRIME
       NPRIMES = NPRIMES + 1
       PRIMES(NPRIMES) = CANDIDATE
18Ø    CANDIDATE = CANDIDATE + 2
       IF NPRIMES <> 1ØØ THEN 1ØØ
       FOR PRIMESELECTOR = 1 TO 1ØØ
       PRINT PRIMES(PRIMESELECTOR)
       NEXT PRIMESELECTOR
       PRINT ' All Done!'
2ØØ    STOP
       END
^L*exit

End Of File hit
.* (Back at SDOS Command Interpreter)
```

DETAILED COMMAND DESCRIPTIONS

This section gives detailed information on all EDITor commands. Editor commands have the following syntax:

```
<VALUE><COMMAND><STRING><DELIMITER><STRING><DELIMITER>
```

<VALUE> is an integer in the range of -32768 to +32767. The value may be further restricted by the individual command that uses it. The value may be supplied as a result of a previous command. Some commands do not allow a value to be supplied.

<COMMAND> is the editing command indicating what operation is to be performed. This is normally a single character; exceptions are the relational commands and the extended commands (E-type). Lower case commands are treated as the equivalent of upper case commands.

<STRING> is an arbitrary sequence of characters, not including the current delimiter or any characters which are given special treatment by the operating system (these are system dependent -- consult your operations manual). The string may include a carriage-return <CR>. A "null" string is one that has zero characters in it.

<DELIMITER> is a character chosen by the user to indicate end-of-string. EDIT initially assumes "\" (backslash) as the delimiter, but this may be changed by the G command.

Each command has some variation of the above syntax, which is indicated with the command description. If the command description indicates a particular format, then the command must be given to the editor exactly as specified, or a syntax error will result (exception: <VALUE> is usually optional, with a default, or the previous command may supply it).

There are ten value registers numbered Ø through 9. These registers are useful for remembering the location of the CP. They can also be used as counter registers or single character registers (since a character can be represented as a value) in the more sophisticated editing sequences.

A special "convenience" allows commands that search the text to search for a previously entered string, if the specified search string is null. See S command description.

In the command descriptions, the following notation is used:

"n" refers to a user or previous command specified value. If the user gives a value, it overrides the default; a value supplied by a previous command may be overridden with the blank command; if n is not indicated in the syntax, it must not be supplied.

"d" refers to a single digit (Ø-9).

\ (backslash) represents the current delimiter.

s1,s2 are strings, not including the current delimiter.

A line is a sequence of characters in the text buffer that is bounded by carriage-return <CR> characters. It includes the CR at the end, but not at the beginning. The current line is defined to be a line containing a character pointed to by CP.

Workspace refers to the total space available to the editor for all buffers, i.e., text buffer, command buffer, and text registers.

"Yank" means to make the buffer empty, and then to fill the buffer using the next page from the input file.

An iteration is a command sequence enclosed by a [ ] pair.

nA

Append. If n > Ø, the next n pages of the source file are appended to the text buffer at the end of the buffer. If n <= Ø, an error is given; the default value for n is 1. CP is positioned to the beginning of the first appended page. The only recoverable error is End of file hit.

B

Begin. Moves CP to the first character in the text buffer. This command is the same as "1J".

nCs1\s2\

Change. Causes the nth occurrence of s1 to be changed to s2. If n = Ø, an error is given. If n > Ø, a forward search is made. If n < Ø, a backwards search is made. The default value of n is 1. CP is left pointing to the first character after s2 in the buffer. See S command if s1 is null. The only recoverable error is "string not found". The change command is equivalent to nSs1\Xs2\.

nD

Delete. Deletes the next n characters relative to CP. If n = Ø, an error is given. If n > Ø, deletion is in the forward direction starting with and including CP. If n < Ø, deletion is in the reverse direction, but does not include CP. There is no default value for n. The only recoverable error is "Delete off the end of the buffer."

nEA

Auto-type. The auto-type flag controls the automatic typing of the current line. If n = Ø, the auto-type flag is reset. If n is 1 or -1, the flag is set. There is no default value for n. EDIT starts with 1EA. If the auto-type flag is set, commands which move the CP or make changes to the text buffer (see table below), will automatically type (ØTT) the current line after successful execution if that command is the last command in the command string. For example:

```
BSHELLO\.EV1
```

will not auto-type, while

```
BSHELLO\
```

will. If n=-1 then auto-type is enabled and the CP position will be indicated by a "^" on an auto type (exception: no caret will be displayed if CP is at ØL of a line). For example, if the buffer in the example above contained "SAY HELLO there.", then the session would appear as

```
*BSHELLO\
SAY HELLO there.
         ^

*
```

This setting is recommended for persons new to the concept of the CP. To find the CP position on the current line use "ØM".

Commands which auto-type:

A(ppend)	N(ext)
B(egin)	P(unch)
C(hange)	Q(uerysearch)
D(elete)	R(emove)
EY	S(earch)
F(ind)	U(ntil)
I(nsert partial line)	V(alueinsert)
J(ump)	W(rite)
K(ill)	X(change)
L(ine)	<tab>(insert partial line)
M(ove)	

EBsl\

Edit Backup. Edit file sl creating a backup file. The precise effect when executed is: ERsl\EWEDITOR.TMP\. When an EF, EXIT, or EW is executed after an EB, the original file sl is renamed with a .BAK extension; then EDITOR.TMP is renamed to sl. NOTE: if sl has a device designation in it, EDITOR.TMP will be created on that device (if possible). The only recoverable error using "?" is "No such file".

ECs1\

Edit copy. Copy the entire contents of file s1 into the text buffer before CP. If there is not enough room in the text buffer, then to make room, lines are output from the beginning of the buffer up to CP, until the file has been copied; the Editor will indicate this has happened by printing out "Text preceding CP flushed". The status of the input (ER) file is not affected. CP is left pointing after the text copied into the buffer. The only recoverable error ("?") is "No such file".

EF

Edit finish. Copy the text buffer to the output file, then copy the remaining portion of the input, if any, to the output file. If no output file is currently open, the input file is closed and the text buffer is cleared. If ERs1\ was used to open an input file, a close s1 is performed. If EWs2\ was used to open an output file, a close s2 is performed. The buffer is left empty. This is identical to EXIT except that control is returned to the EDITor.

EIs1\

Edit input. Causes editor to suspend accepting commands from the keyboard (specifically, channel Ø by SDOS conventions) and to accept commands from the file whose name is s1. When the file is exhausted, the editor will automatically switch back to accepting commands from the keyboard (channel Ø). This command is primarily used by DO files under SDOS which invoke the editor so that keyboard input can be used while the DO file is active, or to execute pre-canned sequences of editor commands stored in a file. The only recoverable error ("?") is "No Such File".

nEOs1\

Edit output. A file named s1 is created and the next n lines of the text buffer are output to the newly created file; then the file is closed. If n<Ø, the previous n lines are copied to the output file. If n=Ø, the current line is written from its beginning up to, but not including CP. If n>Ø, the next n lines are output, starting from CP. There is no default value for n. The output lines are NOT deleted from the text buffer. The only recoverable error is "Write off end of buffer." (i.e., n is larger than number of lines left between CP and end of buffer).

ERsl\

Edit read. Open file sl for input. If there is a currently open input file, close it, then open sl. If sl is the empty string, the input file is closed and no new input file is opened. The only recoverable error ("?") is "No such file".

ETt1,t2,...,tk

Set tabs. The tab stops are set to t1,t2,...,tk (column numbers). Tab stops are used for displaying tab characters. There are a maximum of 20 tab stops allowed on a (displayed) line. If there are unspecified tab stops, they are assigned values that are increments of 8 after the last specified tab stop. The default tab stops are 8, 16, 24, 32... This command must be the last command on a command line.

EUd

Use value. The value of register d is returned as a value for the next EDIT command. NOTE: EU0 returns the negative of the size of last successful search target string (negative if "S", "N", "Q"; positive if "F") or insert if CP has not been moved since; otherwise, it returns 0. Note: d must be a digit 0-9.

nEVd

Store value. The value n is stored into value register d. This command is intended for use with EUd. There is no default value for n. NOTE: d may only be 1-9.

EWsl\

Edit write. If an output file is currently open, close it. If file sl does not exist, it is created for use as the output file. EW\ closes the output file.

EXIT

Exit. Perform an EF, then exit to SDOS command interpreter.

nEY

Yank. Clear the buffer and read in n pages. If n <= 0, an error is given. If n > 0, the command is identical to EZnA. There is no default value for n. CP is left pointing to the first character in the buffer. The only recoverable error is "End of file hit."

EZ

Edit zap. An E-Z way to delete the entire contents of the text buffer.

nFsl\

Find. Starting with CP, search the text buffer for the nth occurrence of sl in the buffer. CP is left pointing to the first character of sl in the buffer. If n = Ø, an error is given. If n < Ø, a backward search is made. If n > Ø, a forward search is made. The default value for n is 1. See S command if sl is null. The only recoverable error is "String not found."

Gchar

Get delimiter. Char is taken as the new delimiter. Char must be a valid printing character, and must not be a letter. The G command must be the last command on a line, and must be followed by a <CR>. For example:

```
*G\
```

H

Returns the number of free bytes in the editor's workspace as a value.

nIsl\

Insert. Place sl into the text buffer just in front of CP. CP remains pointing to the character it was originally pointing to, i.e., after the inserted string. If n <= Ø, an error is given. If n > 1, the insert is performed n times. n defaults to 1.

nJ

Jump. Moves the CP to the nth line in the buffer. If n <= Ø, an error is given. n = 1 does the same as a "B" command. n > Ø, does the same as a "BnL" command sequence. There is no default value for n. The only recoverable error is "Jump off end of buffer."

nK

Kill. Causes the next n lines, starting with CP to be deleted. If n < Ø, deletion is done backwards, starting with the character before the CP. If n = Ø, the current line is deleted from its beginning up to, but not including, the CP. If n > Ø, the next n lines are deleted starting from the CP. There is no default value for n. The only recoverable error is "Kill off end of buffer."

nL

Line. Moves the CP forward across n CR characters. If n < Ø, the CP is moved backwards. If n = Ø, the CP is moved to the beginning of the current line. If n > Ø, the CP is moved forward. The default value for n is 1. The only recoverable error is "Move CP off end of buffer."

nM

Move. Moves the CP n characters. If n < Ø, CP is moved backwards. If n = Ø, CP is not moved. If n > Ø, CP is moved forward. n defaults to 1. The only recoverable error is "Move CP off end of buffer."

nNs1\

Search for next occurrence. Starting with the CP, searches forward for s1 in the buffer. If found, n is decremented; if n goes to zero, the operation is complete. Otherwise, the search is repeated starting from the end of the last occurrence of the string. If s1 is not found in the buffer, then the current page is punched (1P), and the operation continues. If n <= Ø, then an error is given. There is no default value for n. Leaves CP following s1 in the buffer. See S command if s1 is null. The only recoverable error is "String not found." Note: the N command can generally be used in place of S; this has the advantage of making buffer boundaries mostly invisible.

nP

Punch. The current contents of buffer is output, and then the buffer is cleared. If n = Ø, nothing else is done; if n >= 1, then n-1 pages are copied from the input to the output file, and the next input page is yanked. There is no default value for n. The only recoverable error is "End of file hit."

nQs1\

Query. Like the N command, but does not output any pages (does a 1EY instead of a 1P). See S command if s1 is null. The only recoverable error is "string not found."

nRsl\

Remove. Deletes the nth occurrence of sl and leaves CP pointing to the first character following the deleted occurrence of sl. If n = Ø, an error is given. If n < Ø, a backward search is made. If n > Ø, a forward search is made. The default value for n is 1. See S for null Sl. The only recoverable error is "String not found."

nSsl\

Search. Searches the text buffer for the nth occurrence of sl and leaves the CP pointing to the first character following sl in the buffer. If n = Ø, an error is given. If n < Ø, a backward search is made. If n > Ø, a forward search is made. The default value for n is 1. A null search string (i.e., s\) will default the search target to the previous search string (size limited to 25Ø characters). If the search for a default string fails, the target string is displayed, enclosed by the current delimiter. The only recoverable error is "String not found."

nT

Type. Types the next n lines starting with the CP. If n < Ø, the previous n lines are typed. If n = Ø, the current line is typed from its beginning up to, but not including the CP. If n > Ø, then the next n lines are typed. The default value for n is 1. 1T types from CP to the end of the line containing CP, and is useful for determining where the editor left CP. ØTT will type the entire line containing CP. The only recoverable error is "Type off end of buffer."

nUsl\

Until. Deletes all characters from CP to the nth occurrence of sl, but not including any part of the nth occurrence of sl. If n = Ø, an error is given. Backward deletion occurs (not including the character under CP) if n < Ø. Forward deletion occurs (including CP) if n > Ø. Leaves the CP pointing to the first character following the deleted characters. If sl is not found, nothing is deleted. There is no default value for n. See S command if sl is null. The only recoverable error is "String not found."

nV

Value insert. Convert n to its Ascii character equivalent and insert it in front of the CP. If n < Ø, or n > 127, an error is given. There is no default value for n. For example, to insert a FORM character at CP, use "12V".

nW

Write. Writes lines to the output file. If n < Ø, the previous n lines are copied to the output file. If n = Ø, the current line is written from its beginning up to, but not including CP. If n > Ø, the next n lines are output, starting from CP. There is no default value for n. The output lines are deleted from the text buffer. The only recoverable error is "Write off end of buffer."

Xsl\

Exchange. This command only works if the CP has not been moved since the last I, C, S, N, Q, F, or X command (these commands set EUØ to the size of the string inserted or replaced). This command exchanges sl for the last string inserted, changed, searched for or exchanged. Register Ø is used to control the exchange. If EUØ yields Ø, X will not function. If EUØ is non-zero the effect of the X instruction is: EUØDIsl\.

nYsl\

Verify string. In n = 1, sl is compared to the string pointed to by CP. If n = -1, sl is compared to the string before CP. n <> 1 or n <> -1 is illegal. If sl matches, the value 1 is returned, else returns Ø. The default value for n is 1. See S command if sl is null.

Z

Zip to end of buffer. Moves the CP to the end of the text buffer (beyond the last character).

^Isl\

Tab. Like I, but repetition cannot occur, and sl is placed in the buffer preceded by a tab character. CP is left pointing to the first character after the inserted string.

n[

Repeat. Do all commands inside [ ] n times. If n < Ø, an error is given. If n = Ø, commands inside the [ ] are skipped over. If n is not given, the iteration is repeated forever (exit is normally made via ^ or]). The escape key can be used to abort the iteration.

n]

Continue. If n = Ø, command interpretation will exit the [ ] pair. If n <> Ø, decrement the iteration counter and continue interpreting commands in the iteration from the matching [if the counter > Ø. If the iteration count goes to zero, exit the [ ] pair and continue execution with the command to the right of the]. The default value for n is 1.

?

Status. This command must immediately follow a command. A value followed by a "?" is not allowed. Yields Ø if the previous command failed, else yields 1. This value can be used as a value in an expression for the next command.

n^

Abort. Command execution will exit the [ ] pair if n = Ø. If n <> Ø, ^ is a no-op. There is no default value for n.

n$

Complement. If n = Ø, the result is 1, otherwise Ø. This is used to complement the state of relationals, status, etc. There is no default value for n.

#d

Label. Used as a target of conditional branches. If d = Ø, an error is given.

n_

Branch. If n < Ø or n > 9, an error is given. If n = Ø, no action is taken. The editor will continue command interpretation at label n if n > Ø and n <= 9 (i.e., start execution immediately following a #d command). For example:

```
        [CHELLO\GOODBYE\?^ @='.*1_ØTT] IMOM\ #1
```

There is no default value for n. Branching into an iteration is not allowed, but branching out of one is legal.

'char

Value. Yields the value of the character following the single quote. Char must be a printing character.

n;

Continue. If n = Ø, no action is taken. If n <> Ø, the editor decrements the iteration counter, and continues the iteration from the innermost enclosing [if the counter > Ø, else command interpretation exits the innermost enclosing [ ] pair (similar to n] command). There is no default value for n.

%

Iteration count value. Yields the number of executed iterations (i.e., on the first iteration, % yields 1, on the second iteration, % yields 2, etc.

@

At. Yields the numeric value of the Ascii character that CP points to. Only if CP is at the end of the buffer will @ yield 255. A null in the buffer (rare case!) will cause @ to return zero.

n=
n<
n<=
n>=
n>
n<>

Relationals. Each relational must be followed by a value or a value generating command. Yields Ø if the relation is false, 1 if the relation is true. There is no default value for n.

n+
n-

Add and subtract. The default value for n is Ø.

n*
n/
n&
n!

Multiply, divide, and, or. & and ! are bitwise binary operations. For example, 5!11 = 15, 5&11 = 1. There is no default value for n.

.

Period. The value representing CP's position (counted in characters) with respect to the beginning of the text buffer. When "." has the value zero, the CP is at the beginning (B) of the buffer. B23M gives the value "23" to "." if there are more than 23 characters in the buffer.

:

Colon. The value representing the current line number with respect to the beginning of the text buffer. B15L gives ":" the value "15" (if there are more than 15 lines in the buffer).

( )

Precedence change. Editor expressions are normally evaluated with operator precedence determining the order of operations ("algebraically") unless ( ) overrides this. Operator precedence is as follows:

```
* / &   are evaluated first
+ - !   are evaluated next.
Relationals are evaluated last.
```

n<CR>

Print value. If an explicit value precedes the CR, then that value is printed. If n is not supplied, nothing is typed. If a command line consists merely of <CR>, it is treated as if 1L1T had been typed.

n<BLANK>

Eat value. The value is consumed. This command is useful for constructing command sequences that consist of a command that explicitly cannot have a value preceded by a command that yields a value; this can be done by separating the two commands with the blank command. If n is not supplied, the blank command is a no-op. This command can be used to increase readability of macros.

COMMAND STRING PROCESSING

EDIT collects an entire command string before executing any commands. A command string is a series of commands ended by a CR character (a CR embedded in a string argument does not "end" the command string). Once a command string has been entered, the commands are executed from left to right until no more commands remain. Note that looping may occur due to the _ or] commands.

Typed input to EDIT is subject to the operating system conventions for line editing on console input.

Command interpretation may be stopped by using the ESCape key. This key is checked upon encountering a ;,], yank, punch, type, extension command (E-type), or during console input. A message is given when the ESCape key is used.

Command interpretation is normally left-to-right, with changes in order as directed by [ ], ^, ;, and commands.

Any command exhausting workspace room will cause command interpretation to terminate.

ERROR HANDLING

EDITor commands complete in one of 3 ways:

Successful termination
Recoverable error
Abortive error

If an editor command can be performed exactly as requested by the user, the command terminates successfully, and the editor then proceeds to execute the next command.

If a command fails for an unusual reason, then the editor will abort execution of the remainder of the command string, print the line (containing the error) of the command string, a pointer, and an error message describing the problem.

Example:

```
*ER%\
ER%\
^
Filename doesn't begin with A-Z or $
```

Some commands fail for common reasons, e.g., the search command might not find the desired string. Automatic error recovery may be built into macros for handling the most common reasons. The success or (recoverable) failure of a command may be tested by use of a trailing "?" (status) following the command; this ? Will produce a value of 1 for success or Ø for failure. This value may be used by a following command to effect conditional branching or looping.

Example:

```
       sABC\?_ sDEF\#1xXYZ\
```

changes the first occurence of ABC or DEF to XYZ.

Each command that can fail in a recoverable fashion has only one recoverable condition. All other failure conditions are treated as abortive, i.e., a Disk Read Error is a non-recoverable error condition for an N command.

In the case of abortive command termination, the EDITor will allow the user to save (insert) the entire command string (delimiters, commands, text and all) in the buffer at the current CP if the command string is more than 4Ø characters long. This is a useful "do what I mean" error recovery in cases where the user forgot to precede a large chunk of text by an I, and many other circumstances.

Example:

```
.edit

EDIT V1.lq   Copyright (C) 1979 Software Dynamics
*ewdl:lamb\
*Once upon time, there was a little lamb
\who loved to go to school with his
\master.  Everybody knew the lamb was
\gentle and kind.
\\

Once upon time, there was a little lamb
^
Illegal character
Did you really want to insert that whole command string? y
*exit
```

A response of anything not starting with the letter N saves the offending command; any other response throws it away.

If an error (other than 216 [Can't Find String] or 226 [End of File prior to "A" or "EY"]) occurs while the EDITor is running under a DO file and is not operating in EI mode, then at EXIT time, the EDITor will report the existence of that error to SDOS.

VALUES RETURNED FROM COMMANDS

The following commands return explicit values:

```
        @ : * / & + - = < <= !
        >= > <> % $ ? EUd nYsl\ .
```

The following commands return implicit values (the implicit value can be converted to an explicit value by use of a trailing "?" command). These commands either succeed or they give errors and stop. If they are followed by the "?" command, and no error occurs during command execution, a 1 is returned; if a recoverable error occurs, a Ø is returned; and if a non-recoverable error occurs, the command string is aborted.

```
        A C D EB EC ER EY F N P Q R S U
```

The following commands also return implicit values; these commands either succeed, or they fail in a recoverable way. However, if they fail, they never give errors. If they are followed by the "?" command, a 1 or Ø is generated if the command succeeded or failed respectively. These commands all share the property that a recoverable failure has to do with an operation that runs off the end of the buffer, and are friendly about it for the user's convenience.

```
        EO K J L M T W
```

All the remaining commands either succeed or abort and do not generate success/fail values. However, if they are successful and are followed by the "?" command, a 1 is generated.

All the commands that return explicit values can be used to generate values for the following commands that use or require them. For example, ".EV1" would remember CP in register 1, and "BEU1M" would restore CP.

Using the implicit value commands require that they be followed by the "?" command to generate values. For example:

```
        #1 SHELLO\?_
```

is a loop that searches for "HELLO" until it is not found, but

```
        #1 SHELLO\_
```

is an error; the implicit value generated as a result of the success or failure of the search must be made explicit by the "?" command.

REPEATING COMMAND SEQUENCES

Command sequences may be repeated by enclosing the sequence in a [ ] pair. Such loops are particularly useful for performing repetitive editing functions on the text buffer. For example, let's assume that we wanted to change the first 1ØØ occurrences of "LDAA" to "STAA". We could say:

```
        1ØØ[CLDAA\STAA\]
```

If we wanted to change all occurrences, we could say:

```
        [CLDAA\STAA\?]
```

The "?" command used here allows the iteration to cease when "LDAA" is not found. Note that [ ] by itself will keep the editor very busy doing nothing (use the ESCape key to stop).

Any number of commands may be contained in a command string within the [ ] pair which allows for very powerful commands. Below is a example of a command sequence which will change all lines containing a "jmp" to a "bra" and would type each line in which a change took place.

```
        [CJMP\BRA\?^ØTT]
```

A sequence of commands in a loop can be terminated by striking the ESCape key at any time during execution.

SHORT SOPHISTICATED EDITING EXAMPLES

1.) Edit file "TEST1", change "NUTZ" to "NUTS", and quit:

```
*ERTEST1\EWTEST1\1NNUTZ\XNUTS\EXIT
```

2.) Edit file "PROGRAM5", insert "95 INPUT 'ZIP: ' ZIP$" after line 90, verify the insertion, and quit:

```
*ERPROGRAM5\EWPROGRAM6\1N90\
End Of File hit
90 WRITE #SCREEN, ZIPX, ZIPY
*
100 GOSUB 2900 \ ! GO PUT THE ZIPCODE INTO THE FILE
*I95 INPUT 'ZIP: ' ZIP$
\\-2TT
90 WRITE #SCREEN, ZIPX, ZIPY
95 INPUT 'ZIP: ' ZIP$
100 GOSUB 2900 \ ! GO PUT THE ZIPCODE INTO THE FILE
*EXIT
```

3.) Edit file "PROGMAN", and change all the mispelled versions of "AMMOUNT" to "AMOUNT" and show each line; then quit. "PROGMAN" is a large file so we will use the N command.

```
*ERPROGMAN\EWPROGMAN\[1NAMMOUNT\?^XAMOUNT\0TT]EXIT
```

4.) Create a new file "JUNK", insert "PERFORM AN INJUSTICE TO MANKIND", and quit. The typist made mistakes, so corrective measures were used.

```
*EWJUNK\
*IPERFRM A JUSTIXE\0T
PERFRM A JUSTIXE*ITO MANKIND\0LT
PERFRM A JUSTIXETO MANKIND*CRM\ORM\
PERFORM A JUSTIXETO MANKIND*ZI
\\-LSA\0T
PERFORM A*IN\M0T
PERFORM AN *IIN\CX\C\
PERFORM AN INJUSTICETO MANKIND
*CTO\ TO\
PERFORM AN INJUSTICE TO MANKIND
*EXIT
```

5.) Merge file "TEMP1" onto the end of "TEST".

```
*ERTEST\EWTEST\[1A?^1P]ECTEMP1\EXIT
```

ERROR SUMMARY

2ØØ	-	Syntax Error
2Ø1	-	Can't find branch target
2Ø2	-	Can't find "]"
2Ø3	-	Incorrect Bracket Nesting
2Ø4	-	*** EDITor error ***
2Ø5	-	Illegal argument for command
2Ø6	-	Zero is not a valid argument
2Ø7	-	command requires argument
2Ø8	-	Command doesn't want an argument
2Ø9	-	No such "E" command
21Ø	-	Illegal character
211	-	Can't use that as delimiter character
212	-	Too many)s
213	-	Too many (s
214	-	Xchange not valid, must do search or insert first
215	-	Command not allowed while doing edit with EB
216	-	Can't find string
217	-	Q register index must be 1 to 9
218	-	Need to open input file first
219	-	Text Buffer is full
22Ø	-	Command buffer is full
221	-	Don't have enough lines in buffer to J that far
222	-	Illegal tab stop list
223	-	Need to select output file first
224	-	Unbalanced [ ]'s
225	-	Bracket Stack overflow or underflow
226	-	End of File prior to "A" or "EY"
227	-	Buffer approaching full, operation aborted
228	-	Error encountered during EDIT for which no recovery was provided.
229	-	Overflow occurred in operation
23Ø	-	Can't find matching [

INDEX