Integers

The main activity of TECO is to manipulate character string data, but TECO can also operate on integers. The commands make use of integers for three special purposes: to count off the characters or lines in a string, to specify the number of times a command loop is to be executed, and to represent an ASCII character code.

The operators of elementary arithmetic, "+", "-", "*", and "/", are available for making up expressions. The division operator forms the quotient and then drops the fractional part without rounding. It is rare that the TECO user needs an expression with more than one operator in it; if such a case arises, however, the user should use parentheses to indicate the order of evaluation. For example, write "3+(4*5)", not "3+4*5.

RUNNING TECO

The purpose of TECO is to enable a user to enter a program or a document into the TENEX System. The essential steps of this process are (1) start up TECO, (2) type in the text, and (3) file the result; indeed, if a user could avoid mistakes and changes, this much would suffice.

A Complete Session with TECO

The dialog which follows is a sketch of the overall control of TECO by the user:

@TECO(%)	The user (who is at the TENEX
(%)	EXECutive Level) types "TECO" and a Carriage Return.
	TENEX loads and starts TECO.
*xxx(ESC)$(%)	TECO types "*" (go ahead sign); the user types one or more TECO commands, xxx; then types Escape (go ahead, TECO); and
	TECO echoes with "$" and a Carriage Return and then executes the command string.
*xxx(ESC)$(%)	The cycle is repeated until the user has completed his work and filed the results.
...	...
*;H(ESC)$(%)	Finally, the user gives the Halt Command (;H) to exit from TECO, and
@ ...	TECO returns the user to the EXECutive Level.

In the dialog just given, xxx represents a command string.

Several commands can be typed in as a single command string, and no separation is required between the commands. When the user has finished typing a string of one or more commands, he types the Escape character. Only then does the execution of the commands begin, proceeding from left to right.

The examples in this document make use of both upper and lower case letters (although some terminals are limited to upper case). TECO pays attention to the case of a letter only when the letter is part of the text being edited. When a lower case letter is typed as part of a command name TECO treats it as if it were upper case. Thus the Halt Command illustrated above can be typed in as either ";H" or ";h".

An Error

If TECO encounters a command which it cannot execute, it responds as in the following dialog:

*8C;ADD(ESC)$(%)	A string of four commands has been typed. TECO executes the first command, 8C, but rejects the next command ";A".
?42(%)	TECO types an error number
8C;A(%)	and types the command string through the illegal command (thus pointing out the error). TECO then discards the subsequent commands, "DD". Finally,
*	TECO calls for a new and better command string.

The Error Messages Appendix of the TECO Manual lists the error numbers used by TECO to report errors. In that appendix, the interpretation of "42" is "An undefined command character has been given". Therefore, the reason ";A" was rejected in the dialog above is that there is no such command in TECO.

The Illustrations

This writeup uses both isolated examples and dialogs to illustrate TECO. An isolated example shows the command string which a user types but does not show the response from the computer system.

A dialog is a longer and more complete way to illustrate TECO; it shows a sequence of interactions between the user and the computer system. In the dialogs the user type-in is underlined and each use of a control character is explicitly shown. These conventions make clear who has done what.

During an actual session with TECO, the underlines and the control-character names are not typed out. For the dialog given above (TECO's handling of an error), the actual listing would therefore be as follows:

```
*8C;ADD$
?42
8C;A
*
```

TYPING IN THE TEXT

The main storage of TECO is the buffer, and it holds the character string which is being edited. The buffer is initially empty, but it can expand to a capacity of over a million characters as the user enters his text. Since a typed page typically contains 2000 characters, the buffer can accommodate about 500 pages of text. Thus it is almost always the case that the entire program or document being edited fits in the buffer and can be manipulated as a single character string.

There is a pointer associated with the buffer. It specifies the position in the buffer at which the main editing activity is going on, and it is moved by commands which are discussed in a later section. Many TECO commands, including those described in this section, operate relative to this pointer.

Typing in the Text

The Insert-String Command is the command normally used to type text into the buffer. The command is entered by typing an "I", followed by a character string, s, of any length, followed by a Control-D or ESC. The command inserts the character string, s, into the buffer just before the pointer. The inserted string may contain any letter, digit, punctuation mark (including space), or formatting character. The only characters excluded are certain control characters which rarely appear in text.

Two examples of the Insert-String Command follow:

Ia(^D)	Insert the letter "a" just before the pointer.
IThis is(%) a test.(%) (^D)	Inserts two lines just before the pointer.

A full dialog will now be given to illustrate the Insert-String Command. In this dialog the user enters the upper-case alphabet into the buffer, five characters per line.

*IABCDE(%)	The user starts typing
FGHIJ(%)	the alphabet...
KLM(ESC)$(%)	Exhausted, and fearful of losing
(%)	his work, he terminates the command. TECO enters 14 characters (including 2 End-of-Line characters) into the buffer.
*INO(%)	After a pause, the user
PQRST(%)	completes the typing of
UVWXY(%)	the alphabet.
Z(%)	Note that input resumed
(ESC)$(%)	just where it left off, in
(%)	the midst of the line which was left unfinished in the first Insert-String Command.

The purpose of the (^D) at the end of a character string argument is to end the argument. When the argument is already at the end of a command string, the (^D) is not necessary. This useful exception is applied to both Insert-String Commands in the dialog just given, where (ESC) is used to end both the command string and the character string.

A Dangerous Error

Suppose the user, intent on the job of getting some text into the buffer, starts typing the text without preceding it with an "I". When the user types Escape, the text is not entered into the buffer; instead, TECO tries to interpret it as a command string. If the user is unlucky, TECO will be able to proceed through several "commands" before being stopped by an illegal command.

When this error occurs, the user needs to determine whether the text already in the buffer has been affected by the execution of the false commands. He can check this in one of three ways:

-- He can trace, command by command, the action of TECO in its runaway interpretation of the text. Usually this is easy; occasionally it is very

difficult.

-- He can type out and read the entire contents of the buffer.

-- He can start over; that is, he can go back to the previous version of the file being edited.

Usually, TECO stops before the buffer has been modified; but some damage to the user's text is always a possibility when a random sequence of commands is executed.

Recovering Lost Type-in

TECO has a backup register in which it saves the most recent command string which was over 15 characters long. A command string is saved just before execution begins, so the whole string is saved even if it contains an illegal command. This backup register is of great interest to the user who has just typed a long insertion without supplying the initial "I", as described in the previous paragraphs.

The ;Get-Commands Command (;G) makes a copy of the command string in the backup register and inserts the copy into the buffer just before the pointer. The following dialog illustrates the use of this command:

*Daffodils ...	The user is typing a
...	novel as a single
... snow fell.(%)	Insert-String Command...
(ESC)$(%)	But he leaves the "I" off!
?32(%)	TECO tries to interpret the
D(%)	text as a command string and (fortunately) fails before executing a single command.
*;G(ESC)$(%)	The user recovers with a
(%)	;Get-Commands Command, and the novel is copied from the backup register into the buffer.

The Carriage Return

The type-in of a Carriage Return, indicated by "(%)" in the dialog, requires a special explanation. Strictly speaking, it is the function of the Carriage Return key only to move the printing element back to the left margin. A second key, Line Feed, is provided to advance the paper TENEX intervenes as follows:

-- A Carriage Return character (decimal code 13) coming in from the user's terminal is echoed as a Carriage Return followed by a Line Feed and is then entered into storage as an End-of-Line character (decimal code 31).
#
-- An End-of-Line character going out to the user's terminal is converted into a Carriage Return followed by a Line Feed.
#
This arrangement has substantial advantages. When the user wants to end a line, he needs only a single keystroke (Carriage Return). Further, the separation between lines in a stored character string is represented by a single character used only for that purpose (End-of-Line). Of course, the user must use a special technique to enter a true Carriage Return character into storage, but the need for this character is rare.
#
Typing Tricky Text
#
The list of allowed characters given in the description of the Insert-String Command excluded some ASCII characters. The fine points of this matter are discussed in the Character Set Appendix. For the present, it is sufficient to introduce a command which can be used to insert any ASCII character into the buffer. The Insert-Code Command (nI) inserts a single character into the buffer just before the pointer. It inserts the character whose decimal code is given by the integer value, n, which precedes "I" in the command. This command offers full generality but is not, of course, as convenient as the Insert-String Command.
#
In the following dialog, the user wants to type out a line of slashed zeroes by using (1) a line of "O" characters, (2) a true Carriage Return (without a Line Feed), and (3) a line of slashes. He must use the Insert-Code Command to get a Carriage Return (decimal code 13) into the buffer since a typed Carriage Return would be transformed by TENEX into an End-of-Line character.
#
*IOOO(^D)$13II///(ESC)$(%)
(%)
#
FILING THE TEXT
#
TECO communicates with the TENEX file system on behalf of the user; that is, certain TECO commands are used to transmit text between a TENEX file and the editing buffer of TECO. Since the editing buffer has such a large capacity, the user can almost always read his entire file into the buffer rather than process it piece by piece. Under these conditions, input/output is

simple and only three commands are required, as follows:

-- The ;Yank-File Command (;Y) is the input command. It places a copy of the designated file after whatever is already in the editing buffer and leaves the file unchanged.

-- The ;Unget-File Command (;U) is the output command. It replaces any previous contents of the designated file with a copy of the entire contents of the buffer and clears the buffer.

-- The ;Save-File Command (;S) saves the entire buffer on a new version of the file. The buffer is not cleared.

File Designators

Each of these commands requires a file designator in order to continue. A full description of the way in which TENEX files are designated is outside the scope of an introduction to TECO. This discussion will assume the simple case that the desired file is on the main system storage device and is in the user's own directory. In that case, a file is uniquely designated by

-- the name of the file followed by a ".",

-- the extension followed by a ";", and

-- the version number.

The name and the extension are each a sequence of letters and digits; the version number is an unsigned integer. For example, "HENRY.IV;2" is a file designator.

The Designator Dialog

A ;Yank-File or ;Unget-File Command obtains its argument in an unusual way: it asks for it. the dialog proceeds as follows:

-- The user types the command (";Y" or ";U" or ";S") followed by an Escape.

-- TECO types "INPUT FILE:" or "OUTPUT FILE:", as appropriate.

-- The user types all or part of a file designator, followed by an Escape.

-- TECO types a brief message acknowledging the correctness of the file designator and asking for confirmation.

-- The user types a Carriage Return to confirm.

-- TECO performs the input or output operation and then types "*" when it is ready for further commands.

When things do not go smoothly, one of the following cases applies:

-- If TECO finds that a file designator is illegal, TECO types "?" and prompts the user to try again. This occurs when a file requested for input does not exist or when a file designator is ill-formed.

-- If the user decides to abort the command before typing the confirming Carriage Return, he must type two Delete characters. TECO will then go to the await-commands state without performing any input/output.

-- If the user decides to abort the command after typing the confirming carriage return, it is too late. He should let the operation run to completion and then take whatever action is appropriate. Otherwise, he must cope with a partially completed input/output operation, and this requires study of the section called "Complicated Input/Output".

Designator Recognition

When the user is inputting a file, he can type just enough of the name and extension to uniquely specify the file in his directory and then type the Escape character. TECO will then ascertain and type out the remainder of the designator. Occasionally, the designator assumed by TECO will not be what the user wanted, and he can use two Delete characters to get out of the designator dialog and start a new input/output command. On other occasions, TECO will decide that the file designator thus far typed is inadequate to uniquely specify a file and will ring the bell at the user's terminal to ask for more characters of the designator.

As a special case which is very useful, the user can reply to the request for a file designator by typing Escape immediately, without giving any part of the required file designator. TECO will assume (and type out) the designator for

the file last used in a ;Yank Command in the current TECO
session.
#
It is a good rule to let TENEX fill in the version number of
a file. That is, even if the user does not make general use of
designator recognition, he should not type the version number.
When the user types Escape, TENEX will fill in the appropriate
version number. For an output file, TENEX will supply "1" for a
new file or a version number one greater than the highest version
of an existing file. For an input file, TENEX will supply the
highest version number of an existing file.
#
If the rule just mentioned is followed, a new version number
will be created every time a file is edited, and no previous
version will be overwritten. This convention makes it easy to
keep the previous version of a file until the integrity of a new
version has been established. When an old version is no longer
wanted, it can be deleted by the TENEX Executive Command DELETE.
Thus the preparation of text and the housekeeping of the file
directory are separate activities.
#
Preparing a Long File
#
It is prudent to pause from time to time in preparing a large
file and save a copy of the file as it currently stands. This
limits the loss which can result from a system crash, the break
down of the communication line, or a serious user error. In the
following dialog, the user saves two intermediate versions of his
file before outputting the third and final version.
#
#
@TECO(%)   The user starts TECO,
* ...   types in a lot,
then pauses to save a copy.
*;S(ESC)$(%)
(%)
OUTPUT FILE: HENRY.IV;(ESC)1 [New File](%)
(%)

```
#       * ...           The user continues to type in his
#                       file.  The next time he saves a
#                       copy, TECO types the designator.
#       *;S(ESC)$(%)
#       (%)
#       OUTPUT FILE: (ESC)HENRY.IV;2 [New Version](%)
#       (%)
#       * ...           The user types the remainder of his
#                       file and outputs the complete copy.
#       *;U(ESC)$(%)
#       (%)
#       OUTPUT FILE: (ESC)HENRY.IV;3 [New Version](%)
#       (%)
```

In a complicated project and especially in a project in which files are shared among many users, it is useful to maintain a record of the time and source of each new version of a file. The ;Date-and-Unget-File Command (;D) provides a way to keep this record within the file itself. This command is equivalent to the ;Unget-File Command (;U) except that it adds a date line at the beginning of the buffer just before outputting the buffer. The date line consists of

-- a comment string (usually just a single semicolon), followed by

-- the complete file designator, followed by

-- the date and time at which the new version is being written out, followed by

-- the name of the user who has produced the version.

If this command is used to perform output of every new version of a file, a log of all modifications to the file will be accumulated at the beginning of a file. Since each date line begins with a semicolon, it is ignored by the many TENEX subsystems which treat such a line as a comment. Since a date line is just an ordinary line of text, it can be deleted when it becomes superfluous.

POSITIONING THE POINTER

The pointer is always located between two characters or, as a special case, at the beginning or end of the buffer. The pointer is not a character itself, and does not take up any space in the buffer. TECO editing is built up around the moving of this pointer back and forth through the text, and many of the TECO commands operate relative to the current position of the pointer.

TECO maintains several registers which contain integer values with special significance. The register named "." (period) always contains the number of characters in the buffer before the current position of the pointer. The register named "Z" always contains the number of characters in the entire buffer. "B" contains Ø, which is the beginning of the text buffer. ";B" is the character address at the top of the current page, while ";Z" is the location of the bottom of the current page. If ;B is preceded by a number, its value is the location of the top of that page. The user can refer to these registers by name wherever an integer value is required in a TECO command.

The next two commands to be considered are the simplest in TECO. Each specifies the positioning of the pointer by an integer argument. As is the case with many TECO commands, this integer argument can be omitted and the TECO interpreter will fill in a well-defined default value.

The Jump

The Jump Command (nJ) places the pointer after the n-th character of the buffer.

28J	Places pointer after the first 28 characters of the buffer.
ØJ	Places the pointer at the beginning of the buffer (after Ø characters).
J	Means ØJ.
ZJ	Places the pointer at the end of the buffer (after all Z characters).
Z-1J	Places the pointer just before the last character of the buffer.
.+3J	Advances the pointer across 3 characters.
;BJ	Jump to the top of this page.
;ZJ	Jump to the bottom of this page.
24;BJ	Jump to the top of page 24.

The Character Skip

The Character-Skip Command (nC) advances the pointer n characters through the buffer.

3C	Advances the pointer across 3 characters.
C	Means 1C.
-29C	Moves the pointer 29 characters backward (toward the beginning of the buffer).
-C	Means -1C, moves pointer backward one character.

The Line-Skip

Certain commands in TECO are line-oriented. They consider each End-of-Line character in the buffer to be the last character of a "line" of characters, and consider any character after the last End-of-Line character in the buffer to be a line. They consider the line which contains the character just after the pointer to be the "current" line of the buffer. The line-oriented commands operate on the buffer in terms of these imaginary divisions.

The first of the line-oriented commands is the Line-Skip Command (nL). This command advances the pointer to the beginning of the n-th line after the current line. The argument to this command does not have to be positive. When n is zero, the phrase "the n-th line after the current line" means "the current line" and when n is -5 (for example), the phrase means "the fifth line before the current line".

3L	Advances pointer to the beginning of the third line after the current line.
L	Advances pointer to the beginning of the next line (1L is assumed).
ØL	Moves pointer back to the beginning of the current line.
-L	Moves pointer back to the beginning of the previous line (-1L is assumed).
-12L	Moves pointer back 12 lines.
:L	Move to the end of the current line.
-:L	Move to the end of the previous line.
2:L	Move to the end of the next line.

Combinations

Since an individual TECO command can be just one or two characters long, it is convenient to run a few commands together when they perform an operation which, from the user's point of view, is unitary. The first of the following examples is particularly useful.

L-C	Places the pointer just after the text of the current line (and just before the End-of-Line character).
J3L	Places the pointer at the beginning of the fourth line of the buffer.
ZJØL	Places the pointer at the beginning of the last line of the buffer.
ZJØLC	Places the pointer after the first character of the last line of the buffer.

The last example of a command string could give an error message. If the last character of the buffer is an End-of-Line character then the last line of the buffer is the empty string of characters between the End-of-Line and the end of the buffer. In this case, "ØL" leaves the pointer at the end of the buffer. This situation arises often in actual practice.

TYPING OUT TEXT AND INTEGERS

The user examines the contents of the buffer by means of the Type-Out Commands. The simplest of these is the Type-String Command (m,nT). It types everything from just after the m-th character of the buffer to just after the n-th character.

TECO has a convenient abbreviation, "H", for the argument pair, Ø,Z which designates the contents of the whole buffer. This abbreviation can be used with the Type-String Command.

HT	Types the whole buffer.
Z-1Ø,ZT	Types the last 1Ø characters in the buffer.
Ø,.T	Types the buffer up to the pointer.
;B,;ZT	Types the current page.

Line-Oriented Type-Out

A more convenient Type-Out Command is the Type-Lines Command (nT), which types the characters between the pointer and the beginning of the n-th line after the current line. The argument, n, can be zero or negative, as with the Line-Skip Command.

3T	Types characters from the pointer up to the beginning of the third line after the current line.
T	Types the part of the current line which is after the pointer (1T is assumed).
ØT	Types the part of the current line which is before the pointer.
-12T	Types the previous 12 lines and the part of the current line before the pointer.

The Type-String and Type-Lines Commands have the same code name, "T", but they are distinguished by having a pair of arguments and one argument, respectively.

A second line-oriented type-out command is the View Command (m,nV), which types m-1 lines before the current line, then types the current line, then types n-1 lines after the current line. When the two arguments, m and n would be equal, the command can be used with a single argument (nV).

1Ø,2V	Starts with the ninth line before the current line and ends with the first line after.
2V	Types the preceding line, the current line, and the following line ("2,2V" is assumed).
V	Types the current line ("1,1V" is assumed).

Three Type-Out Commands have been described here although, in theory, one would be sufficient. Each has its particular application. The Type-String Command, with its fussy generality, is seldom useful except for the special form, "HT", which types out the whole buffer. The Type-Lines Command, with its sensitivity to the position of the pointer, is used to check the pointer position just before the insertion or deletion of text. Finally, the View Command, with its convenient simplicity, is used to look at the general context in which the pointer appears.

Examples of Type-Out

For this dialog, the buffer contains the alphabet as it was typed in earlier, in the examples of the Insert-String Command.

```
*HT(ESC)$(%)          This command types out
ABCDE(%)              the whole buffer and, as
FGHIJ(%)              promised, reveals the
KLMNO(%)              alphabet, stored 5
PQRST(%)              letters per line.
UVWXY(%)
Z(%)
(%)

*10,15T(ESC)$(%)      What does this do?
J(%)                  (Everybody who expected
KLM(%)                "KLMNO" raise their hand.)

*15J(ESC)$(%)         Puts the pointer before "N".
(%)
*2T(ESC)$(%)          Types the remainder of
NO(%)                 the current line and
PQRST(%)              all of the next line.
(%)
*0T(ESC)$(%)          Types the current line
KLM(%)                up to the pointer.

*V(ESC)$(%)           Types the entire
KLMNO(%)              current line.
(%)
*2V(ESC)$(%)          Types the current
FGHIJ(%)              line and one neighboring
KLMNO(%)              line in each
PQRST(%)              direction.
(%)
```

TECO follows the successful completion of any command by typing out an End-of-Line. It follows that a blank line will appear after a type-out if and only if the text being typed out ends with an End-of-Line. For example, the alphabet typed out above ends with an End-of-Line and is followed by a blank line, indicated by the solitary "(%)".

Abbreviated Commands

A line feed character simulates the command string "LT(ESC)" if it appears as the first character in a command. Thus, it advances the pointer to the beginning of the next line and then types that line.

Backspace (^H) types the preceding line by simulating the command

"-LT(ESC)". As with linefeed, backspace operates in this fashion only if it is the first character in the command.

Typing Out Integers

The Type-Integer Command (n=) types out the value of the integer expression, n. The command is useful in obtaining the value of "." (the number of characters before the pointer) or "Z" (the number of characters in the buffer), but it is not limited to this purpose.

The Access-Code (1A) Function of TECO has as its value the ASCII code for the character which immediately follows the pointer. This function can be used wherever an integer value is allowed. In particular, it can be used as the argument to the Type-Integer Command just described.

Under certain circumstances, the Access-Code Function can be essential. The situation is analogous to the problem of inserting control characters into the buffer, which was discussed earlier when the Insert-Code Command was introduced. Just as there are certain characters which can be inserted only by means of their integer codes, so there are certain characters which can only be examined in this way. Some of the characters do not print out anything (not even a space) when the string of which they are included is typed out; for example, Control-A. Other characters are modified by TENEX as they are typed out; for example, a lower-case letter is capitalized when it is transmitted to a terminal which does not have lower case.

The following example assumes the buffer contains the alphabet, as before.

*15J(ESC)$(%) (%)	Puts the pointer after the 15-th character.
*.=(ESC)$(%) 15(%) (%)	Shows that the pointer is after the 15-th character.
*Z=(ESC)$(%) 32(%) (%)	Shows that there are 32 characters in the buffer.
*1A=(ESC)$(%) 78(%) (%)	Shows that the character after the pointer is "N" (decimal code 78) and not "n" (decimal code 109).

DELETING CHARACTERS

The user can delete a substring from the buffer by any of three commands. The simplest of the three is the Kill-String Command (m,nK). It deletes everything from just after the m-th character

of the buffer to just after the n-th character and then moves the
pointer to the position of the deleted characters. The special
form "HK" deletes the contents of the whole buffer.
#
A given Kill-String Command deletes exactly what a
Type-String Command with identical arguments types out. This
makes it easy to "simulate" a deletion (by typing out the string
to be deleted) before performing the actual deletion.
#
The Kill-Lines Command (nK) is the line-oriented Deletion
Command of TECO. It deletes the characters between the pointer
and the beginning of the n-th line after the current line.
Again, in parallel with the Type-Out Command, this command
deletes exactly what the Type-Lines types out.
#
#
K Kill the (remainder of) current line.
#
:K Kill the (remainder of) current line, but
not the end-of-line character.
#
3K Kills the (remainder of) current line and
the two following it.
#
3:K Same as 3K but the final end-of-line is
left.
#
;B,;ZK Kills this entire page.
#
.,;ZK Kills the part of the page after the
pointer.
#
-K Kill the preceding line and the initial
portion (before ".") of this line.
#
ØK Kills the part of this line to the left
of the pointer.
#
-:K Same as -K except an additional
end-of-line character is also killed.
#
#
The Delete-Characters Command (nD) operates relative to the
pointer. It deletes the n characters just after the pointer.
There is no Type-Out Command which is directly analogous to this
command; but this command is normally used to delete just a few
characters and is used more casually.
#
8D Deletes the 8 characters just after the
pointer.

D	Deletes the character just after the pointer (1D is assumed).
-D	Deletes the character just before the character (-1D is assumed).

The End of the Alphabet

In the following dialog, the alphabet (as entered by the Insert-String Command many pages ago) makes its farewell appearance.

*JCDDV(ESC)$(%)	Starts at the beginning, skips
ADE(%)	a letter, deletes two, checks.
(%)	
*2,7T(ESC)$(%)	Simulates deletion of
E(%)	characters
FGH(%)	3 through 7.
*2,7K(ESC)$(%)	Performs deletion.
(%)	
*C6DC4DC(ESC)$(%)	Does more deletions.
(%)	
*3T(ESC)$(%)	Simulates deletion
T(%)	from pointer
UVWXY(%)	through next
Z(%)	2 lines.
(%)	
*3KHT(ESC)$(%)	Performs deletion.
ADIOS(%)	and checks result.
(%)	

CONTROLLING TECO

At the beginning of this section, we observed that the ordinary commands of TECO are executed only when an Escape is typed. In addition to these ordinary commands, however, TECO has certain control commands which are single characters and which are executed immediately. These commands are used to control TECO itself rather than to edit the text in the buffer.

Interrupt TECO

The user sometimes needs to access the TENEX EXECutive. For example, he may want to use the DIRECTORY command to check existing file names before choosing a name for a new file he has prepared; he may LINK to another TENEX user without wrapping up his TECO session; or he may wish to eliminate an unwanted file by means of the DELETE command. To get back to the TENEX EXECutive he uses the TECO Control Command Control-C (^C) and ends with the EXECutive command CONTINUE, as in the following dialog:

```
*IA(HT)          B(%)     The user inserts a line
(^D)-LT(ESC)$(%)          which includes a tab
A       B(%)              and types it out
(%)                       with standard tab stops.
*(^C)^C(%)                Interrupts TECO.
@STOPS 3(%)               Uses Exec to set stop.
@CONTINUE(%)              Returns to TECO.
T(ESC)$(%)                Types out the line
A  B(%)                   with new tab stops.
(%)
```

The Control-C Command does not always result in an immediate interrupt. If TECO is performing input/output, the interrupt will be delayed until the buffers have been properly emptied. To obtain an immediate interrupt, the user can type a second Control-C; when he does so, however, data in the buffers may be lost. Usually this is acceptable only during a type-out at the user's terminal.

Abort a Command

A different kind of interrupt is produced by the Delete Command (DEL), also called RUBOUT. Two cases must be considered:

-- If Delete is typed during execution of a command, that command is immediately aborted and TECO types "*" and enters its await commands state.

-- If Delete is typed during type-in of a command (when TECO is not executing a command), TECO rings the bell on the terminal and waits to see what the user does next.

   -- If the user types a second Delete, TECO discards the command string thus far typed in, types "*", and enters the await commands state; however,

-- If the user types anything but a Delete, TECO assumes the first Delete was a mistake and forgets it.

The cautious handling of this case is appropriate because the user could type many lines of text as an unfinished Insert-String Command and then accidentally type a Delete. Rather than wiping out the type-in, TECO rings the bell to warn the user not to type Delete again.

In the following dialog, the user starts an input command, gives the wrong file designator, and deliberately aborts the command instead of confirming it. A second try causes the desired file to be read in. The user instructs TECO to type the whole file, reads the first few lines, decides it looks right, and aborts further type-out.

```
*;Y(ESC)$(%)
(%)
INPUT FILE: HENRY.VI(ESC);1 [Confirm](DEL)(BEL)(DEL)
                           (%)
*;Y(ESC)$(%)
(%)
INPUT FILE: HENRY.IV(ESC);3 [Confirm](%)
21982 Chars(%)
(%)
*HT(ESC)$(%)
So shaken are we,(%)
so wan with care,(%)
Find we (DEL)(%)
(%)
(%)
*...
```

Checking up on TECO

An interrupt of a special kind is produced by the Control-T Command. This command can be used at any time at all and will promptly report on the status of the System, giving the user the status of TECO (waiting for input or running), the TENEX load average, and the CPU and console time used. Since the command never disturbs the command being executed by TECO, it is a safe and convenient way to check up on TECO when TECO seems to be taking a long while to execute a command.

The Control-T Command can be used to determine whether a delay in the response of TECO is due to a heavy load on the TENEX System, a minor breakdown in TENEX, or an infinite loop entered by the user (this facility will be explained later). But an

especially interesting example is the following recovery
procedure.
#
When there has been a failure in the TENEX System, the user
may find himself in a somewhat uncertain position in TECO.
Either (1) there has been a crash of TENEX TECO and the system
has been restored without the loss of its previous state or (2)
the communication link between the user and TENEX has been broken
and the user has used the EXECutive ATTACH command to
re-establish the connection and resume at the point of
interruption. The uncertainty results from certain random
processes which may occur during the breakdown. The user should
explore the situation as follows:
#
(^T)IO WAIT AT 4262(%)
LOAD AV. = 0.07, USED 0:02:08.6 IN 0:48:06(%)
First,the user determines the status
of TECO.
Apparently TECO is waiting for a
command.
Next, the user looks at the current
command string:
(^R)(%)
BTRFSK#!(DEL)(BEL)(DEL)(%)
The command string looks like line
noise and the user erases it.
* ... The user proceeds, if necessary, to
check the state of TECO in other
ways appropriate to the situation.
#
Erasing Typing Errors
#
A command string is not executed as it is typed in. Instead, it
is accumulated in a special command register and is executed only
when an Escape is typed. The three commands given here are
specialized commands designed to assist in correcting a command
string as it awaits execution in the command register.
#
Each erasing command is a single control character, and acts
immediately when it is entered. The Control-A causes the last
character in the command register to be erased. The Control-Q
Command causes the last non-empty line in the command register to
be erased. The Control-R (for "Retype") Command causes the last
non-empty line in the command register to be typed out.
#
In the following dialog, the user inserts three words and
types out the buffer. Along the way, he illustrates the use of
single and multiple erasures and of the retyping of a line to
"clean it up" after it has been cluttered by erasures.

```
#       *HASTE(%)                       The user starts
#       MAKES(%)                        an insertion.
#       (^Q) (%)                        He notices the "I"
#       (^Q)(%)                         is missing, erases
#       *IHASTE(%)                      the command string
#       MAKE(%)                         and starts over.
#       WA(^A)\A(^A)\W(^A)\(%)          He erases "A", "W",
#       (^R)(%)                         and an End-of-Line
#       MAKES(%)                        and Retypes the line.
#       WASTE.(%)
#       (ESC)$(%)
#       (%)
#       *HK(^A)\KT(ESC)$(%)             The user almost clears
#       HASTE(%)                        the buffer, but
#       MAKES(%)                        changes "K" to "T".
#       WASTE.(%)
#       (%)
#
# Erasing typing errors using Backspace Key
#
#
#       Some terminals are able to perform the backspace function.
# Hardcopy devices do this by moving the print head; CRT displays
# can move the cursor. To take advantage of this, Control-H or
# Backspace deletes characters just as Control-A does but indicates
# what has been deleted by using the mechanical backspace
# mechanism. In order to activate this function, the user must
# tell TECO what style terminal he is using. This is done by
# commands such as 3^H$. (Four characters: 3, ^, H, and ESCape.)
# Instead of 3 the user should select one of the following:
#
#       Code            Terminal
#
#       Ø           No mechanical backspaces (Model 33)
#
#       1           Mechanical backspace but no eraser.
#                   (TI, 2741)
#
#       2           Scope that uses ^H to backspace
#                   the cursor.
#
#       3           Bendix scope, Backspace sequence
#                   is ESCape D.
#
#       4           Terminal, VTØ6 scope.  Backspace
#                   character is ^Y.
#
#       5           Beehive.  Backspace character
#                   is ^D.
#
#       6           Infoton, Backspace character is ^Z.
```

2. LARGE-SCALE EDITING
#
The previous section described a collection of commands. Those commands can be used to select any position in the text being edited and then insert or delete any characters at that position. Additional commands are required when the file being edited is large and the modifications being performed are complicated.
#
When a file is more than a few dozen lines long, it is not efficient to locate a position within the file by counting lines or characters; instead, commands are required which can locate a particular phrase or identifier or number within the buffer. When a passage of text which is more than a few words long must be moved, it is not efficient to delete the passage and then retype it elsewhere; instead, commands are required which can extract, move, and insert the passage without retyping. When a particular modification must be made over and over (as in the case of a consistently misspelled word), it is not efficient to type the necessary command string over and over; instead, some kind of loop command is required.
#
The commands described in this section fill the requirements just mentioned. Although the commands described are introduced here by the problems of large-scale editing, they are useful for all editing jobs. The commands of the previous section are the framework of TECO; the commands in this section supply the power.
#
SEARCHING THROUGH THE TEXT
#
The Search Command is used to search the buffer for an occurrence of a particular substring. It is entered by typing "S" followed by a character-string argument. The character-string argument is a sequence of characters, the citation, terminated by typing Control-D. The Control-D can be omitted when a Search Command occurs at the end of a command string.
#
Ordinary searches look after the pointer for a character sequence which matches the citation. If the repetition count (v.i.) is negative, a reverse search has been specified and the search will proceed backwards from the pointer. In either case a match is found, the pointer is moved to the position just after the matched character sequence; otherwise, the pointer is not moved from its original position and TECO types the error message "?SEARCH?35".
#
The Search Command can be preceded by an integer value, n, and will thereupon be repeated n times. The effect is to search for the n-th occurrence of the citation after the pointer.

Sa(^D)	Finds the first occurrence of "a" after the pointer and moves the pointer to just after that occurrence.
-Sa(^D)	Finds the first occurrence of "a" before the pointer.
Sit(^D)	Finds "it" in any context, either as an independent word or within another word.
S(%) Here (^D)	Finds an occurrence of the word "Here " at the beginning of a line.
3S;(^D)	Finds the third occurrence of a semicolon after the pointer and moves the pointer to just after that occurrence.
-3S;(^D)	Finds the third occurrence of semicolon before the pointer. The pointer is left after the find.

A successful search of the buffer always moves the pointer. This is taken for granted in the explanation of some of the examples.

The Search Command can be deceptive. It is quite natural for a user to give a search command for a particular pattern, get an error message in response from TECO, and conclude that the specified substring is not present in the buffer. However, this conclusion is unwarranted if the user forgot to set the pointer to the beginning of the buffer before the search.

The Replace Command

A natural extension of the Search Command is the Replace Command, which not only searches the buffer but modifies it. The command is entered by typing "R" followed by two character string arguments, the citation and the replacement. Each character string argument is terminated by typing Control-D. The command does exactly what the Search Command does and one thing more: if the substring specified by the pattern is found, it is deleted and a copy of the replacement is inserted. Replace commands also may take a repetition count which can be negative to specify a reverse replace.

Ra(^D)b(^D)	Finds the first occurrence of an "a" after the pointer, moves the pointer to just after that occurrence, and replaces it with "b".

```
#       R(%)                    Replaces the next "Here " which
#       Here (^D)(%)            begins a line with a "There ".
#       There (^D)
#
#       -3R@#!(^D)(^D)          Replaces the past three "@#!"
#                               strings with nothing; that is,
#                               deletes them.
#
#       R(%)                    Deletes the next End-of-Line.
#       (^D)(^D)
#
#       Ra page(^D)a(%)         Starts a new line between
#       page(^D)                the words "a" and "page".
#
```

The Replace Command is perhaps the most frequently used command in TECO. When a person is making the changes indicated in a marked-up listing of a file, he can often proceed from beginning to end with one Replace Command after another. When there is danger that a Replace Command may apply in the wrong place, the user can simply include a little more context in the pattern. Consider, for example, the following ways of making "big plans" into "big plane".

```
#       Rs(^D)e(^D)             This works if the site of the
#                               change is just a few characters
#                               after the pointer and there is
#                               no intervening "s".
#
#       Rplans(^D)plane(^D)     This is more selective and will
#                               be right unless there is
#                               another "plans" along the way
#                               to the site of the change.
#
#       Rbig plans(^D)big plane(^D)
#                               This is still more selective.
#
#       Sbig plans(^D)-DIe(^D)
#                               This saves a few keystrokes but
#                               it is more complicated and
#                               error prone.
#
```

It is good practice to follow a Replace Command with a View Command. The type-out verifies that the citation and replacement were correct and that the modification was applied at the right place in the buffer. Further, when the user is in the habit of typing out each change, he can risk small errors, such as a misplaced replacement, in order to work faster.

Something Extra

Three match control characters are provided for use in the pattern of a Search Command or Replace Command. They are:

(^X)	Matches any character which appears at the corresponding position in the buffer.
(^S)	Matches a Separator; that is, any character except a letter, digit, ".", "$", or "%". (These are the characters which are commonly used in identifiers.)
(^N)	Matches any character except the character which immediately follows the (^N) in the citation.

The match control characters can be used in a citation with other characters in any combination or sequence. However, a Control-N and the character which follows it act as a pair to match (or not match) a single character of the buffer.

S[(^X)(^X)](^D)	Finds any two characters enclosed in brackets.
S(^S)it(^S)(^D)	Finds an "it" which is not a part of a longer word,
S(^S)(^N)(^S)a(^D)	Finds a word whose second letter is "a". Note that "(^N)(^S)" are used in combination to match anything except a Separator character.

An Application

The Search Commands cannot be convincingly applied to the example we have been using (the alphabet). This is precisely because they are designed to look at the content of the buffer. Accordingly, we assume conventional text has been typed into the buffer, errors found, and a decision to correct them.