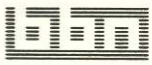
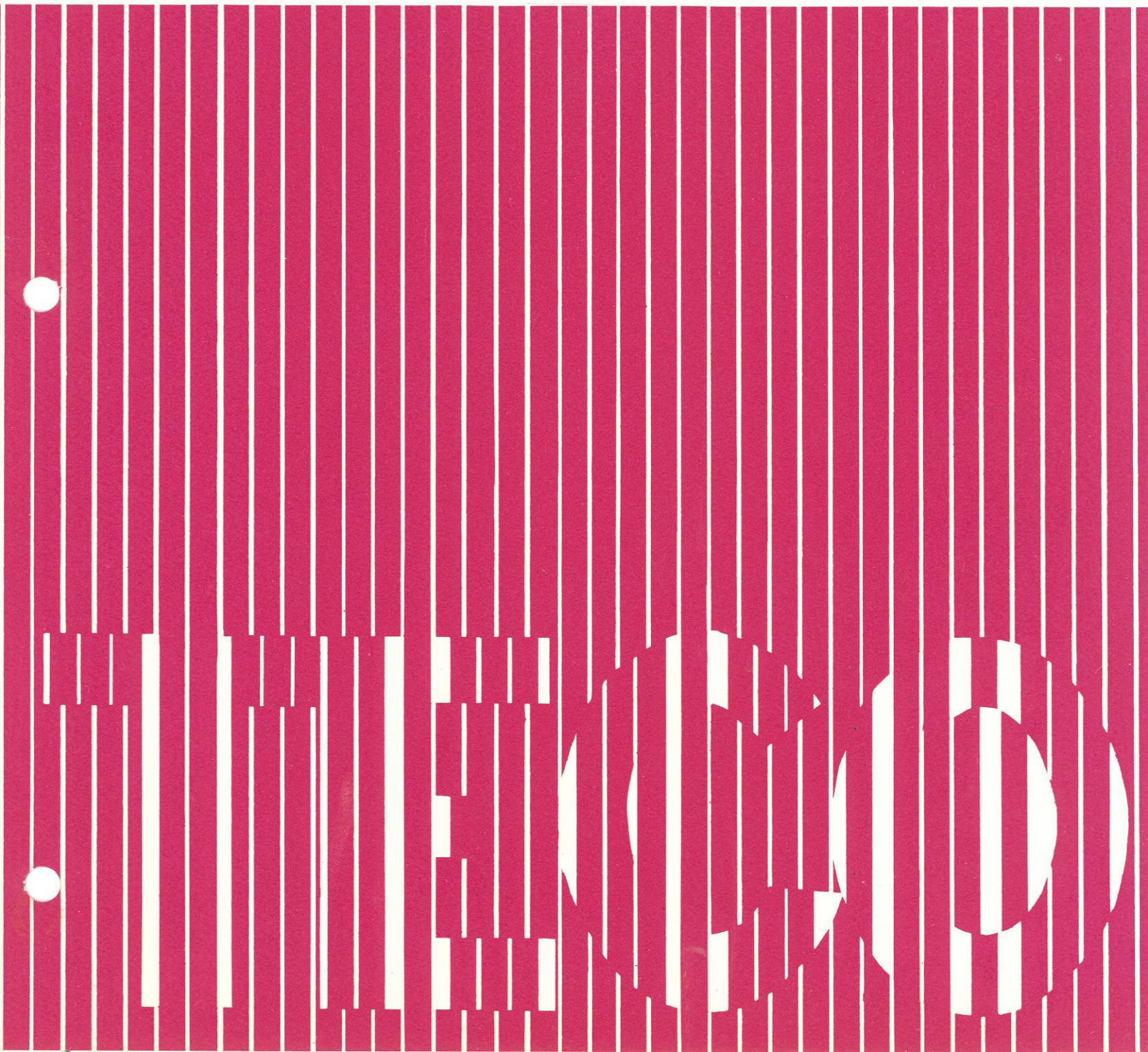


Bolt Beranek and Newman Inc.

TENEX

Text Editor and Corrector Manual



TENEX TECO

Text Editor and

Corrector Manual

Bolt Beranek and Newman Inc.
5Ø Moulton Street
Cambridge, Massachusetts Ø2138

TABLE OF CONTENTS

TUTORIAL - PART I

HANDBOOK - PART II

Appendices

PART I

TUTORIAL FOR TENEX TECO

This tutorial is designed for a single fast reading. Its purpose is to convey the details of TECO to a reader who is acquainted with the TENEX System and who has some idea of what to expect from a text editor.

The first chapter describes a subset of TECO commands which are sufficient to do any ordinary editing job. The second chapter describes commands which are essential for editing large files, and includes some of the most useful commands in TECO. The final chapter gives brief descriptions of the commands of TECO not already described, and introduces the reader to groups of specialized commands for which he may later have a need.

Chapter 1

BASIC EDITING

The text which TECO edits is a pure character string, without restrictions or exceptions. A sequence of characters of almost unlimited length can be accommodated, and any one of the 128 ASCII characters can appear at any position in this sequence. Even printer-control operations such as carriage return or tab are represented by ASCII character codes, and they require neither a special form of data to store them nor special commands to manipulate them.

The Character Set

The complete ASCII Character set is described in Appendix B of this manual. The characters can be divided into three groups, as follows :

-- Each of the graphic characters (95 of them) prints a single letter, digit, or punctuation mark and then advances the printing element one position. The Space character is included in this group.

-- Each of the format control characters (7 of them) causes the output device to take a special action, such as starting a new line, spacing to a tab stop, or ringing the bell.

-- Each of the command control characters (26 of them) can be used by TENEX software for a one-character, special-purpose command.

Names for Control Characters

Since a control character does not type a particular graphic, such as "a" or "+", it must be given an artificial name. In the text of this manual, a name is used which indicates the mode of production of the character on a terminal. For example, the character produced by holding down the CTRL key and striking the "D" key is called Control-D. In the listings of interactive dialog between the user and TECO, a shorter name is used for a control character, and this name is set off from the rest of the dialog by an enclosing pair of parentheses. For example, "(↑D)" is used for Control-D.

The following table gives the short names of the most important control characters and suggests the way in which the characters are used in TECO:

(%) A Carriage Return was transmitted at this point in the dialog. It was used to end a line.

(HT) A (Horizontal) Tab was transmitted. It was produced by Control-I and caused the terminal to space to the next tab stop.

(↑C) A Control-C was transmitted. It interrupted TECO.

(↑D) A Control-D was transmitted. It terminated the character string argument of a TECO command.

(ESC) An Escape was transmitted. It was produced by a key with "ESC" or "ALTMODE" or "PREFIX" written on it and caused TECO to execute a string of previously typed commands.

(DEL) A Delete was transmitted. It was produced by a key with "DEL" or "RUBOUT" written on it and caused TECO to abort the command it was executing.

The functional descriptions just given are introductory sketches. Subsequent sections will describe the use of these characters in detail.

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. TECO commands for these essential steps are included in this section and the next two sections.

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 manual 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 this 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 manual 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 an the computer system. In the dialogs in this manual, 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 2ØØØ characters, the buffer can accommodate about 5ØØ 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(%) FGHIJ(%) KLM(ESC)$(%) (%)	The user starts typing the alphabet... Exhausted, and fearful of losing his work, he terminates the command. TECO enters 14 characters (including 2 End-of-Line characters) into the buffer.
*INO(%) PQRST(%) UVWXY(%) Z(%) (ESC)$(%) (%)	After a pause, the user completes the typing of the alphabet. Note that input resumed 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 directoy. 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 chapter 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).
0L	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)$(%)
ABCDE(%)
FGHIJ(%)
KLMNO(%)
PQRST(%)
UVWXY(%)
Z(%)
(%)
```

This command types out the whole buffer and, as promised, reveals the alphabet, stored 5 letters per line.

```
*1Ø,15T(ESC)$(%)
J(%)
KLM(%)
```

What does this do? (Everybody who expected "KLMNO" raise their hand.)

```
*15J(ESC)$(%)
(%)
```

Puts the pointer before "N".

*2T(ESC)$(%) NO(%) PQRST(%) (%)	Types the remainder of the current line and all of the next line.
*ØT(ESC)$(%) KLM(%)	Types the current line up to the pointer.
*V(ESC)$(%) KLMNO(%) (%)	Types the entire current line.
*2V(ESC)$(%) FGHIJ(%) KLMNO(%) PQRST(%) (%)	Types the current line and one neighboring line in each 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 1Ø9).

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 chapter, 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. =    Ø.Ø7,  USED Ø:Ø2:Ø8.6 IN Ø:48:Ø6(%)
                  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.

Chapter 2

LARGE-SCALE EDITING

The previous chapter 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 chapter 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 chapter are the framework of TECO; the commands in this chapter 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(%) Here (↑D)(%) There (↑D)	Replaces the next "Here " which begins a line with a "There ".
-3R@#!(↑D)(↑D)	Replaces the past three "@#!" strings with nothing; that is, deletes them.
R(%) (↑D)(↑D)	Deletes the next End-of-Line.
Ra page(↑D)a(%) page(↑D)	Starts a new line between 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.

*Sfield(↑D)$V(ESC)$(%) a great battlefield(%) (%)	The user finds a "field", checks, and sees that it is the wrong one.
*Sfield(↑D)$V(ESC)$(%) field as a(%) (%)	The user finds the next "field", and makes sure it is the right one.
*I,(↑D)$V(ESC)$(%) field, as a(%) (%)	He inserts a comma, and checks his work.
*Rbefoer(↑D)$before(↑D)$V(ESC)$(%) remaining before us,(%) (%)	The user corrects a typing error.
*JSbefoer(ESC)$(%) ?SEARCH?35(%) JSbefoer$(%)	The user checks for another instance of this error, but there is none.

The last Search Command in the dialog produced an error message; nevertheless, it illustrates a useful and legitimate application of the Search Command. The user wanted to know if there was an instance of "befoer" in the buffer, and the error message supplied the answer "no". A Search Command which fails does not move the pointer.

MOVING TEXT

It is often necessary to transport a string of text from one place in the editing buffer to another. This operation is required when multiple copies of a given string must be made or when a string must be moved which is too long to be conveniently deleted and retyped. The Q-Registers and their associated commands are used for this operation.

There are 37 Q-Registers, and each of them can hold a character string of virtually unlimited length. Each Q-Register has one of the 26 letters or the 10 digits or @ as its name, and all the Q-Register Commands (except one) end with the name of a Q-Register. The upper and lower case forms of a letter are equivalent as a Q-Register name, just as they are when used in a command name.

Setting a Q-Register

There are two commands for moving a substring of the buffer into the Q-Register. The Extract-String Command (m,nXq) extracts from the buffer everything from just after the m-th character to just after the n-th character. The Extract-Lines Command extracts everything between the pointer and the beginning of the n-th line after the current line. Each command removes the selected string from the buffer and places it in the Q-Register q. The pointer is left where the extracted string was, and the previous contents of the Q-Register are lost.

HXF	Extracts the contents of the entire buffer (leaving the buffer empty) and puts it in Q-Register F.
0,23XA	Extracts the first 23 characters and places the sequence in Q-Register A.
5XX	Extracts a substring and puts it in Q-Register X. The substring extends from the pointer to the beginning of the 5-th line after the current line.
-4X@	Extract the preceding four lines and put them into Q-Register @.
:X1	Extract the remainder of the current line except the end-of-line.

The two Extract Commands move exactly that substring of the buffer which the corresponding Type-String or Type-Lines Command types out. Thus the user can use one of these

Type-Out Commands to simulate an extraction.

Using a Q-Register

The Get-QR Command (Gq) inserts a copy of the character string in Q-Register q into the buffer just before the pointer. The contents of the Q-Register is not changed. When an Extract Command which refers to Q-Register q is immediately followed by "Gq", the total effect is to copy a substring of the buffer into a Q-Register without deleting it from the buffer.

The ;Type-QR Command (Qq;T) types out the complete character string contained in Q-Register q. It can be used to check on the successful execution of an Extract Command. This is the one Q-Register Command which does not have the Q-Register name, q, at the end of the command.

GA	Inserts a copy of the contents of Q-Register A into the buffer just before the pointer.
5,62X3G3	Extracts characters 6 through 62, puts them in Q-Register 3, and copies Q-Register 3 into the buffer. Leaves the buffer unchanged.
QZ;T	Types-out the character string in Q-Register Z.

Merging Files

The commands just described can be used for very general and extensive manipulations of files -- merging, interleaving, and so on. For example, to merge parts of File A into File B proceed as follows:

-- On listings of Files A and B, mark the parts of File A with the names of Q-Registers, and use these Q-Register names to indicate where the parts are to go in File B.

-- Read File A into the buffer, extract the parts into the appropriate Q-Registers, and delete the remainder of the file.

-- Read File B into the buffer and, for each insertion, position the pointer in the buffer and insert the contents of the appropriate Q-Register.

This is a good procedure for making large-scale patches to any program or document.

An Application

The following dialog places at the beginning of the buffer an introductory sentence which cites the opening words of the text which is already in the buffer:

```
*JSago(ESC)$(%)                 Finds end of the opening.
(%)
*Ø,.XAGA(ESC)$(%)               Extracts and restores
(%)                             the opening.
*QA;T(ESC)$(%)                  Types out the contents
Fourscore and seven(%)          of Q-Register A
years ago(%)

*J(ESC)$(%)                     Moves pointer to the
(%)                             top of the buffer.

*IThe Address begins(%)         Makes up the sentence.
"(↑D)$GAI"(%)
The entire text is:(%)
(ESC)$(%)
(%)

*HT(ESC)$(%)                    Types the result.
The Address begins(%)
"Fourscore and seven(%)
years ago"(%)
The entire Text is:(%)
Fourscore an(DEL)(BEL)(%)
(%)
(%)
*
```

STORING INTEGERS

A Q-Register can also be used to hold an integer value. This value is almost always used to save the position of the pointer in the buffer, but it is not restricted to that purpose. If the user wants to do some integer calculations, he can use Q-Registers as his variables.

Only two commands are required and they are very simple. The Update-QR Command (nUq) loads the integer value n into Q-Register q, destroying the previous contents. The Q-Value Command (Qq) is actually a function. Its value is the contents of Q-Register q, and it can be used wherever an integer value is accepted.

23UA	Puts the integer 23 into Q-Register A.
.U5	Puts the current pointer position (the number of characters before the pointer) in Q-Register 5.
Q5J	Puts the pointer at the position specified by the integer in Q-Register 5.
QA+2UA	Increases Q-Register A by 2.
QA=	Types out the integer in Q-Register A.

The Q-Register Commands for character strings and for integers are not mutually compatible. For example, if a Q-Register is loaded with the character string "398" by an Extract-String Command and an attempt is then made to get its value with the Q-value Function, TECO will object and type an error message. All of the Q-Registers initially contain the integer value Ø (as if a ØUq had been executed).

An Application

In the preceding dialog based on the Gettysburg Address, the first sentence of the Address was extracted. That was relatively easy because the beginning of the sentence had a known position, Ø, in the buffer. In the dialog which follows, a sentence is extracted from the middle of the Address and both ends must be located. Q-Register Ø is used to save the position of the beginning of the sentence while the end is found.

```
*Sdid here.(ESC)$(%)
*V(ESC)$(%)
they did here.(%)
*LT(ESC)$(%)
 It is for(%)
*.UØ(ESC)$(%)
*S.(↑D)$T(ESC)$(%)
 It(%)
*QØ,.XC(ESC)$(%)
*QC;T(ESC)$(%)
It is for(%)
us the living,(%)
rather, to be(%)
...
```

Finds previous period, types end of previous sentence.

Gets to beginning of desired sentence and saves pointer.

Finds end of sentence and checks.

Picks up sentence.

Types out sentence.

LOOPS

Although TECO has general facilities for both conditional and unconditional transfer of control, it is the simple and specialized Iteration Command which is most useful.

In order to repeat any command string n times, (1) place the command string in angle brackets, "<" and ">", and (2) put n in front of the bracketted string. If n is omitted, an approximation to infinity (2↑35) is assumed, and the loop will continue until something in the command string stops it.

`3<R;(↑D),(↑D); V>` Starts at the current position of the pointer and replaces the next 3 semicolons with commas. Types each modified line.

`5<L18<I.(↑D)>>` Puts 18 periods at the beginning of each of the next five lines.

```
J<Ryclepped(↑D)yclept(↑D); V>
```
Corrects all misspellings of "yclept" in the buffer, however many there are.

```
J<S(↑S)command(↑S)(↑D); V>
```
Types out every line in which the word "command" appears.

S and R commands inside iteration brackets cause the iteration to terminate if the search fails. Thus, if the buffer contains exactly one occurrence of the string "abc" the command J<Sabc(D)V> will print the line containing the "abc" twice -- once when the search succeeds and again when it fails. This is because the V command is seen before the > which will cause the iteration to stop.

A Q-Register can be used to count the number of times a loop is executed. A special function, %q, is provided which increases the integer in Q-Register q by one and then assumes the resulting integer value. This function can be used as a free-standing command if the user types Control-D after it to "absorb" its integer value.

96UA26<%AI>	Inserts a complete lower-case alphabet into the buffer.
ØUAJ<S;(↑D)%A(↑D)>	Counts the semicolons which appear in the buffer and leaves the result in Q-Register A.

A Final Example

The following loop searches the buffer for the occurrences of the word "command". For each occurrence of the word, the loop types the line in which the word appears, stops to let the user type in a single character, and then capitalizes the words if the user typed "Y" (for "yes").

```
J<Scommand(↑D); V↑T-↑↑Y"E-7CRc(↑D)C(↑D)'>
```

This command string uses some commands which have not been described yet and, in any case, needs some explanation. An annotated listing follows:

J	Puts pointer at beginning.
<	Starts loop.
Scommand(D)	Searches for "command".
; V	Skips to end of loop if search fails. Types out the line containing "command".
↑T-↑↑Y	"↑T" is a function which assumes the value of the code for the next character the user types (TECO waits for the user). "↑↑Y" is the code for "Y".
"E	If the preceding expressions is Equal to zero, the following commands are executed; otherwise, they are skipped up to the apostrophe.
-7C	Back up to just before "command".
Rc(↑D)C(↑D)	Capitalize "c".
'	End conditional expression skip.
>	End loop.

This example is important. It represents TECO at its best, namely in a short, interactive loop in which the user makes the difficult judgements and TECO carries out the editing details.

Chapter 3

SPECIAL EDITING

All the TECO commands which have not been described in the previous chapters are mentioned in this chapter. Each section of this chapter describes a group of commands designed for a particular purpose. The descriptions are brief because the commands are not of universal interest. The purpose of this chapter is to inform the reader of the existence of specialized commands of TECO. A complete description of each command appears in the TENEX TECO Handbook and can be found by looking up the command name in the command index.

Automatic Indentation

An effective technique for organizing information on a page is to use the "outline" form; that is, to indent lines by varying amounts to indicate the grouping and relative importance of the information. This formatting technique is often applied to improve the readability of programs in block-structured languages, such as LISP, Algol, and PL/I.

The simplest means of indenting a line is to type in the appropriate number of leading spaces. For a long program with many different indentations, this process is tedious and error prone. TECO has a set of four special commands which can be used to supply these leading blanks automatically. The commands are:

(↑W) Records the number of spaces in a new indentation.

(↑U) Inserts spaces required for the indentation which is currently in use.

(↑B) Reverts to the indentation which was recorded just before the current one and inserts the required spaces.

(↑Y) Reverts to the indentation which was recorded just after the current one and inserts the appropriate spaces.

The first time a particular indentation is required, the user types in the necessary spaces himself and uses the Control-W command to record the indentation. Indentations are recorded in a list which is a special part of TECO storage.

Logical Operators

An integer may be represented as a sequence of binary digits (as the reader may know). TECO has operators which convert an integer into a bit string, apply a logical operation to the strings, and convert the result back to an integer. The operators are:

m#n the i-th bit of the result is the logical "or" of the i-th bits of m and n

m&n the i-th bit of the result is the logical "and" of the i-th bits of m and n.

Conversion of Integers

In some cases it is useful to convert a sequence of digits which occurs in the buffer into an integer argument for a command. Conversely, it may be useful to convert an integer argument into a character-string representation of the integer. The necessary commands are:

n;N This function interprets the digit string which follows the pointer and assumes that integer value. The argument n specifies the base to be used in interpreting the digits (for example, n=8 for octal digits).

n\ This command expresses its argument, n, as a (possibly signed) sequence of digits of base 10 and inserts the sequence into the buffer just before the pointer.

Flow of Control

TECO has a rather complete set of commands for flow of control. These commands are supplied because even a short command string occasionally needs a conditional transfer or a custom-made loop to make it go. The following commands provide conditional execution of an arbitrary command string:

n"Ec' Executes the command string c if n Equals Ø; otherwise, skips over c.

n"Nc' Executes c if n Not-equals Ø.

n"Lc' Executes c if n Less-than Ø

n"Gc' Executes c if n Greater-than Ø

n"Cc' Executes c if n is the character code of a letter, digit, ".", "$", or "%".

The following commands provide an unconditional transfer of control:

!s! This is a label.

Os(↑D) This is a transfer to the label "!s!".

Sometimes it is not appropriate to have a Search Command produce an error message when the search fails. This is the case, for example, when a search is part of a stored program. The following alternative is provided:

:Ss(↑D) The ":" before a Search Command causes the whole command to assume an integer value of -1 or Ø according as the command succeeds or fails. Thus the command can be used wherever an integer argument is accepted. Note: Search commands and Replace commands inside iteration brackets (< ... >) act as if they have the : modifier on.

n;(SP) The ";(SP)" causes a skip out of the enclosing loop for non-negative values of n and otherwise is ignored. (The semicolon must be followed by a Space character.) If a ":"-Search Command is used as the argument to this command, the command will skip out of the enclosing loop when the search fails.

Stored Programs

TECO has important commands which make it possible to create a TECO program, store it, and later execute it. Specifically, the command string is typed into the buffer like any other passage of text, is placed in a Q-Register by an extract Command, and is then executed by the Macro Command, as follows:

Mq This command causes TECO to execute as a command string the contents of Q-Register q.

Since the character string in the Q-Register q may itself contain a Macro Command, this command provides for subroutines which can be called through one another as well as directly by the user.

A minor difficulty arises in preparing programs. An ordinary Insert Command cannot be used to enter into the buffer an Insert or Search Command because the Control-D which is a part of the stored command will prematurely terminate the overall insertion. The following commands solve this problem:

@Itst This command inserts the character string s, which may include (↑D). Any character which does not appear in s can be used as the character t which delimits the character string.

@Stst This command searches for the character string s, which may include (↑D).

@Rtslts2t Replace string s1 by s2, both of which may contain the terminator t.

TECO has a few additional commands whose principal use is in stored programs. They are:

↑T This is a function whose value is the integer code for the next character typed in by the user. (TECO waits until the character is typed.)

;Ts(↑D) Types out s and is used to output a message from a running program.

[q Pushes down into a special pushdown stack the current value of Q-Register q.

]q Pops up the pushdown stack into Q-Register q.

? Causes TECO to "trace" its execution; that is, to type out commands as they are executed. The second "?" turns the trace off, the third turns it on again, and so on.

The basic input/output commands of TECO, as described in Chapter 1, obtain a file designator not from the command string but from a special dialog with the user. This operation is not appropriate for use in a stored program, and the following command sequences can be used instead:

;Rf(↑D);Y
Reads in all of file f and adds it to whatever is in the buffer.

;Wf(↑D);U
Writes out the entire buffer onto file f and clears the buffer.

These commands are further described in the discussion of paged input/output which follows this section.

The stored program commands and the rather complete set of flow of control commands combine to make possible some relatively complicated symbol manipulation. However, the proper use for the programming facility is to bridge the gap between simple editing and full scale symbol manipulation. TECO should not be used in competition with complete programming systems like LISP or SNOBOL. TECO does not have the debugging facilities, the data structures, or the efficiency to support such an activity.

TECO Paging

The computer on which TECO was originally implemented did not have virtual memory and its actual memory was relatively small. It was therefore necessary to devise a means by which a file could be edited piece by piece. Toward this end, the Form Feed character (Control-L) was selected as an "End-of-Page" character and each substring of a file which ended with a Form Feed was called a TECO page. (There is no relation between this "TECO page", which is purely a software notion, and the "page" which is the unit of the TENEX virtual memory.)

With the introduction of virtual storage and the consequent very great increase in the capacity of the buffer, the need for paging declined. For TENEX TECO, the possibility of the division of a file into TECO pages arises under the following conditions:

-- When a file exceeds the (approximately) one-million-character capacity of the buffer, it must be broken into parts.

-- When a file is to be merged with another file or rearranged in some way, it must be broken into parts.

-- When a file exceeds 60,000 characters (a very rough estimate) and will be subjected to extensive editing (that is, many insertions and deletions), efficiency considerations suggest that it should be broken into parts.

However, a decision to break a file into parts does not necessarily lead to paging the file. It will often be more appropriate to maintain each part of the file as a file in itself. Thus the cases in which paging commands are essential is rare.

For LISTing or TYPEing it is useful to have ↑Ls on at logical places to make page boundaries simply for human use. Also can be handy in moving through file using n;BJ.

The commands for handling paged files will now be described briefly. They fall into several subgroups according to the steps of the input/output process.

The first step is to open files for input and/or output, as follows:

;R*f*(↑D) Selects the file whose designator is *f* and opens it for input.

;W*f*(↑D) Selects the file whose designator is *f* and opens it for output.

The following commands read a page from the file which is open for input.

Y Deletes the contents of the buffer and reads the next page of the input file into the buffer.

A Adds the next page of the input file to the end of the current contents of the buffer.

The following commands output a portion of the buffer which is specified by its arguments. Two, one, or no arguments can be used, and their significance is described in the Handbook.

PW Does output without modifying the buffer.

W Does output and deletes from the buffer the portion which is output.

The following commands perform a combination of input and output. The ;Y and ;U Commands which appear below were discussed in Chapter 1; but here they have a different interpretation because they are used when an input or output file is open.

P Outputs the current contents of the buffer and then reads in the next page of the input file. In effect, the command "turns a page" of the file being processed.

;Y Reads in the remainder (however many pages) of the input file and adds it to the end of the buffer.

;U Repeatedly executes P (Page-Turn) Commands until the remainder of the input file has been copied into the output file and then closes the output file.

The following command concludes the output process:

;C Closes the output file. (For certain technical reasons, the output file is not permanently saved until it is closed.)

Two commands are provided to search an entire paged file in a single operation. Each command starts at the position of the pointer (as usual) but does not stop at the end of the buffer; instead, subsequent pages are read from the input file and scanned until a match is found or the end of the file is reached. The commands are:

nFs(↑D)
Searches page after page. After a page has been searched unsuccessfully, it is written on the output file so that nothing is lost. This command is used when a file is being modified.

n;Fs(↑D)
Also searches page after page. However, after a page is searched unsuccessfully, it is discarded. This command is used when the file is being examined but not modified.

In contrast to the Search commands just mentioned, the indices described here are designed for use with a multiple-page file which is entirely in the buffer; no automatic input/output is involved.

;B Supplies the number of characters in the buffer before the current page. The current page is the page which contains the character just before the pointer.

;Z Supplies the number of characters in the buffer through the end of the current page.

n;B Supplies the number of characters in the buffer before the beginning of the n-th page in the buffer.

n;BJ Jumps to start of n-th page.

Quoting Control Characters

Certain control characters can be used literally in the character string argument of an Insert or Search Command when they are properly quoted. The commands for this purpose are:

(↑V) In most cases, when this character is used immediately before a command control character it will cause that character to be entered into the command register literally.

(↑V)(↑Q) When this character pair is used before one of the three match control characters (as used in a search command), the match control character is interpreted literally.

Complete instructions for quoting characters on input, recognizing them on output, and specifying them in a search are given in the Character Set Appendix.

Abbreviations

There are a number of commands in TECO which can be described as abbreviations for commonly used command strings. Since TENEX TECO is very concise in any case, abbreviations do not play an important role and most of them are of limited interest. They are:

EDIT f This is an Executive command which is an alternative to the "TECO" command. It enters TECO and then causes the file f to be read into the buffer. For a subsequent editing of the file in the same TENEX session, the user can just type "EDIT".

EG This is the exit from TECO which is used when TECO has been called automatically by some other subsystem of TENEX.

(LF) This Command (the Line Feed character) moves the pointer to the beginning of the next line and types that line. The command must be the first character in a command string. (The command was imported from DDT). Note: (LF) and (↑H) are done immediately (i.e. (ALT) is not needed).

(↑H) This command moves the pointer to the beginning of the previous line and types that line. The command must be the first character in a command string. (The command was imported from DDT).

↑↑c This is a function whose value is the integer code of the character c.

;P This is a function whose value is the integer code of the current character and which moves the pointer one character to the right. It thus combines a "1A" Function with a "1C" Command.

B This is an index which is always 0. It is useful because "B" is slightly easier to type than "0".

(SP) This operator (space Character) is equivalent to the "+" operator. It is easier to type than "+".

(HT)s(↑D)
This command can be used to insert a string which begins with a Tab (HT) character. In effect, TECO supplies an "I" at the beginning and makes this into an Insert-String Command.

This completes the list of useful TECO commands. The Command Index contains some other commands; they are either obsolete names which have modern synonyms or are characters such as End-of-Line or "$" which have no effect when they appear as TECO commands.

PART II

HANDBOOK FOR TENEX TECO

Chapter 4

THE DATA

A good way to begin the study of a programming system is to forget the commands, for the moment, and concentrate instead on the data. This chapter follows that approach, describing both the values on which TECO operates and the buffers and registers in which these values are stored. An important supplement to this discussion is Appendix B, which describes the ASCII character set in detail.

DATA TYPES

TECO manipulates two types of data, namely,

-- The character string, a sequence of any number (possibly zero) of ASCII characters, and

-- The coded integer, an integer value with magnitude less than 2↑35.

The text which TECO edits is, of course, a character string value. Less obviously, the command string by which the user controls TECO is also a character string value. The counters and indices for character string manipulation, and the repetition counts for loops are coded-integer values.

Character strings and coded integers have distinct internal representations and this is reflected in the design of the TECO commands. Commands designed for character-string values do not work on coded-integer values and vice versa.

DATA STRUCTURES

In this section, several data structures are defined which are frequently applied to character strings. In all cases, the structuring is not "built-in" to the data, but rather is attributed to the data by particular commands. For example, a command which handles TECO lines sees a character string as divided into lines by occurrences of the End-of-Line character; but a command which deals only in characters does not see any division into lines, and views the End-of-Line character as just another character. The structure exists only in the eye of the beholder.

The following are parallel definitions for the line and the page in TECO:

-- Any character string can be divided into TECO lines by ending a line (1) immediately after each End-of-Line character and (2) at the end of the given character string.

-- Any character string can be divided into TECO pages by ending a page (1) immediately after each Form Feed character and (2) at the end of the given character string.

These data structures are used to achieve two quite separate results: the formatting of type-out and the logical division of data.

The main storage of TECO is the buffer, which will be described shortly among the other storage registers of TECO. The buffer stores a single character string which TECO edits. A useful adjunct to this editing is a pointer, which is a fictitious sliver positioned between two characters of the buffer (but not taking any space itself) and which can be moved from one position to another by certain commands. Useful definitions of "current" objects are made with respect to this pointer, as follows:

-- The current character of the buffer is the character just to the right of the pointer. If the pointer is at the end of the buffer, there is no character to the right of the pointer, and the current character does not exist.

-- The current line of the buffer is the TECO line which contains the current character. In the special case that the pointer is at the end of the buffer, the current line is everything back to (but not including) the last End-of-Line character.

-- The current page of the buffer is the TECO page which contains the current character. In the special case that the pointer is at the end of the buffer, the current page is everything back to (but not including) the last Form Feed character.

When the pointer is at the end of the buffer and TECO needs a current character, TECO assumes the Null character, ASCII code Ø. When the pointer is at the end of the buffer and the last character of the buffer is an End-of-Line or Form Feed character then the current line or page, respectively, is an empty string according to the definitions just given. In this case, TECO performs the required operation (output, deletion, or so on) on this empty string.

DATA STORAGE

Each storage unit of TECO will be described in terms of a little table. To start with, those registers which are under the direct control of the user and which are thought of as the user's data will be given.

MAIN BUFFER

Name: H

Value: character string

Purpose: To hold the text being edited by TECO.

Set by: Input, Insert, Replace and other Commands.

Used by: Output, Search, Replace, and other Commands.

Typed by: HT

POINTER REGISTER

Name: .

Value: coded integer

Purpose: To hold an integer which is the number of characters before the pointer.

Set by: The Position-pointer, Search, and other Commands.

Used by: All position-relative commands and, under its name ".", wherever a coded integer argument is accepted.

Typed by: .=

COUNT REGISTER

Name: Z

Value: coded integer

Purpose: To hold an integer which is the number of characters in the entire main buffer.

Set by: Any command which inserts or deletes from the main buffer.

Used by: Wherever a coded integer argument is accepted.

Typed by: Z=

Q-REGISTERS (A TOTAL OF 37 REGISTERS)

Names: A, B, C, ..., Z, Ø, 1, 2, ..., 9, @

Value: Each register contains a character string or a coded integer.

Purpose: (1) To hold an integer, especially a pointer position, (2) To hold a block of text, or (3) To hold a TECO command string for execution as a subroutine.

Set by: The X Command (for character string) or the U Command (for coded integer).

Used by: The G Command (for character string) or the Q Command (for coded integer).

Typed by: The ;T Command (for character string) or the = Command (for coded integer).

Executed by: The M Command (for character string).

Q-REGISTER PUSHDOWN STACK

Values: This is a stack of values, each a character string or a coded integer.

Purpose: (1) To save for later restoration the value of a Q-Register, and (2) To pass on an argument value to a TECO subroutine.

Set by: The [(pushdown Q-Register) Command.

Used by: The] (pop Q-Register) Command.

Note: The stack is cleared every time TECO returns to the await-commands state; it is therefore useful only when used by a stored TECO program.

COMMAND STORAGE

In addition to the data registers just described, TECO has three registers which are used to hold the command string. The contents of these registers cannot be manipulated directly by the ordinary TECO commands, but they are nevertheless accessible to the user.

COMMAND REGISTER

Value: character string

Purpose: To accumulate the command string as TECO accepts it, character by character, from the type-in buffer, and hold it until it is ready to be executed.

Set by: Type-in from the user terminal and the action of the ↑A and ↑Q Commands.

Used by: The TECO command interpreter.

Typed by: The ↑R Command.

Executed by: The (ESC) Control Command.

COMMAND REGISTER PUSHDOWN STACK

Values: This is a stack of character-string values.

Purpose: To save the current command string when it calls a subroutine (by the M Command).

Set by: TECO, when a M Command is begun.

Used by: TECO, when a M Command is completed.

COMMAND REGISTER BACKUP REGISTER

Value: character string

Purpose: To retain the most recent command string which is 16 characters or more in length.

Set by: TECO from the command register.

Used by: The ;G Command (to put the command string into the main buffer).

INDENTATION CONTROL STORAGE

TECO has a specialized facility for controlling indentation from the left margin which is intended for use with such block-structured languages as LISP, Algol, and PL/I. The data required for these commands is as follows:

INDENTATION RING

Values: A list of 16 coded integers

Purpose: Each item in the list specifies the column number at which a line should begin when that entry is selected by the Indentation Selector.

Set by: The ↑W Command.

Used by: The other Indent Text Commands.

INDENTATION SELECTOR

Value: pointer to item in Indentation Ring.

Purpose: To select one of the 16 entries in the Indentation Ring.

Set by: The Indent Text Commands except ↑U.

Used by: All of the Indent Text Commands.

TENEX BUFFER STORAGE

Finally, there are several buffers which belong to TENEX rather than TECO which perform functions which are of interest to the TECO user. They are the following:

TYPE-IN BUFFER

Value: character string

Purpose: To hold incoming characters from the user terminal until TECO is free to echo them and interpret them.

Set by: Type-in from the user terminal.

Used by: TECO.

Note: This register is mentioned because its presence allows the TECO user to continue typing even when TECO is too busy to accept the type-in.

INPUT/OUTPUT BUFFERS

Value: character string

Purpose: To hold characters transmitted between TECO and the TENEX files until they can be accepted at their destination.

Note: These buffers are mentioned because, as will be seen in the next chapter, a too-sudden exit from TECO can cause the loss of or duplication of the contents of these buffers.

Chapter 5

THE COMMANDS

The remaining chapters of this Handbook are a description of the commands of TENEX TECO. This chapter includes only that information which applies to all the commands; namely, (1) a description of the organization of the Handbook, and (2) a general discussion of command syntax. Chapter 6 contains the Control Commands, a small number of single-character commands by which the user directs TECO to interrupt, execute, edit, and format the Ordinary Commands which are being typed in. Chapter 7 gives the Ordinary Arguments; that is, all the operators, functions, indices, names, and constants which can be used as arguments in Ordinary Commands. Finally, Chapter 8 lists the Ordinary Commands; that is, the numerous and varied commands which are available for the actual business of text editing.

The Organization of the Command List

This Handbook provides three separate access paths to the description of a command. They are as follows:

-- Each command can be located by looking up its name in the Command Index. For example, the Search Command (which will be cited throughout this chapter) is indexed under "S".

-- Each command can be located by looking in the function group in the appropriate chapter. The function groups are listed in the Table of Contents. For example, the Search Command is the first command in the Search Group in the chapter on Ordinary Commands.

-- Each command can be quickly assessed by its relevance indicator, which appears in a conspicuous position in the command description. This expedites a casual scan of the Handbook. For example, the Search Command has relevance indicator "****", which means it is of general interest.

The relevance indicators provide a somewhat unusual summary of TENEX TECO. They are defined in the following list.

-- "****". Applies to a command which is useful for all editing activities. An alternative to reading the entire Handbook is to read only those commands which are marked "****". Covers about 1/2 of the commands.

-- "paging". Applies to a command which uses the division of a file into TECO pages. Such a command is used in the rare case that (1) a file is very large or (2) extensive interleaving of several files is required. The applications are discussed at length at the beginning of the Complicated Input/Output group. Covers about 1/8 of the commands.

-- "programs". Applies to a command which is used only in the writing or executing of a stored TECO program, which is an unusual activity. Covers about 1/8 of the commands.

-- "special". Applies to a specialized command other than those already covered by the "paging" or "programs" indicators. Covers about 1/8 of the commands.

-- "abbrev". Applies to a command which is a simple abbreviation of a specialized sequence of commands. Such a command is valuable to the large-volume user of TECO. Covers about 1/16 of the commands.

-- "obsolete". Applies to a command which has an exact equivalent in a more modern (and possibly more general) command. Covers about 1/16 of the commands.

The Organization of a Command Description

A command description is the following list of items, some of which are optional:

-- The headline is always given. It is a single line of concise and important information about the command, and contains:

   -- The syntax specification (which is discussed later),

   -- The relevance indicator (already described),

   -- A brief description of the action of the command, and

   -- A mnemonic informal name.

   For example, the headline for the Search Command is:

   nSs(↑D) *** (search for string) "Search"

-- The where-clauses give definitions for the syntactic variables (if any) which appeared in the syntax specification of the headline. For example, the Search Command requires the definitions "where n is an integer expression greater than zero, and where s is a character string of less than 72 characters which does not contain the character entered by (↑D)."

-- Various equivalences and defaults are given as appropriate. If the command is obsolete, its modern equivalent is given; or, if an argument can be omitted, the default value assumed is given. For example, for the Search Command, the integer argument can be omitted and, for that case, the default "Ss(↑D) means 1Ss(↑D)" is given.

-- The main description of the command, in plain English, is given.

-- Whatever Error Handling conventions or Warnings apply to the illegal or dangerous use of the command are given. Examples of both of these appear in the main listing for the Search Command. Finally,

-- Some examples of the command are given. These are usually intended to clear up difficult or ambiguous points, and are not a complete list of the uses of the command.

The Syntax Specification

The syntax specification with which a command description begins is composed of syntax variables and literals, as follows:

-- A syntax variable is a single lower case letter which is underlined. It is always defined by a following where-clause and may be subject to various special conditions.

-- Anything which is not a syntax variable is a literal. Some literals are more literal than others. The graphic characters are given just as they would appear, but the control characters, whose function is something other than typing a single symbol, are represented by parenthesized names. These distinctions are clearly made in the Character Appendix to this Manual.

The syntax specification for the Search Command is

nSs(↑D)

This contains the syntax variables n and s, the graphic character "S", and the control character designated by (↑D). As the Character Appendix points out, when the user types a Control-D, the TENEX TECO actually types out a "$", and not (↑D).

When TECO is interpreting a command name, it does not distinguish between upper and lower case letters. For example, the Search Command can be written with a small "s" as its command name.

Finally, as the examples above have already shown, control characters are represented in this Handbook by parenthesized mnemonic abbreviations, such as "(↑D)" for the type in of a Control-D.

Chapter 6

THE CONTROL COMMANDS

At its simplest, the control of TECO cycles through the following four steps over and over again:

-- TECO types "*", asking for commands;

-- The user types a sequence of commands;

-- The user types Escape; and

-- TECO executes the commands.

However, various conveniences soon arise, and the control of TECO becomes more complicated. This chapter describes a special class of TECO commands, the Control Commands, and relates them to the remainder of the TECO commands which, in accordance with their role as the useful and productive elements of TECO, are called Ordinary Commands.

Every character typed by the TECO user is a command or part of a command; but the Control Commands are obeyed in a more direct and immediate way than the Ordinary Commands. There are only a small number of Control Commands in TECO, and each of them is a single character. In contrast, there are about a hundred different Ordinary Command and Argument forms in TECO, and an Ordinary Command can vary in length from a single character to thousands of characters.

Each Control Command is a single one of the non-printing ASCII characters called Control Characters. There are fifteen Control Commands, as follows:

Interrupt: (↑C), (↑T), (DEL), and (↑E);

Execute: (ESC);

Edit: (↑A), (↑H), (↑Q), and (↑R);

Quote: (↑V), and (↑Q); and

Indent: (↑W), (↑U), (↑B), and (↑Y).

These commands will be discussed in this chapter in five Groups as indicated by their arrangement above.

The path of a single character will now be traced from the time it enters the TENEX system until the time it is executed as all or part of a command.

-- An Interrupt goes through immediately.
The character is accepted from a terminal by the TENEX system, which supervises and supports TECO in its interaction with the user. TENEX looks at the character and acts as follows:

   -- If the character is an interrupt Command, TENEX immediately acts in co-operation with TECO to obey the command;

   -- Otherwise, TENEX places the character at the end of the type-in buffer. The character is not echoed at this point, so the user has not yet gotten anything back for his key-stroke.

-- Anything else waits for TECO.
TECO usually processes characters without delay; however, if TECO is finishing up a previous command (or if the system is heavily loaded) the characters accumulate and wait in the input buffer.

-- TECO does a Control Command.
When TECO is ready for the character, it takes it from the beginning of the input buffer, looks at it, and acts as follows:

   -- If the character is a Control Command, then TECO executes it immediately;

   -- Otherwise, TECO accepts the character as part of an ordinary Command, sends the echo (type-out) back to the user's terminal, and adds the character to the end of the command register.

-- Finally, Escape triggers Ordinary execution.
A character of an Ordinary Command waits in the Command buffer as other characters accumulate to form a sequence of Ordinary Commands. When an Escape is entered it causes TECO to execute the whole command string.

INTERRUPT

The TECO user is supplied with four Interrupt Commands. One, Control-C, allows him to leave TECO and go to the TENEX EXECutive and then take up in TECO where he left off. The second, Control-T, provides the user with information on the status of TENEX. The third, Delete, aborts whatever command TECO is doing and allows the user to enter a new string of TECO commands. The last, Control-E, is an obsolete equivalent of Delete.

↑C *** (interrupt TECO) "Go-to-EXEC."

The Control-C Command has the effect of interrupting any TECO command in a relatively non-destructive way and proceeding to the TENEX EXECutive command level. There are two cases to be considered:

-- If TECO is not doing input/output, a single Control-C causes the following actions:

   -- TENEX stops TECO immediately;

   -- the Control-C is echoed with "↑C" and End-of-Line; and

   -- TENEX turns control over to the Exec, which types an "@" to request a command.

   On the other hand,

-- If TECO is typing out, the interrupt does not occur until the output buffer is empty. If the user types Control-C for a second time, however, TENEX will immediately clear both input and output buffers whatever their contents and honor the interrupt in the manner described above.

The TECO user may find the double Control-C useful when he wants to exit to the EXEC during a type-out at his terminal. However, he should not use the double Control-C under other circumstances since, if TECO happens to be doing input/output with a permanent file, an unpredictable portion of the file may be lost.

The Control-C interrupt can be used according to the following sequence:

-- Interrupt TECO with one or two Control-C's, as mentioned above.

-- Perform only those commands at the EXEC level which do not modify the user's address space, such as typing, deleting files, and checking the state of the system.

-- Return to TECO by means of the Exec Command CONTINUE.

The net effect of this sequence will be nil from TECO's point of view. Note however, that some typed input characters (less than 64) may be lost.

The effect of some system crashes is equivalent to one or two Control-C Interrupts. If the user was in TECO before the crash and finds himself in the EXECutive Level when the system is re-established, he should try recovering with the CONTINUE Command.

(↑T) *** (state and time report) "Time-Check"

The Time-check Command interrupts TECO, types a brief message on the status of TECO, and allows TECO to resume at the point of interruption. The message includes the activity of TECO (running or waiting for input), the location of the machine instruction which was interrupted (of interest only to TECO implementers), the TENEX System load average, and a statement of the total time used in the current TENEX session.

(DEL) *** (abort command string) "Abort-Coms."

The Delete Command has the effect of interrupting any TECO command and placing TECO in its basic await-commands state. There are two cases to be considered:

-- If TECO is executing a command, a single Delete causes the following sequence:

   -- TECO immediately stops whatever it is doing;

   -- TECO echoes with a Bell (if TECO was typing out) or with two End-of-Line characters and the message "ABORTED" (if TECO was not typing out), or gives "?search?35 xxxxx" - when doing Search or Replace.

-- TECO deletes the contents of both the input buffer and the command register;

-- TECO types an End-of-Line and "*" to request a new command.

-- If TECO is waiting for input, a single Delete is echoed by the Bell. In this case the bell is asking whether this Delete was a (dangerous!) typing error or a real abort command. If the user wants an abort, he types another Delete and the sequence given above occurs. If the user has made a typing error or changed his mind, he continues typing with any character other than Delete and TECO forgets it saw the Delete.

The type-out from TECO also passes through a TENEX buffer, and because of this the Delete may sometimes appear to behave incorrectly. Specifically, the user may type Delete during type-out and get a response from TECO characteristic of interruption when TECO is <u>not</u> typing out. This is because TECO has already sent <u>its</u> output off to the output buffer when the Delete was pushed, and was doing other things.

(↑E) obsolete (abort command string) "Abort-Coms."

<u>equivalent to</u> (DEL)

EXECUTE COMMAND STRING

The single command in this group causes the currently entered command string to be executed. The basic interpretation cycle which TECO applies to Ordinary Commands is included in the description of this command.

(ESC) *** (execute command string) "Execute-Coms."

The Escape Command takes the following preliminary action:

-- If there are 16 or more characters in the previously executed command register, a copy is written into the command register backup (and the previous contents of the backup are lost).

Then the command repeats the following command interpretation process until the entire command has been scanned, until an erroneous command has been encountered, or until an abort occurs:

-- Any numeric arguments encountered are scanned, evaluated, and saved;

-- The command name itself is encountered and recognized;

-- If the command requires one or two suffix arguments, the arguments are scanned and saved;

-- The command is applied to the accumulated arguments. Then,

  -- If the command is successfully executed, TECO continues its scan through the command register by restarting the command interpretation process; otherwise,

  -- If the command was illegal, either in its form, its arguments, or its action on TECO data, TECO responds as follows:

    -- TECO types an error number (listed in the Error Message Appendix of this manual), an English error message, or both; this type-out will appear on one or more separate lines; then,

-- TECO types the ten preceding characters of the command register up to and including the entire illegal command; next,

-- TECO clears the command register and the input buffer; finally,

-- TECO types an End-of-Line and a "*", requesting a new command string.

When TECO has successfully executed a complete command string, TECO

-- clears the command register and the Q-Register pushdown stack,

-- types an End-of-Line character and a "*", and

-- turns to the input buffer to process any characters which have accumulated.

EDIT COMMAND STRING

The Control Commands which are in this group constitute a small but very useful editor for correcting errors in a command string which has been entered but not yet executed.

The first two commands, Control-A and Control-Q, erase the last character or the last line, respectively, from the command string which is being entered. The two keys "A" and "Q" were not chosen for their names, which are not at all mnemonic, but for their positions on the keyboard, which are adjacent to the CTRL key. Each command thus can be entered by a single motion of the left hand.

These commands can be used in any combination and sequence; and if they are applied often enough, they can erase a command string of any length. The Control-A can erase any ASCII character, including the End-of-Line character; so it can be used to erase past the beginning of the current line of type-in and on into the preceding line.

The third of these commands, Control-R, is used to get a type-out of the last line of the command string being entered. This is useful after the user's image of the line has been confused by applications of Control-A and Control-Q, by line noise, or by some other interruption. TECO leaves the typing element just after the last character typed out, so the user can resume typing just as if he had typed the corrected line himself.

(↑A) *** (erase last character) "Erase-Character"

TECO responds to Control-A by erasing the last character of the command register; specifically,

-- If the command register is not empty, TECO

  -- deletes the rightmost character and

  -- types a "\" followed by the echo of the deleted character;

-- If the command register is now empty, TECO types an End-of-Line and "*" (requesting a fresh start).

(↑Q) *** (erase last line) "Erase-Line"

TECO responds to Control-Q by erasing the last line, complete or incomplete, of the command register; specifically,

-- If the command register is not empty, TECO deletes the last (non-empty) TECO line in the command register;

-- TECO reports the state of the command register as follows:

   -- If the command register is not yet empty, TECO types "←" and an End-of-Line character (presenting the user with an empty line); however,

   -- If the buffer is now empty, TECO types two End-of-Line characters and a "*" (indicating a fresh start).

(↑R) *** (retype last line) "Retype-Line"

TECO responds to Control-R by typing out the last line, complete or incomplete, of the command register; specifically,

-- If the command register is not empty, TECO types the last (non-empty) TECO line in the command register; otherwise,

-- If the command register is empty, TECO types an End-of-Line character.

QUOTE A CHARACTER

It is sometimes useful to include in an Ordinary Command a character which happens to be a Control command. For example, the user may wish to use an Insert-String Command to insert the character in the editing buffer. TECO normally frustrates this attempt by executing the Control Command before it reaches the command register. The Quote Command described here protects any non-Interrupt Control Command from this fate, and delivers it safely to the command register. The Interrupt Control Commands must be handled by other means (the Insert-Code Command).

This group also includes the Quote-Match Command, which is used in Searches, and the Control-D command, which is used to terminate the character string argument in an Ordinary Command.

(↑V) special (quote control character) "Quote"

The command always combines with the first character which follows it which is not an Interrupt Control Command. Its effect is to enter a single character in the command register, thus adding that character to the Ordinary Command which is being accumulated, as follows:

-- An Interrupt Control Command bypasses the quotation mechanism. It is neither quoted nor does it "use up" a Quote Command.

-- A non-Interrupt Control Command is protected from its usual interpretation and is entered in the command register "as itself", that is, as the character code actually generated by the key stroke.

-- A non-control command is entered "as itself" just as it would have been if the Quote Command had been omitted.

(↑V)(↑Q) special (take match literally) "Quote-Match"

When used in the argument string of a Search Command or in the first argument string of a Replace Command, the quoted Control-Q character itself becomes a quoting character and causes a match control character to be

interpreted literally. This command is fully described under its main entry in Ordinary Commands, Search.

When it appears outside of the contexts just mentioned, this command does not have a special interpretation; that is, it is an ordinary application of the Quote Command, Control-V.

(↑D) *** (generate string terminator) "Terminate-String"

This command enters a single Escape character into the command register. The Escape is the standard delimiter for a character string in Ordinary commands. The Escape character cannot be entered directly for this purpose because it is a control command itself.

The user will naturally tend to think of Control-D itself as being the character string terminator, since this is what he types in. This point of view is quite right for all the ordinary TECO applications, and this manual encourages it by referring to the string terminator as "the character entered by (↑D)".

Nevertheless, the conversion can be detected and can affect attempts to deal with Control-D as a character in itself (as opposed to its role as the terminator). The following dialog is a simple example:

```
*HK@I/(↑D)/J;P=(ESC)(%)
27(%)
```

Here the user cleared the buffer, used the special @I command to insert the "character entered by (↑D)", typed out the character code, and got "27" instead of "4".

INDENT TEXT

Programs in a block-structured language like LISP, Algol, or PL/I lend themselves to a formatted arrangement on the page which greatly improves readability. The main feature of this format is the variation of indentation from the left margin. The text within a block is indented more than the surrounding text; and since this is applied recursively to blocks within blocks (and so on), quite a few different indentations may be required.

The desired indentation can be achieved in three ways.

-- First, the user can simply begin each line with the appropriate number of spaces, using the space bar.

-- Second, the user can set the tab stops appropriately (using the Exec Command STOPS) and then use one tab character (Control-I) to get to the first level of indentation, two tabs to get to the second level, and so on.

-- Third, the user can use the commands in this group to automate the insertion of spaces at the beginning of each line.

The commands in this group are handy but take a while to learn; so the choice of indentation method depends on the volume of text to be typed in.

These commands make use of two data structures which are built into TECO for this purpose only. The first is the indentation ring, a list of entries, each of which contains a column number. The second is the indentation selector, which always points to one of the 16 entries in the indentation ring. At any point during type-in, the indentation selector points to the column number at which the user wants a line of text to begin. All entries in the indentation ring are initially 0 (indicating the first column of a line).

Certain commands in this group move the indentation selector from one entry to an adjacent entry. For this discussion, the last entry of the indentation ring is assumed to be immediately followed by the first entry; and so it is called a "ring".

(↑W) special (set column entry) "Write-Indent"

The Write-Indent Command advances the indentation selector to the next entry and stores in that position the column number at which the next character will be typed.

(↑U) special (indent to current level) "Usual-Indent"

The Usual-Indent Command positions the printing element of the terminal so that the next character typed will be in the column, c, contained in the currently selected entry of the indentation ring. Specifically:

-- If the number of characters in the current line is less than c, the command supplies blank characters until the line is c characters long;

-- If the number of characters in the current line is identical to c, the command does nothing;

-- If the number of characters in the current line is greater then c (so the point of indentation has been passed), the command supplies an End-of-Line character (starts a new line) and supplies c blank characters

---

(↑B) special (indent to previous level) "Back-Indent"

The Back-Indent Command moves the indentation selector backward one entry in the indentation ring and then performs the actions of the Usual-Indent Command. The effect is to indent, but less than before.

(↑Y) special (indent to next level) "Yet-More-Indent"

The Yet-More-Indent Command moves the indentation selector forward one entry in the indentation ring and then performs the actions of the Usual-Indent Command. The effect is to indent, but more than before.

WARNING:

Do not use any of these commands in the first line of an Insert-text Command and do not use a Tab character in a line before one of these commands; in both cases, the indentation

will be incorrect. In these cases, the user must type the blanks which these commands would normally supply.

The reason for these restrictions is that these commands are based on the number of characters in the current line of the command register, and not on the actual position of the typing element of the terminal. In the case of the first line of an Insert-text Command, that line begins with an "I" (as well as any commands which precede it on the same line); and this "I" is counted by the Indent Text Commands but does not, of course, appear in the final text. In the case of the Tab character, the character is entered in the command register and counted as a single character; but it is echoed and typed out as the number of blanks required to get to the next tab stop. A use of a Tab character which is not followed in the same line by an Indent Text Command will do no harm.

EXAMPLE:

In the left-hand column which follows, the complete dialog by which an Algol program is formatted is shown; that is, the user's Indent Text Commands are shown explicitly, spaces are indicated ((5s) means five spaces), and the system type-out is underlined. In the right hand column, the dialog is shown just as the user sees it on the listing.

```
*I(5s)BEGIN(%)                          *I      BEGIN
(5s)(↑W)xxx(↑U)(%)                            xxx
(5s)xxx(↑U)(%)                                xxx
(5s)(3s)(↑W)BEGIN(↑U)(%)                        BEGIN
(8s)xxx(%)                                      xxx
LOOP:(↑U)(3s)xxx(↑U)(%)                 LOOP:  xxx
(8s)(3s)(↑W)BEGIN(↑U)(%)                          BEGIN
(11s)xxx(↑U)(%)                                   xxx
(11s)END;(↑B)(%)                                  END;
(8s)xxx(%)                                      xxx
LONGLABEL:(↑U)(%)                       LONGLABEL:
(8s)END;(↑B)(%)                                 END;
(5s)xxx(↑Y)(%)                                xxx
(8s)BEGIN(↑U)(%)                                BEGIN
(8s)xxx(↑U)(%)                                  xxx
(8s)END(↑B)(%)                                  END
(5s)END(%)                                    END
(ESC)(%)                                $
```

Some patterns emerge. It turns out to be convenient to end almost every line with one of the Indent Text Commands, and thus save the trouble of typing Carriage Return. This is possible because when TECO finds that it has passed the column required for indentation, it starts a new line and then indents. The only exception is with a labelled line; in this case, the user did not want to indent at all and

used a Carriage Return. When a program label extends into the part of the line which would normally be occupied by an Algol statement, TECO neatly starts a new line.

The user only had to make three settings of the Indentation Ring (namely, for columns 5, 8, and 11), and thereafter these were called up automatically. Note that the user waited until the second line to give the first Write-Indent Command.

Chapter 7

THE ORDINARY ARGUMENTS

There are two kinds of Ordinary Arguments. The first kind is the integer expression, which is a prefix argument. One or two integer expressions can be used at the beginning of a command, just before the command name. The integer value of such an expression is usually interpreted by the command as a count of the characters or lines in a string, as the number of times a command is to be repeated, or as the integer code for an ASCII character.

The second kind of Ordinary Argument is the non-numeric argument, which is a postfix argument. One or two of these arguments can occur at the end of a command. This argument is always taken literally; that is, it does not evaluate to something else. It is used to name a Q-Register or a file, or to specify a character string.

Most of this chapter is devoted to the various functions and indices which can occur in an integer expression.

INTEGER EXPRESSIONS

Wherever an integer expression is called for in this manual, any of the following forms can be used:

Mathematical Arguments,	
Arithmetic Operators:	m+n m-n m*n m/n +n -n
Logical Operators :	m#n m&n
Parenthesis Pair:	(n)
Numerical Constant:	[unsigned-integer]
Character Functions :	;P 1A ↑T ↑↑c
Special Functions:	n;N \ ↑H s
Q-Register Functions:	Qq %q
Buffer Indices:	. B Z H n;B ;Z

In the expressions just given,

-- m and n are themselves integer expressions;

-- c is an ASCII character;

-- s is a Conditional-Search Command; and

-- q is a Q-Register name.

The "-" can be used by itself as a complete argument (meaning "-1"), and ";N" and ";B" can be used without a prefix argument, n.

The rules for evaluating a TECO numerical expression are simpler than those of conventional mathematics because, aside from the usual effect of parentheses, execution proceeds strictly from left-to-right. That is, in evaluating an expression, TECO repeatedly applies the leftmost operator to its neighboring operands to produce a single new integer value. This is contrary to conventional notation which assigns a "precedence" to each operator and then evaluates in order of precedence. For example, in conventional notation, 2+3*4=2+12=14; but in TECO 2+3*4=5*4=20. Since the TECO user rarely uses an expression with more than one operator, he does not often encounter this behavior; and the order of evaluation can always be specified explicitly by using parentheses. For the example above, one can write 2+(3*4), which is 14 in TECO.

ERROR HANDLING:

Undetected Errors. An ill-formed integer expression is any sequence of integers, parentheses, function names, or indices which does not satisfy the definition just given for the integer expression. In most cases, TECO assigns a value to an ill-formed expression, in one way or another, and does not detect the error.

This behavior is of no interest to the user until he makes an error. However, when a user does inadvertently type in an ill-formed expression, it may be very important for him to know how TECO interpreted the expression. For example, see the discussion of "runaway execution" in the description of the Insert-String Command. TECO interprets an ill-formed expression as follows:

Empty Parentheses. An empty parenthesis pair is assumed to be zero. That is, "()" becomes "Ø".

Unmatched "(". When an expression has one or more unmatched left parentheses, the last unmatched left parenthesis and everything to the left of it is ignored. For example, "(1+(2+(3+4)" becomes "2+(3+4)".

Unmatched ")". When an expression contains an unmatched right parenthesis, TECO types an error message and awaits new commands. (This is the only error TECO detects in the form of an expression.)

Leading Operator. When an expression begins with an operator, a preceding zero is assumed. (This is not an error for "+" or "-".) For example, "*3+4" becomes "Ø*3+4".

Trailing Operator. When an expression ends with an operator, the operator is ignored. For example, "3+4*" becomes "3+4".

Adjacent Operators. When two or more operators occur in sequence, all but the last is ignored. For example, "6+-*/2" becomes "6/2".

Adjacent Operands. With the exception noted below, when two operands appear in sequence a "+" is assumed between them. For example, "(4*5)(6-7)" becomes "(4*5)+(6-7)".

Exception. When an operand is followed by an n-digit integer (signed or not), the first operand is evaluated, n zeros are appended to the result, and the two operands are added. For example, "(4*5)92" becomes "2ØØØ+92" or "2Ø92".

Solitary "#". When a Number Sign is used as the entire argument of a command which takes one argument, it is interpreted as "-1". For example, "#L" becomes "-1L".

MATHEMATICAL ARGUMENTS

There are six mathematical operators, four of them arithmetic operators and the remaining two logical operators. They perform operations on coded integers and require no introduction. The parenthesis pair and unsigned integer are also included here.

---

mon *** (arithmetic operators) "Plus,Minus,Times,Over"

where m and n are integer expressions, and

where o is "+", "-", "*", "/", or the Space character, (SP).

+n means Ø+n

-n means Ø-n

- means -1 (This is allowed as the whole argument of a one-argument command.)

m(SP)n means m+n

The action of plus, minus, and times are in accord with common usage. The division operation is thought of as first producing a signed decimal number from which the fractional digits are then deleted (leaving the "integer part" of the number). The Space character is allowed as an alternative to "+" because it is easier to type than "+".

ERROR HANDLING:

Overflow. The result of any of these operations is undefined if the value of the correct result would be out of the range (-2↑35) through (2↑35-1); that is, the result will be a wrong answer which is not worth specifying precisely. TECO does not report an error in such a case, but instead uses the resulting value.

EXAMPLES:

-5*-4 --- is seen by TECO as -5-4 and equals -9. Always parenthesize to avoid adjacent operators.

492/1ØØ --- equals 4, the integer part of 4.92.

---

mon special (logical operators) "Or,And"

where m and n are integer expressions, and

where o is one of the two characters "#" or "&".

These commands express each of the arguments, m and n, as 36-bit binary representations of their values, apply the logical operators to the bit strings, and finally, interpret the resulting bit string as an integer value. The binary representation used for a negative number is "twos complement" (used by the PDP-10 hardware).

The operators act on the bit strings as follows:

-- m#n means apply the logical "or" operation to the corresponding bits. The i-th bit of the result is therefore "1" if the i-th bit of m or n is "1", and is "0" otherwise.

-- m&n means apply the logical "and" operation to the corresponding bits. The i-th bit of the result is therefore "1" if the i-th bits of m and n are "1", and is "0" otherwise.

EXAMPLES:

QA&1 --- is 0 or 1 according as the coded integer value of Q-Register A is even or odd.

QA#QB --- is zero only if both QA and QB contain zero.

(n) *** (collect operand) "Parentheses"

where n is an integer expression

The parentheses serve their usual purpose; that is, they group together a sequence of arguments and operators which are to be evaluated to produce a single coded-integer value.

ERROR HANDLING:

Missing Left Parenthesis. If an unmatched right parenthesis is encountered, TECO types an error message and awaits new commands.

---

n *** (unsigned-integer) "Unsigned-Integer"

where n is a sequence of one or more decimal digits.

The unsigned-integer is always interpreted as decimal (base 10) when it appears in an integer argument of a command.

ERROR HANDLING:

Out of Range. If the integer is out of the range (2↑35) through (2↑35-1), it is reduced to an undefined integer value in that range. No error message is typed out.

CHARACTER CODE FUNCTIONS

There are four functions in TECO which, when applied to a single ASCII character will produce as the result the value of the ASCII code of the character. The first two are applied to the character which follows the pointer in the buffer. The next is applied to a character solicited from the user's terminal. The last is applied to the character which is the literal argument of the function.

1A *** (get character code) "Access-Code"

The Access-Code Function takes as its value the integer equivalent of the ASCII code for the current character. The current character is the character just to the right of the pointer. If the pointer is at the end of the buffer, this function takes 0 as its result value.

It is convenient to use "1" in front of "A"; but any integer will do. The argument is ignored and serves only to distinguish this command from the Append-page Command, which has no argument.

;P abbrev (get code and adv. pointer) ";Pickup-Code"

The ;Pickup-Code Function takes on the same value as the Access-Code Function. In addition, however, the ;Pickup-Code Function also moves the pointer one character to the right (unless the pointer is already at the end of the buffer). Thus the ;Pickup-Code Function is an abbreviation for a use of the Access-Code in the argument of some command followed by a "1C" Character-Skip as the next command.

↑T programs (get code from user) "Take-Code"

where the name is "↑" followed by "T", not Control-T.

This function causes TECO to pause and wait for a single character to be typed in at the user's terminal; when the character is typed, the function assumes the coded-integer value of the ASCII code of the character, and TECO continues execution.

This is the total input facility for writing TECO programs. It is especially useful for inputting the user's

"Y" or "N" answer to a simple yes or no question; however, used in a suitable loop, it could input any character string required.

↑↑c abbrev (get code from argument) "Lookup-Code"

where the name is "↑" followed by "↑", not Control-↑, and

where c is any ASCII character other than an unquoted control character.

The value of this function is the coded-integer value of the ASCII code of c. For those characters to which it can be applied, this function saves the user the trouble of looking up the ASCII code.

SPECIAL FUNCTIONS

TECO has two functions which each take an integer which is a substring of the buffer and convert it into a coded-integer value. Also included here is a function to measure elapsed time.

---

n;N special (get integer) "Number-Pickup"

where n is an integer expression

;N means 1Ø;N

This function has a coded integer value which is obtained as follows:

-- If the pointer is followed by an unsigned integer in the buffer, the pointer is moved to the end of the string of digits. The integer is interpreted as being base-n and is taken as the result; otherwise,

-- If the pointer is not followed by an integer, the pointer is not moved and Ø is taken as the result.

ERROR HANDLING:

Bad Radix. Error number 53, "Negative argument to :N" is typed.

EXAMPLE:

Suppose the buffer contains the character string "64ABC-78" and the pointer is at the beginning. The following sequence of commands are executed:

;N= --- which types "64";

;N= --- which types "Ø";

3C;N= --- which still types "Ø";

C;N= --- which types "78"; and

J8;N= --- which types "52".

\ obsolete (get integer) "Old-Pickup"

equivalent to 1Ø;N

↑E (form feed flag) "FF-Flag"

where the name is "↑" followed by "E", not Control-E.

Set by reading a page. If reading terminates normally because a ↑L was seen, subsequent ↑E commands will return 1 as a value. If reading terminated because the text buffers filled, ↑E will return Ø as a value.

Q-REGISTER FUNCTIONS

Two functions which obtain the value contained in a Q-Register are mentioned here. The descriptions given are summaries of of the main entries which appear in the chapter on Ordinary Commands under Q-Registers.

Qq *** (evaluate Q-Register) "Q-Value"

where q is a Q-Register name.

The value of this function is the coded-integer value contained in the Q-Register whose name is q.

%q *** (increment and evaluate QR) "Step-QR"

where q is a Q-Register name.

This function increases the Q-Register named q by one and then assumes the coded-integer value contained in that Q-Register.

BUFFER INDICES

TECO maintains indices which each contain a coded-integer value and which are always used in a context in which they represent character counts from the beginning of the buffer. Thus each index specifies a position in the buffer between two characters, at the beginning of the buffer, or at the end of the buffer.

. *** (current location) "Pointer-Value"

This symbol is the name of the Pointer Register and can be used wherever a coded-integer value is required. It contains the number of characters between the beginning of the buffer and the pointer.

B abbrev (index of beginning of buffer) "Begin-Buffer"

equivalent to Ø

This index, B, was introduced for reference to the beginning of the buffer, and is useful because "B" is slightly easier to type than "Ø". It originated in an earlier version of TECO in which the buffer did not begin at "Ø".

Z *** (index of end of buffer) "End-Buffer"

Z represents the number of characters in the entire buffer, and is, therefore, the position of the end of the buffer.

H *** (index-pair for buffer) "Whole"

equivalent to B,Z

This symbol, H, is a very specialized abbreviation which can be used only to replace a pair of coded-integer arguments. It is, in that context, the name of the entire contents of the buffer.

---

n;B paging (beginning of page) "Begin-Page"

where n is an integer expression greater than zero

;B means (see below)

Used without the integer argument, this index is the number of characters in the buffer before the first character of the current TECO page.

Used with the integer argument n, this index is the number of characters in the buffer before the first character of the n-th TECO page.

ERROR HANDLING:

Bad Count. If n is not positive, TECO types an error message and awaits new commands.

Too Big. If n is greater than the number of pages in the buffer, TECO types an error message and awaits new commands.

;Z paging (index of end of page) "End-Page"

This index is the number of characters in the buffer before the end of the current TECO page. ;Z is the same as top of the next page.

NON-NUMERIC ARGUMENTS

Included in this group are the Q-Register name, the file designator, and the character string.

---

c *** (Q-Register name) "QR-Name"

where c is a letter or a digit.

There are 37 Q-Registers, one each for each letter, each digit and @. A capital letter and the corresponding lower case letter refer to one and the same Q-Register.

---

d:<dn>fn.e;vn *** (file designator) "File-Designator"

where d is the device;

where dn is the directory-name;

where fn is the file-name;

where e is the extension; and

where vn is the version-number

The file designator is used to specify the origin of any input to TECO or the destination of any output from TECO (except for type-in and type-out at the user's terminal). When the device is "DSK" (the system disk file) it can be omitted (with the following ":"). When the directory-name is the user's name, it can be omitted (with the enclosing "<" and ">"). Both of these omissions almost always apply. Other abbreviations can be made of a file designator; they are not uniform and are described under the various Input/Output Commands.

The file designator is a construct which is native to the TENEX file system, not TECO; and a full discussion of the file designator can be found in the TENEX documentation.

---

s *** (character string) "Character-String"

where s is a sequence of any number (possibly zero) of characters drawn from the ASCII set of 128 characters.

The character-string is used as an argument of an Ordinary Command, usually to specify a substring of the buffer which is to be inserted or searched for. There are

several rules which apply to all character string arguments, as follows:

-- Whenever one of the Control Command characters (listed at the beginning of the chapter on Control Commands) is used without protection in a character string argument, it does not become a part of that argument. Instead, it is executed for its own effect when TECO first sees it.

-- In addition to the unquoted Control Commands, there is always one other character which cannot be included in the character string. It is the character used to terminate a character string argument. For all commonly used commands, it is the character entered by Control-D, and can be omitted as a terminator when it would be the last character in a command string. In a tag the terminator is "!", and in certain specialized commands it can be any character the user chooses.

Aside from the exclusions just mentioned, any ASCII character can be used in a character string argument. Specifically, all of the control characters which do formatting (End-of-Line, Tab, and so on) can be used freely.

Chapter 8

THE ORDINARY COMMANDS

Most TECO Ordinary Commands have the following form:

-- Prefix Arguments. A command can begin with two arguments separated by a comma, with a single argument, or with no arguments. Each of these prefix arguments must have a coded-integer as its value; and the argument can be any combination of operators, functions, indices, and constants (as described in the preceding chapter) which will produce this value. Usually a change from two arguments to one argument changes the nature of the command; for example, Ø,1ØT types the first ten characters of the buffer, but 1ØT types the ten lines which follow the pointer. On the other hand, when all prefix arguments are omitted from a command which needs these arguments, TECO assumes a specified "default" argument or argument pair for most commands.

-- Command Name. The command name is always required. For most of the important commands, it is a single mnemonic letter. Most of the commands added to TECO since it was imported from DECsystem-1Ø (1971 Version) have names consisting of ";" (semicolon) followed by a mnemonic letter. The remaining commands use various character combinations as more-or-less mnemonic names.

-- Suffix Arguments. A command can end with two character string arguments, a single character string argument, or no arguments. In all cases, these suffix arguments are non-numeric. In the case of the Q-Register Commands, a single letter or digit is used and is the name of the Q-Register involved. In all other cases, these arguments are character strings. The usual technique for delimiting a character string argument is to follow it with the character entered by (↑D); however, other special purpose methods are occasionally used. In contrast to Prefix Arguments, the suffix arguments cannot be omitted; that is, every command

has a specific requirement for Ø, 1, or 2 suffix arguments and this requirement must be filled explicitly.

The prefix argument policy of TECO contains many exceptions and anomalies. As an especially remarkable example, consider the letter "A". When used without any arguments, this is the Append-page input command; but with a preceding integer argument it is the Access-Code function, which has nothing to do with input/output. The interesting point is that the integer argument is not used by the Access-Code function; its only purpose is to distinguish the function from the Append-page Command.

The command form just described applies to all Ordinary Commands except for several in the Flow of Control group. These latter commands require special forms; for example, the Iteration Command contains within it a command string, and it uses the angle brackets, <...>, to delimit this command string.

When a command which would end with the character entered by (↑D) is immediately followed by Escape, then typing of the final (↑D) can be omitted. This abbreviation is especially useful because it frequently applies. In a typical editing session, Insertion, Search, and Input/Output Commands, all of which end with the character entered by (↑D), are often given singly (so they constitute a complete command string) and are terminated by an Escape.

COMMAND STRING

A command string is a sequence of Ordinary Commands. The user responds to TECO's "*" by typing a command string and then typing an Escape. The Escape is not part of the command string; rather, it is the means by which the user causes a command string to be executed. There are other ways in which a command string can be executed. It can be part of an enclosing command, as in the case of an Iteration Command, or it can be loaded in a Q-Register and then executed as a subroutine by the Macro Command.

ERROR HANDLING:

Undetected Errors. An ill-formed command is a character sequence which is not explicitly defined as a legal command. It may be ill-formed because it contains an ill-formed argument; this case was described at the beginning of the preceding chapter. It may be ill-formed because it has an illegal command-name or the wrong number of arguments; these cases are described here. In some cases TECO detects an ill-formed command, types an error message, and asks for new commands. However, in other cases TECO interprets an ill-formed command as if it were a (similar) legal command, as follows:

Too Many Arguments. Unwanted arguments are sometimes ignored by TECO. For example, "5,12C" becomes "12C", and if "HT6" is a complete command string it becomes "HT".

Parenthesized Argument Pair. When a pair of arguments (separated by a comma) is illegally enclosed in parentheses, the parentheses are ignored. For example, "(3,5)T" becomes "3,5T".

Up-Arrow Commands. Several names of commands and functions have the form "↑c"

Illegal No-Ops. There are many characters which are not legal command names.

c	***	(no operation)	"No-Op"

where c is the End-of-Line character, the Space character, or the character entered by (↑D).

The No-Op Command causes no action. It has two purposes, as follows:

-- A No-Op can be used to help format a long command string, especially one which is filed for later use as a stored program. The Space and End-of-Line No-Ops are used for this purpose.

-- A No-Op command can be used to "use up" any integer arguments which precede it (see the description of the % Command in the Q-Register Group). The (↑D) No-Op is used for this purpose.

WARNING:

When a Space character occurs between two integers, it is interpreted as "+", not No-Op. In addition, Space is used as the second character of the Skip-Out Command, ";(SP)".

ENTER/EXIT

These commands are used for the initial entry into TECO and for the final exit back to the TENEX Executive. Other commands designed for a temporary visit to the EXEC are discussed elsewhere, under Control Commands, Interrupt.

TECO *** (call TECO) "Call-Text-Editor-and-COrrector"

where this is a TENEX Executive Command.

This command calls the TECO subsystem of TENEX and establishes TECO in its initial state. In this state, those registers which can accommodate coded integers each contain a Ø (zero) and the remaining registers and buffers are empty. When the initialization process is complete (after a few seconds of real time), TECO types "*" and waits for the first string of TECO Ordinary Commands from the user.

EXAMPLE:

A recommended beginning for the usual TECO editing session is as follows:

```
@TECO(%)
*;Y(ESC)(%)
INPUT FILE:  HENRY(ESC).TEXT;3 [Confirm](%)
389 CHARS(%)
```

Notice that the ;Yank Command accepted just a small part of the total file designator (namely, "HENRY"), then informed the user of the extension and version number ("TEXT;3") which it had assumed, and finally let the user confirm this assumption (with a Carriage Return). Other input commands allow partial specification of a file designator, but none type out the assumed completion and allow confirmation or rejection. The ;Yank Command is described in the Simple Input/Output section of this chapter.

EDIT(SP)f abbrev (call TECO on a file) "Edit-a-File"

where (SP) is a Space character;

where f is a file designator; and

where this is a TENEX Executive Command.

This command is an abbreviation for a particular way of entering TECO, opening an input file, and reading the entire

file into the buffer. Specifically, the dialog

```
@EDIT HENRY(%)
```

is approximately equivalent to the dialog

```
@TECO(%)
*;RHENRY;Y(ESC)(%)
```

As a side effect, the TENEX EXECUTIVE remembers the name and extension of the file being edited. Subsequent "EDIT" commands will use these for defaulting so re-edits are done by typing as little ED% to the EXEC.

ERROR HANDLING:

Not Found. If the designated file does not exist in storage, TENEX types "?".

Edit file deleted. The name being remembered by the TENEX EXEC no longer represents a file.

Bad Designator. If f is not in a legal form, TENEX types "?".

;H *** (simple exit from TECO) "Halt"

The ;Halt Command is intended to be used as a final exit from TECO after the user has completed his editing session and safely filed his results. This role contrasts with the role of the Control-C Control Command, which is intended for a temporary exit from TECO.

In practice, however, the two commands are not so distinct. In accordance with the general error-forgiving nature of TENEX, the user can recover from a ;Halt Command which was given prematurely, provided he acts promptly. If no command which modifies the user-memory has been given since exit from TECO, the user can resume his TECO session with the EXECutive Command CONTINUE.

WARNING:

If the output file is open when this command is given, portions of the output may be lost. However, this danger does not apply to most editing sessions since the output file cannot be left open unless commands described in the section on Complicated Input/Output have been used.

EX paging (close file and exit) "Exit-from-TECO"

equivalent to ;U;H

ED paging (date, close file, exit) "Exit-with-Date"

equivalent to ;D;H

EG special (execute edited program) "Exit-and-Go"

The Exit-and-Go Command exits from TECO and enters the CCL COMPIL Module. Any required output activities must be performed before this command is executed.

INSERT CHARACTERS

This group of commands is used to insert new text data into the buffer. The group begins with the Insert-String Command, which can handle an almost arbitrary character string and proceeds to related but much more specialized commands. Next, the Insert-Code Command is given, which is the only command in all of TECO which can deal with ASCII characters with no exceptions. Then a command is given for inserting the decimal representation of a coded-integer value. Finally, the group includes the ;Get-Commands Command, which is used to rescue the argument string of a previously given illegal Insert-String Command.

Is(↑D) *** (insert character string) "Insert-String"

where s is a character string which does not contain the character entered by (↑D).

The Insert String Command inserts the entire character string, s, immediately to the left of the pointer. The actual insertion only occurs when the command is executed (as a result of a terminating Escape), and until that time the character string (which may be many pages of formatted text) accumulates as part of one long command string.

If an unquoted Control Command occurs in the argument, s, of this command, it is not treated as part of the string to be inserted but instead is executed as a Control Command. The Control Commands are listed at the beginning of the chapter on Control Commands.

WARNING:

Because of the importance of the Insert Command, the mistakes to which it lends itself are also important. The user can (1) leave off the initial "I" or (2) attempt to include the character entered by (↑D) and thus terminate the argument character string prematurely. In either case, TECO will attempt to execute as a command string a character string which the user intended to be text data. TECO will continue execution until it detects an illegal command, and will then print the appropriate diagnostic message and return to the await-commands state. At this point, the user should trace the runaway execution which has occurred. The Command Index Appendix will assist the user in this process. The runaway usually does no damage to the TECO data; but on other occasions the buffer may be chewed up, the pointer may have slipped, a Q-Register may have been clobbered, and so on.

EXAMPLES:

Iknow thyself(↑D) --- inserts "know thyself" in the buffer just before the pointer. The (↑D) can be omitted if it is at the end of a command string.

know thyself(↑D) --- deletes the remainder of the line after the pointer (the "k" command) and searches for "ow thyself" (the "n" command).

(HT)s(↑D) abbrev (insert tabbed string) "Tab-Insert"

where (HT) is the (Horizontal) Tab character; and

where s is a character string which does not contain the character entered by (↑D),

equivalent to I(HT)s(↑D)

This command allows the user to drop the "I" from the beginning of a regular Insert-String Command if the argument string of that command begins with the Tab (Control-I) character.

@Itst programs (special insert string) "@Insert-String"

where t is any character which does not appear in s and is not an unquoted Control Command; and

where s is a character string which can contain the character entered by (↑D) but no other unquoted Control Command.

The @Insert-String Command can be thought of as the result of taking a regular Insert-String Command, changing the way in which its argument string is delimited, and putting an "@" character in front to signal the change. When the argument is specified in this way, the character entered by (↑D) can be included in the argument but the new terminator, t, cannot.

Clearly the value of this command over the regular Insert Command is for the case when s must include the character entered by (↑D). This case occurs when the user is preparing a character string which will later be executed as a TECO command string; specifically, when the user wants to insert Insert, Search, Replace or Input/Output instructions into the buffer. This is why the @Insert-String Command is classified as "programs" (for use in connection with writing TECO programs).

See the "Warning" in the description of Insert-String.

ERROR HANDLING:

No Argument Terminator. If TECO comes to the end of the current command string before finding the second (and closing) instance of t (whatever character that may be), then TECO types an error message and awaits new commands.

EXAMPLES:

@I/Iabc(↑D)/ --- inserts into the buffer a command which (if later executed by a Macro Command) will insert "abc" into the buffer.

@I+Iabc(↑D)+ --- equivalent to the example above.

---

nI *** (insert character by code) "Insert-Code"

where n is an integer expression between Ø and 127

The Insert-Code Command determines which ASCII character has a code whose numeric value is identical to the value of n; then it inserts this character in the buffer just before the pointer. Any one of the 128 ASCII characters can be inserted by using a suitable integer value.

ERROR HANDLING:

Out of Range. If n is not in the range Ø to 127, its value modulo 128 is used (high order bits are dropped).

EXAMPLES:

3I --- inserts a Control-C into the buffer just before the pointer. Control-C is one of the Interrupt Control Commands, and this is the only way it can be inserted into the buffer.

↑TI --- waits for the user to type a character and then inserts it into the buffer.

---

nG *** (Get copy) "Get-copy"

where n is an integer expression

A copy of the next n lines is inserted after the pointer

---

m,nG *** (Get copy) "Get-copy)"

where m and n are integers

A copy of the (n-m) characters beginning at m are inserted after the pointer.

---

n special (convert and insert integer) "Insert-Number"

where n is an integer expression

This command expresses the coded-integer value of n in ASCII characters as the shortest possible decimal representation and inserts that representation into the buffer just before the pointer.

WARNING:

If an integer argument is not supplied to this command TECO does not report an error; instead, TECO interprets this as quite a different command, namely the obsolete version of the ;Number-Pickup function.

;G *** (retrieve a bad type-in) "Get-Commands"

The ;Get-Commands Command copies the contents of the Command Register Backup Register into the buffer just before the pointer. The contents of the Command Register Backup Register are unchanged.

The Command Register Backup Register always contains the most recent previously executed command string which had 16 or more characters; therefore this command can be used to recover from an ill-formed Insert-String Command which had a long (and therefore valuable) argument string.

POSITION POINTER

These commands position the pointer in the buffer as specified by an integer-valued argument. When the argument is omitted, the command is accepted and given a default interpretation.

The argument can designate the number of characters between the beginning of the buffer and the desired position of the pointer (for the Jump Command), the number of characters to move the pointer forward or backward relative to its current position (for the Character-Skip Command), or, in an especially useful way, the number of lines to move the pointer forward or backward relative to the current line (for the Line-Skip Command).

Several TECO functions move the pointer as a side effect. They are

-- The ;Pickup-Code Command, which gets the coded-integer value of the character code for the character just after the pointer and then moves the pointer one character to the right;

-- The ;Number-Pickup Command, which gets the coded-integer value of the unsigned decimal representation (if any) which immediately follows the pointer and then moves the pointer to the right of that integer representation; and

-- The \ Command, which is similar to the ;Number-Pickup Command.

These three functions are discussed under Ordinary Arguments, Character Code and Special Functions.

---

nJ *** (set pointer) "Jump"

where n is an integer expression.

J means ØJ --- Note exceptional default

nJ places the pointer just after the n-th character of the buffer. For the case of n=Ø, the command places the pointer at the beginning of the buffer.

ERROR HANDLING:

Off The Edge. If n specifies a position beyond the beginning or end of the buffer (n<Ø or n>Z), then the

pointer is not changed, and TECO types an error message and awaits new commands.

EXAMPLES:

ØJ --- places the pointer just before the first character of the buffer.

1J --- places the pointer just after the first character of the buffer.

ZJ --- places the pointer just after the last character of the buffer.

QØJ --- If QØ has the numeric value 24, places the pointer just after the 24-th character of the buffer.

---

nC *** (skip characters) "Character-Skip"

where n is an integer expression

C means 1C

-C means -1C

Suppose the pointer is just before the i-th character of the buffer. Then the nC command moves the pointer to just before the (i+n)-th character of the buffer.

ERROR HANDLING:

Off The Edge. If n specifies a position beyond the beginning or end of the buffer (.+n<Ø or .+n>Z), then the pointer is not changed, and TECO types an error message and awaits new commands.

EXAMPLES:

ØC --- does nothing.

3C --- moves the pointer three characters forward.

-C --- moves the pointer backward one character.

---

nL *** (skip lines) "Line-Skip"

where n is an integer expression

L means 1L

-L means -1L

Suppose the line which contains the character just after the pointer is the i-th line of the buffer. Then the nL command moves the pointer to the beginning of the (i+n)-th line of the buffer.

The command always leaves the pointer at the beginning of a line or (as a special case) at the end of the buffer. If this command attempts to move the pointer beyond the boundary of the buffer, the action is not treated as an error. Instead, the pointer is left at the beginning or end (whichever applies) of the buffer.

EXAMPLES:

ØL --- moves the pointer to the beginning of the current line.

L --- moves the pointer to the beginning of the next line (if there is one) or the end of the buffer (if the pointer was in the last line).

-L --- moves the pointer to the beginning of the preceding line.

L-C --- moves the pointer to the end of the current line.

TYPE-OUT

The most important of these commands type portions of the buffer at the user terminal, as specified by one or two integer-valued arguments. When the arguments are omitted, the command is accepted and given a default interpretation.

A pair of integers can be used with the Type-String Command to specify character counts for the start and end of the type-out. A single integer can be used with the Type-Lines Command to specify the number of lines to be typed out just before or after the pointer. Finally, a pair of integers or single integer can be used with the View Command to specify in a simpler way the number of lines to be typed out before and after the pointer.

The group also includes two commands carried over from DDT which "turn the platen" one line forward or one line back.

Finally, this group contains commands which type the value of a coded-integer or type a message or position the paper at the user's terminal for a clean type-out.

---

m,nT *** (type character string) "Type-String"

where m and n are integer expressions

HT means Ø,ZT

m,nT types out at the user terminal the character string which begins just after the m-th character of the buffer and which ends just after the n-th character of the buffer. (It does not change the buffer or the pointer.)

If this command attempts to type out characters beyond the boundaries of the buffer, the action is not treated as an error so long as part of the specified character string lies within the buffer.

ERROR HANDLING:

Backward Type-Out. If m>n, TECO types an error message and awaits new commands.

Entirely Off The Edge. If an attempt is made to type characters none of which are even adjacent to the actual buffer (that is, n<Ø or Z<m), then TECO types an error message and awaits new commands.

EXAMPLES:

.,.T --- types nothing.

Z-2Ø,ZT --- types the last twenty characters of the buffer.

Q1,Q2T --- If (for example) Q1 and Q2 have the integer values 45 and 873, the command types from the 46th character to the 873rd character, inclusive.

HT --- types the whole buffer. If the user has second thoughts about any type out, he can abort the typing out process with a single Delete.

---

nT *** (type adjacent lines) "Type-Lines"

where n is an integer expression

T means 1T

-T means -1T

Suppose the line which contains the character just after the pointer is the i-th line of the buffer. Then the nT command types out the characters between the pointer and the beginning of the (i+n)-th line of the buffer.

If this command specifies lines which are beyond the beginning or end of the buffer, it is not treated as an error. Instead, the command is treated as if it had specified lines up to the beginning or end of the buffer, respectively.

EXAMPLES:

T --- types that portion of the line after the pointer, and is the best way to find out where the pointer is.

ØT --- types that portion of the line before the pointer and is useful for finding the pointer when it happens to be at or near the end of a line.

-3T --- types three lines before the current line and continues typing into the current line up to the pointer.

---

m,nV *** (type adjacent lines) "View"

where m and n are integer expressions

V means 1V

nV means n,nV

m,nV types the (m-1) lines before the current line, types the current line, and types the (n-1) lines after the current line. The command does not change the buffer or the pointer.

If this command attempts to type lines beyond the boundary of the buffer, the action is not treated as an error. Instead, the command types those specified lines which are within the boundary.

ERROR HANDLING:

Negative Arguments. A negative argument is treated as infinity (some number greater than 2↑35). No error message is typed out.

EXAMPLES:

V --- types the current line, and is most useful for finding the entire line in which the pointer appears. Can be followed by

T --- to determine where in the line the pointer is.

2V --- types the current line and gives a context of one line in each direction. It is useful for confirming the pointer position in a long and repetitive listing.

---

n= *** (type coded integer) "Type-Integer"

where n is an integer expression

The command types out the decimal representation of the value of n followed by an End-of-Line character.

ERROR HANDLING:

Illegal Value. If the command is not preceded by an argument, n, which produces a coded-integer value, TECO types an error message and awaits new commands.

EXAMPLES:

.= --- types the number of characters before the pointer.

2*(Z-QA)= --- types the value of the given integer expression.

;Ts(↑D) programs (type message) "Type-Message"

where s is a character string which does not contain the character entered by (↑D).

The command types out the character string, s, which is its argument. No End-of-Line character or any other character is added to the type-out. The command is used in a TECO program to prompt input, identify output, or report on progress.

(LF) abbrev (type next line) "Next-Line"

where (LF) is the Line Feed character.

equivalent to LT(ESC)

The command moves the pointer to the beginning of the next line and then types that line.

The command must be the first character of a command string; when a Line Feed character appears as a command elsewhere in a command string, it is ignored.

(↑H) abbrev (type previous line) "Previous-Line"

equivalent to -LT(ESC)

The command moves the pointer to the beginning of the previous line and then types the line.

The command must be the first character of a command string; when a backspace character (↑H) appears as a command elsewhere in a command string, it is a terminal-dependent character delete command, similar to ↑A, See Chapter 6.

DELETE CHARACTERS

These commands delete characters from the buffer as specified by one or two integer-valued arguments. When the arguments are omitted, the command is accepted and given a default interpretation.

A pair of integers can be used with the Kill-String command to specify the character counts for the start and end of the deletion. A single integer can be used with the Kill-Lines Command to specify the number of lines to be deleted before or after the pointer. Finally, a single integer can be used with the Delete-Characters Command to specify the number of characters to be deleted just before or after the pointer.

---

m,nK *** (delete character string) "Kill-String"

where m and n are integer expressions

HK means Ø,ZK

m,nK deletes the character string which begins just after the m-th character of the buffer and which ends just after the n-th character of the buffer. Then it places the pointer just after the m-th character of the buffer.

If this command attempts to delete characters beyond the boundaries of the buffer, the action is not treated as an error as long as part of the character string lies within the buffer.

WARNING:

In contrast to the other deletion commands, this command may move the pointer relative to the surviving characters. This command does not delete relative to the current pointer position, and the pointer may be anywhere in the buffer before the deletion. After the deletion, it will be at the point where the deleted character string began.

ERROR HANDLING:

Backward Deletion. If a "backward" deletion is requested (that is, m>n), then the buffer and pointer are not changed, and TECO types an error message and awaits new commands.

Entirely Off The Edge. If an attempt is made to delete characters none of which are even adjacent to the

actual buffer (that is, n<Ø or Z<m), then TECO types an error message and awaits new commands.

EXAMPLES:

Ø,2ØK --- deletes the first twenty characters and places the pointer at the beginning of the buffer.

2Ø,ZK --- deletes all but the first twenty characters and places the pointer after the 2Ø-th character (at the end of the buffer).

Z-2Ø,ZK --- deletes the last twenty characters and places the pointer at the end of the buffer.

---

nK *** (delete adjacent lines) "Kill-Lines"

where n is an integer expression

K means 1K

-K means -1K

Suppose the line which contains the character just after the pointer is the i-th line of the buffer. Then the nK command deletes the characters between the pointer and the beginning of the (i+n)-th line of the buffer.

If this command specifies lines which are beyond the beginning or end of the buffer, it is not treated as an error. Instead, the command is treated as if it had specified lines up to the beginning or end of the buffer, respectively.

EXAMPLES:

K --- deletes that portion of a line after the pointer.

ØK --- deletes that portion of the line before the pointer.

-3K --- deletes all characters from the beginning of the third line before the current line to the pointer.

---

nD *** (delete adjacent chars.) "Delete-Characters"

where n is an integer expression

D means 1D

-D means -1D

Suppose the pointer is just before the i-th character of the buffer. Then the n I command deletes the characters between the pointer and the (i+n)-th character of the buffer.

ERROR HANDLING:

Off the edge. If an attempt is made to delete characters beyond the boundaries of the buffer (.+n<Ø or .+n>Z), then the buffer and pointer are not changed and TECO types an error message and awaits new commands.

EXAMPLES:

ØD --- does nothing.

-DD --- deletes one character on each side of the pointer.

D-D --- and so does this.

SEARCH

These commands position the pointer in the buffer by searching for a particular substring of the buffer and placing the pointer after the substring when and if it is found. They are a valuable complement to the Position Pointer Commands, which operate by counting characters.

TECO will also search backward from the current location by supplying a negative repetition count. Thus, -3Sfoo$ will find the third occurrence of "foo" before "." Note that -Sfoo$ is short for -1Sfoo$.

When a backward search finishes, the pointer will be left at the end of the string found. Thus, one cannot tell whether a forward or backward search was used to find the string.

A backward search begun with the pointer at the beginning of the buffer will automatically bring the pointer to the end.

The first of the commands, Search, is adequate for most applications. Two special commands (and their obsolete alternative forms) are given which search through a file which is divided into TECO pages. Two "modifiers" are given which can be applied to the commands in this group (1) to cause a search to affect the flow of control of a TECO program and (2) to allow the inclusion of the character entered by (↑D) in the search argument string. Finally, four "match control" characters are given which have a special, non-literal meaning when they appear in the argument string of a Search Command.

---

nSs(↑D) *** (search for string) "Search"

where n is an integer expression which is negative to specify reverse search direction, and

where s is a character string of less than 72 characters which does not contain the character entered by (↑D)

Ss(↑D) means 1Ss(↑D)

The Search Command seeks to match its argument string, s, against the contents of the buffer. Suppose that s specifies a string of i characters. The first attempt at a match is made with the string of i buffer characters which begins with the first character after the pointer; the

second attempt at a match is made with the string of i characters which begins with the second character after the pointer (or first character before the pointer if reverse search), and so on.

The magnitude of the integer argument, n, of the command specifies how many times the argument string must be successfully matched with the contents of the buffer. The search must conclude in one of the following ways:

-- If an attempted match succeeds on n separate occasions, the pointer is moved to the position immediately after the n-th successfully matched substring of the buffer; and the execution of the command is complete and is said to "succeed". If n is Ø, the search succeeds immediately without moving the pointer. Otherwise,

-- If n separate matches cannot be found before the end of the buffer is reached, the pointer is left in its original position (where it was just before this command); and the execution of the command is complete and is said to "fail".

For the purposes of matching with the buffer, the argument in the Search Command, s, is interpreted character by character as follows:

-- If a character is an unquoted Control Command, it is executed when it is first seen by TECO and is not included in the argument;

-- If the character is the character entered by (↑D), it causes the premature termination of the argument and is not, of course, included in the argument;

-- If the character is one of the four control characters Control-X, Control-S, Control-N, or quoted Control-Q it is handled as indicated under the separate entries in this group, below. The Control-N and quoted Control-Q each "uses up" the character which follows it; finally,

-- If the character is none of the above, it is matched literally against the corresponding character of the buffer.

WARNING:

In entering the Search Command, the user can (1) leave off the initial "S" or (2) attempt to include the character entered by (↑D) and thus terminate the

argument character string prematurely. In either case, TECO will attempt to execute as a command string a character string which the user intended to be a text substring. After such a "runaway" execution has occurred, the user should carefully trace its effect.

WARNING:

When a Search Command is aborted (by the Delete Command) TECO will type out the error message which normally means the search was completed and failed. This indicates that the search had not yet succeeded when the command was aborted (and thus the pointer was not moved). It does not necessarily mean that the search was completed and failed.

ERROR HANDLING:

Not Found. If a Search Command without the ":" modifier fails to find a match, TECO types an error message and awaits new commands.

Too Big. If the string being searched for, s, contains 72 or more characters, TECO types an error message and awaits new commands.

EXAMPLES:

Sis(↑D) --- searches for the next "is" in the buffer. The (↑D) can be omitted if it is at the end of a command string.

S(↑S)is(↑S)(↑D) --- searches for the next "is" which is preceded by and followed by a separator character (and is therefore a complete word).

---

nFs(↑D) paging (search page after page) "Find"

where n is an integer expression which is not negative, and

where s is a character string of less than 72 characters which does not contain the character entered by (↑D)

Fs(↑D) means 1Fs(↑D)

The Find Command is intended for use only when a file is being edited according to its division into TECO pages, that is, it is a member of the large family of TECO "paging" operations. The command is identical to the Search Command except for the extent of its search.

The search begins at the current pointer position, but continues, if necessary, until the end of the input file is reached, not just to the end of the current contents of the buffer. Specifically, if n matches are not found before the end of the buffer, the contents of the buffer are written out on the output file, the buffer is cleared, and the next page of the input file is read in. This page-turning process is continued until one of the following conditions apply:

-- If the nth match is made, the buffer is moved to the position immediately after the successfully matched substring of the buffer; and the execution of the command "succeeds". Otherwise,

-- If the n-th match is not found before the end of the file is reached, the effect is that the entire remainder of the input file has been copied, page by page, through the buffer to the output file and the buffer has been cleared; and the execution of the command "fails".

This command does not close the output file in any case.

The command makes its last attempt at a match for a given page with the last characters in the buffer; and it makes its first attempt for the next page with the first characters of the new buffer contents. Accordingly, it will not match a substring which extends from the end of one buffer load into the beginning of the next buffer load.

---

nNs(↑D) obsolete (search page after page) "Next"

equivalent to nFs(↑D)

---

n;Fs(↑D) paging (search pages without output) "Find"

where n is an integer expression which is not negative, and

where s is a character string of less than 72 characters which does not contain the character entered by (↑D)

;Fs(↑D) means 1;Fs(↑D)

The Find Command is intended for use only when a file is being edited according to its division into TECO pages and the pages searched over can be discarded. The command is identical to the Find Command, except that whenever the buffer would normally be written on the output file (as the

search reaches the end of a page), the contents of the buffer are instead discarded.

The main application of the command is to a situation in which the user is reading through a file but not modifying it.

---

n←s(↑D) obsolete (search pages without output) "Old-Find"

equivalent to n;Fs(↑D)

:c programs (convert search to test) "Conditional-Search"

where c is any Search Command from this group which is not already a conditional search.

Each of the five Search Commands described has a "success" or "fail" termination. In the absence of the ":"-modifier, a "fail" exit produces an error message. The ":"-modifier has the curious effect of combining with a search command to produce a function which has an integer value. This function has the value -1 if the search command "succeeds" and Ø if the search command "fails".

A Conditional Search Command is used in a TECO program when the program flow depends on the success or failure of the search. Specifically, the Conditional Search is used as an argument to the Conditional Commands in the Flow of Control group; and these commands execute one sequence of commands if the value of the Conditional-Search is -1 and another if it is Ø.

@ctst programs (special search) "@Search"

where c is any Search Command from this group which is not already an "@"-search and which has had its search argument, s(↑D), removed;

where t is any character which does not appear in s and is not a Control Command; and,

where s is a character string.

The @Search command can be thought of as the result of taking any other Search Command in this group, changing the way in which its argument string is delimited, and putting an "@" character in front to signal the change. When the search argument is specified in this way, the character entered by (↑D) can be included in the argument but the special terminator, t, cannot.

The "@"-modifier and the ":"-modifier may be used together to begin a Search Command, in either order, "@:" or ":@"; but only one of each may appear in a command.

The Conditional-Search Commands are similar to the @Insert-String Command described earlier in this chapter. The @Insert a Command was included in TECO so the user could conveniently type strings into the buffer which would eventually become TECO programs. Similarly the Conditional-Search Commands exist so that a program-to-be in the buffer can be searched for a substring containing the character entered by (↑D).

ERROR HANDLING:

No Argument Terminator. If TECO comes to the end of the current command string before finding the second (and closing) instance of t (whatever character that may be), then TECO types an error message and awaits new commands.

(↑X) *** (accept any character) "Any-Match"

When used in the argument string of a Search Command, the Control-X character matches any character whatever. It can thus be used as a "don't care" position in a search argument.

(↑S) *** (accept separator character) "Separator-Match"

When used in the argument string of a Search Command, the Control-S character matches any separator character. A separator is any character other than a letter, a digit, a period, a dollar sign, or a percent sign; that is, a separator is any character except those commonly used in names.

(↑N) *** (reject what follows) "Not-Match"

When used in the argument string of a Search Command, the Control-N character works in conjunction with the character which immediately follows it. It specifies that the next character of the buffer will be accepted as a match only if it does not match the character which follows the Control-N. The Control-N can be applied to any literal character. In addition, it can be applied to the Control-S character. The result quite properly matches a buffer

character which is not a separator.

(↑V)(↑Q) special (take match literally) "Quote-Match"

When used in the argument string of a Search Command, the quoted Control-Q character works in conjunction with the character which immediately follows it. If the following character would normally be interpreted literally, then the quoted Control-Q has no effect, and could be omitted; however, if the following character is one of the match control characters, namely Control-X, Control-S, Control-N and the quoted Control-Q itself, then that character is matched literally against the corresponding character of the buffer. Thus the match control characters do not take anything out of the repertoire of possible searches.

Outside of the context just mentioned, this command does not have a special interpretation; that is, it is an ordinary application of the Quote Command, Control-V, which is described under Control Commands.

REPLACE

This group contains a single command, Replace, which searches for a particular string in the buffer and, if the search is successful, replaces the string with another string, also specified.

---

nRs(↑D)t(↑D) *** (replace string) "Replace"

where n is an integer expression which is negative to specify reverse direction of the "SEARCH" implicit in "REPLACE", and

where s and t are character-strings which do not contain the character entered by (↑D).

Rs(↑D)t(↑D) means 1Rs(↑D)t(↑D)

The command performs n replacements. Each replacement searches for s, deletes s, and inserts t. The command is

equivalent to n<Ss(↑D)-cDIt(↑D)>

where c is the number of characters matched by s.

EXAMPLES:

Rnot(↑D)now(↑D) --- replaces the next "not" by "now" and leaves the pointer after "now". The (↑D) can be omitted if it is at the end of a command string.

Rnot(↑D)(↑D) --- deletes the next "not" and leaves the pointer at the position of the deletion. The two (↑D)'s can be omitted if they are at the end of a command string.

-3Rthis(↑D)these(↑D) --- proceeding towards the beginning of the buffer from the correct location, three instances of "this" are replaced by "these".

FLOW OF CONTROL

The commands in this group together with the Macro command of the Q-Register Group provide many of the features of a full-scale high-level programming language. They provide for nested loops with conditional exits, for conditional exits, for conditional execution based on a general integer expression, and for unconditional transfer of control. But because the normal use of TECO is not as a programming language system but as an interactive text editor, the most valuable control mechanism is the most specialized: the single Iterate Command.

Two commands in this group, the Iterate and Conditional Commands, have an unusual form (for TECO); they each contain an arbitrary string of commands as one of their arguments. These commands must be properly "nested"; that is, if a command contains part of another command, it must contain all of that command.

---

n<c> *** (repeat a command string) "Iterate"

where n is an integer expression which is not negative, and

where c is a command string.

<c> means 34359738368<c>

where 34359738368 is 2↑35, which, for most purposes, is equivalent to infinity.

The Iterate Command causes the command string, c, which is delimited by the angle brackets to be executed n times.

The O Transfer Command (the transfer of control, described below) can be used to transfer within an Iterate Command but not into or out of an Iterate Command. In the latter cases, the effect of the transfer of control is not reliable (that is, it is undefined).

ERROR HANDLING:

Missing Left Bracket. If TECO finds a right angle bracket without a preceding left angle bracket, TECO types an error message and awaits new commands.

Bad Count. If n is negative TECO types an error message and awaits new commands.

---

n;(SP) programs (test in iteration) "Skip-Out"

where n is an integer expression; and

where (SP) is a Space character.

The Skip-Out Command takes one of two actions, according to the value of n:

-- If n is negative, TECO proceeds to the immediately following command; but

-- If n is positive or zero, TECO skips to the next unmatched right angle bracket, >, and resumes execution immediately after it.

The user can think of this command as being a lazy command which asks the question, "can I exit the loop now?" If the argument is negative, the answer is "No".

The most important use of this command is its combination with a Search Command. Any Search or Replace Command which is contained in an Iterate Command is treated as if it were a Conditional-Search, whether the ":"-modifier is actually present or not. Thus each Search Command has a value -1 or Ø (according as the search succeeds or fails), and can be used as the argument n to the Skip-Out Command. Specifically, any Search Command immediately followed by a Skip-Out Command takes one of two actions, as follows:

-- If the search succeeds, TECO proceeds to the command which immediately follows the Skip-Out command; but

-- If the search fails, TECO skips to the next unmatched right angle bracket, >, and resumes execution immediately after it.

The user can think of this sequence of commands as asking the question, "Shall I exit repeat loop?"; and, if the search succeeded (and found something to work with), the answer is "No".

Any number of Skip-Out Commands can be used in a given Iterate Command; each one will skip to the end of the Iterate Command if its argument is non-negative.

WARNING:

The space after the ";" is part of the command name, just as, for example, the "H" is part of the command name in the ;Halt Command.

WARNING:

Do not use an angle bracket, "<" or ">", in a character string argument between a Skip-Out Command and the end of the smallest Iterate Command which contains the Skip-Out Command.

This skip to the "unmatched" right bracket will usually get to the "right" one; that is, if TECO encounters embedded Iterate Commands it will skip them properly. However, if TECO encounters an angle bracket in a character string argument (of an Insert or Search Command, for example), it will mistakenly treat this angle bracket as the beginning or end of a Iterate Command and will be misled in its search for the correct right angle bracket.

EXAMPLE:

J<Sword(↑D); Is(↑D)>HT --- changes all occurrences of "word" to "words" and then types out the entire buffer.

ZJ<-R/PUSHJ P,/CALL/; .=V> Goes through the buffer from end to beginning replacing "PUSHJ P," by "CALL", typing the pointer after each replacement and Viewing the entire line.

---

n"xc' programs (conditional execution) "Conditional"

where n is an integer expression;

where c is a command string; and

where x is one of the letters: "E", "N", "L", "G", or "C".

The Conditional Command is actually a set of five different commands, as follows:

-- n"Ec' means "If n Equals Ø, then execute c";

-- n"Nc' means "If n Not-equals Ø, then execute c";

-- n"Lc' means "If n Less-than Ø, then execute c";

-- n"Gc' means "If n Greater-than Ø, then execute c"; and

-- n"Cc' means "If n is the ASCII code of a character which is a normal Constituent of a name (specifically, a letter, a digit, a period, a dollar sign, or a percent sign), then execute c".

In every case a test of n is made and then the command string c is executed or skipped according as the test succeeds or fails.

The O-Transfer Command (the transfer of control, described below) can be used in any way in relation to a Conditional Command; that is, to jump into, within, or out of the command string c.

WARNING:

Do not use a quote (") or an apostrophe (') in the command string argument of a Conditional Command. When the test for one of these commands fails, TECO will usually skip to the "right" apostrophe, that is, if TECO encounters embedded Conditional Commands, it will skip them properly. However, if TECO encounters a quote (") or an apostrophe (') in a character string argument (of an Insert or Search Command, for example), it will mistakenly treat this character as the beginning or end of conditional and will be misled in its search for the correct apostrophe.

ERROR HANDLING:

Missing Apostrophe. If TECO fails to find the apostrophe which closes this command, TECO types an error message and awaits new commands.

EXAMPLE:

↑T-↑↑Y"Exxx' --- does xxx if the user types "Y" when the ↑T Command asks for a character.

!s! programs (program label) "Tag"

where s is a character string which does not contain the "!" character.

This "command" does nothing; its purpose is to label a point in a program to which an O Transfer command will transfer.

Because the tag does nothing, it can be used as a comment as well as a label; and this is the conventional way of annotating a complicated TECO program.

ERROR HANDLING:

Missing "!". If TECO fails to find the exclamation point which closes a tag, TECO types an error message and awaits new commands.

Os(↑D) programs (transfer of control) "O-Transfer"

where s is a character string which does not contain the character entered by (↑D).

The O-Transfer command is the unconditional transfer of control. It searches the complete command string for a tag which has the same character string, that is !s!, and continues execution with the command which follows that tag.

WARNING:

Do not use an O-Transfer Command to transfer into or out of the command string contained in an iterate Command.

ERROR HANDLING:

Undefined Label. If there is no tag with the same s which appears as the argument of an O-Transfer Command, TECO types an error message and awaits new commands.

Multiple Labels. If there is more than one tag with the same s which appears as the argument of an O-Transfer Command, TECO chooses one of them in a way not defined here and continues execution after that tag. No error message is typed out.

EXAMPLE:

↑T-↑↑Y"ExxxOT(↑D)'yyy!T! --- does xxx if the user types "Y" when the ↑T Command asks for a character; otherwise (when the user types something else), does yyy.

Q-REGISTERS

The commands in this group do not have a particular action in common as do the other command groups. Instead, they are the commands which supply the full range of actions for a particular set of data registers, the Q-Registers. Included are commands to set, use, save, and restore both types of values (character-string and coded integer) contained in these registers. There is a special command to increment a coded-integer, and commands to type-out or even execute a character-string value in a Q-register.

For each of the Q-Register Commands the following applies:

ERROR HANDLING:

Illegal Q-Register Name. If the command has a character other than a letter, digit, or @ as the name of the Q-Register, TECO types an error message and awaits new commands.

---

nUq *** (set Q-Register to integer) "Use-QR"

where n is an integer expression, and

where q is a Q-Register name.

nUq places the coded-integer value of n in the Q-Register whose name is q. The previous contents of the Q-Register are lost.

ERROR HANDLING:

Missing Argument. If the command is not supplied with a coded-integer prefix argument, TECO types an error message and awaits new commands.

Qq *** (evaluate Q-Register) "Q-Value"

where q is a Q-Register name.

This is a function, not a free-standing command. Its value is the coded-integer value contained in the Q-Register whose name is q.

The ;Type-Q-Register Command (described below), which has the form Qq;T, does not constitute a use of the Q-Value command, even though it begins with "Qq". Instead it is a

separate command, complete in itself.

ERROR HANDLING:

Wrong Type. If the Q-Register contains a character-string value, TECO types an error message and awaits new commands.

%q *** (increment and evaluate QR) "Step-QR"

where q is a Q-Register name.

The command adds one to the Q-Register named q and then assumes the coded-integer value contained in that Q-Register.

If the function is followed by a (↑D) (which will "absorb" the numeric value), it can be used as an Ordinary Command.

ERROR HANDLING:

Wrong Type. If the Q-Register contains a character-string value, TECO types an error message and awaits new commands.

---

m,nXq *** (put string in Q-Register) "eXtract-String"

where m and n are integer expressions, and

where q is a Q-Register name.

HXq means Ø,ZXq.

m,nXq removes from the buffer the character string which extends from just after the m-th character to just after the n-th character, and places this character string in the Q-Register named q. Then it places the pointer just after the m-th character of the buffer. The previous contents of the Q-Register are lost.

If this command specifies lines which are beyond the beginning or end of the buffer, it is not treated as an error. Instead, the command is treated as if it had specified lines up to the beginning or end of the buffer respectively.

ERROR HANDLING:

Backward Substring. If m>n, then the buffer, pointer, and Q-Register are not changed, and TECO types an error message and awaits new commands.

Entirely Off the Edge. If an attempt is made to pick up characters none of which are even adjacent to the actual buffer (that is, n<Ø or Z<m), then TECO types an error message and awaits new commands.

---

nXq *** (put lines in Q-Register) "eXtract-Lines"

where n is an integer expression, and

where q is a Q-Register name.

Xq means 1Xq

-Xq means -1Xq

The command removes a substring from the buffer and places it in the Q-Register named q. The previous contents of the Q-Register are lost. Suppose the line which contains the character just after the pointer is the i-th line of the buffer. Then the nXq command extracts the characters between the pointer and the beginning of the (i+n)-th line of the buffer.

Gq *** (copy Q-Register into buffer) "Get-Q-Register"

where q is a Q-Register name.

The Get-Q-Register Command makes a copy of the entire character string in the Q-Register named q and inserts the copy in the buffer immediately before the pointer. The contents of the Q-Register is unchanged.

ERROR HANDLING:

Wrong Type. If the Q-Register contains a coded-integer value, then TECO types an error message and awaits new commands.

Qq;T *** (type-out a Q-Register) "Type-Q-Register"

where q is a Q-Register name.

This Command types out the entire character string which is the contents of the Q-Register named q.

ERROR HANDLING:

Wrong Type. If the Q-Register contains a coded-integer value, then TECO types an error message and awaits new commands.

Mq programs (execute commands in Q-Register) "Macro"

where q is a Q-Register name.

The Macro command is executed as follows:

-- TECO marks its execution position in the command string currently being executed as just after this Command;

-- TECO pushes the current command string down into its private pushdown list (separate from the one mentioned under the two Q-Register pushdown commands);

-- TECO takes the contents of the Q-Register named q as the current command string and executes it;

-- TECO pops up the command pushdown stack (to get the command string in which the Macro Command appears);

-- TECO continues executing the restored command string where it left off, just after the Macro Command in question.

The commands in this group require the following definition:

-- A complete command string is (1) a command string which was accumulated in the Command Register and placed in execution by an Escape or (2) a command string which was loaded into a Q-Register and placed in execution by a Macro Command (see Q-Register Group).

No flow of control structure ever reaches across the boundary of a complete command string.

WARNING:

It has been stated earlier that when a command string is typed in, a (↑D) at the end of the string can be omitted. However, this convention cannot be applied to a command string which is stored in a Q-Register and executed by a Macro Command.

ERROR HANDLING:

Wrong Type. If the Q-Register contains a coded-integer value, TECO types an error message and awaits new commands.

EXAMPLES:

MA --- if Q-Register A contains "ØT;T↑(↑D)T", then this MA command types out the current line with a "↑" where the pointer is.

JIS(↑D)3I@I/(↑D)/ØXBMB --- inserts the command "S(↑C)(↑D)", puts it in Q-Register B, and executes it. This is the only way to search the buffer for an occurrence of Control-C since Control-C is an Interrupt and cannot be typed into a command string directly.

[q programs (push Q-Register down) "Push-Stack"

where q is a Q-Register name.

where the mnemonic for this command is: just as left bracket must precede a right bracket (in ordinary usage), so a push must precede a pop (in stack usage).

This command makes a copy of the contents of the Q-Register named q (whether character string of coded-integer) and pushes it into the Q-Register Stack. The contents of the Q-Register named q are unchanged.

ERROR HANDLING:

Pushdown Overflow. The pushdown stack is 512 deep. When it is already full, this command causes a machine language error and interrupt to the Executive Level. An attempt to return to TECO by the CONTINUE Command will just cause the error interrupt to occur again. A return to TECO by the REENTER Command will loose the current command string, but will otherwise recover successfully.

WARNING:

The stack is cleared each time TECO returns to the await-commands state.

]q programs (pop Q-Register up) "Pop-Stack"

where q is a Q-Register name.

where the mnemonic for this command is: just as a right bracket must follow the left bracket (in ordinary usage), so must a pop follow a push (in stack usage).

This command pops the Q-Register stack and places the value produced (whether character-string or coded-integer) in the Q-Register named q. The previous contents of the Q-Register named q are lost. If the stack is empty, it delivers the value Ø.

BOOTSTRAP FEATURE

Frequently sophisticated programs are written in TECO macros and the author wishes to save the program in a way that it automatically starts itself when run, without having to type TECO commands at all. This can be accomplished using the Bootstrap Feature described below.

If a TECO command string is put into Q-Register @, TECO halted with ;H, and all of core saved using the EXEC SSAVE command, the result will be a RUNable .SAV file. When this file is RUN, the TECO it contains automatically executes the command left behind in @ (that is, it does an M@ command). The canned command string may contain any legal TECO commands, including those which read files, load Q-Registers, and execute Q-Registers.

SIMPLE INPUT/OUTPUT

The commands in this group are used to move information between the TECO editing buffer and any TENEX file. They are recommended for all applications except those rare processes which operate on extremely large files or perform input/output without interaction as part of a stored program of TECO commands.

A single execution of one of these commands performs a complete input or output operation, including the opening and closing of the file. These commands were introduced especially for use in TENEX TECO because the TENEX system, with its large virtual memory, can accommodate the entirety of almost any file the user wants to edit. In contrast, the earlier systems for which TECO was originally designed had smaller memories. The user usually had to divide his file into TECO pages and then process it page by page, using a sequence of more complicated Input/Output Commands.

Although TENEX has a very large memory, there are two practical factors to be mentioned. First, virtual memory has 256K words and so a file of somewhat more than a million characters cannot be loaded into the TECO buffer. This is not a limitation, because very large files can be editted using "paged" commands.

Second, the cost of editing a file increases as the file becomes larger. Each command which inserts or deletes characters from the buffer causes the shifting of all the characters which follow the position of the change. When the size of a file exceeds about 60,000 characters, it becomes appropriate to consider dividing it into TECO pages, doing the extra work of processing it page-by-page, and thus saving a considerable fraction of CPU time. However, the figure given (60,000 characters) is not a sharp cutoff, and is given for "typical" work. If the user is performing an exceptional number of modifications to a file of this size, he should consider using paging; but the expected saving in cost must be balanced against the considerable increase in the complexity of the editing process.

Large files arise in the use of TENEX to prepare documentation; for example, a page of a single-spaced document prepared on TENEX is about 2000 characters long, so a few dozen pages are enough to exceed the efficiency limit (60,000 characters) just mentioned. However, considerations of convenience and prudence usually lead to the dispersal of a large document into several files (along the lines of chapters, for example), and the problem is solved in a natural way.

;Y *** (append full file) "Yank-File"

This command reads into the buffer from a file. There may or may not be a file open for input at the time this command is executed. The usual case is no file open for input. In this case, the command conducts a dialog with the user to obtain a file-designator, opens the designated file for input, copies the entire file contents into the buffer after any characters already in the buffer, and closes the input file. In short, it appends the designated file to the buffer.

The rare case is file already open for input. This can occur only when commands from the Complicated I/O Group have been used or when an input command has been aborted (due to a Delete or I/O error interrupt) during the input process. In this case, the command proceeds immediately to the file which is open for input, begins reading at the point at which any previous input process left off, copies the entire remainder of the file into the buffer after any characters already in the buffer, and closes the input file. In short, it appends the remainder of the designated file to the buffer.

In both cases, the pointer is not moved. If the buffer was empty, the pointer stays at the beginning of the buffer.

In the "usual" case, TECO types "INPUT FILE: " to request a file designator from the user. The dialog is patterned after the similar dialog at the TENEX Executive level. Principal features are

-- Explicit Designation. The user can type the full designator of the required file and terminate it with an Escape.

-- Recognition Designation. The user can start to type the file designator and then, when he thinks he has specified it uniquely, he can type Escape and let TECO try to finish the whole designator. Alternatively, the user may type (↑F) and let TECO try to finish just that descriptor which is currently incomplete; that is, a directory name, a file name, an extension or an (empty) version number, each with its appropriate terminator. Should TECO find the input wanting, it will ring the bell to invite the user to type several more characters followed by Escape or (↑F). The loop continues until TECO can decide what is wanted.

-- Same Old Input. If the last file name used for input in the current TECO session happens to be the one now required, the user can just type Escape right away, and TECO will type the entire file designator.

-- Version Number. If the user lets TECO type the version number, then the version number supplied will be that of the highest-numbered version of the designated file which exists in storage.

-- Confirmation. When the correct file designator has been typed by any of the methods just described, the system types a request for confirmation; the user types a Carriage Return or another Escape; and finally, the system performs the input and then types the number of characters in the whole buffer.

-- Abort. The user may decide to abort. This can occur because he has made a typing error, or, more frequently, because TECO made an unexpected recognition and filled in the wrong file designator. The user should then type Delete twice to abort both the dialog and the Yank Command which started it. The result of a single Delete followed by, say, Escape is anomalous: if TECO has decided on the file requested, it will go ahead (in spite of the single Delete, and open the file, and read one(!) character into the buffer.

-- Other Devices. Ordinarily, the user will be doing his input/output with the TENEX disk files; however, this command can be used with any TENEX file device which is available to the user for input.

WARNING:

If this command is aborted after the user has confirmed the file designator, the input file may be partially read into the buffer and may be left open. A subsequent input command will continue reading the file.

ERROR HANDLING:

Not Found. If the designated file does not exist in storage, TECO does not abort; instead, it repeats its "INPUT FILE " request and the dialog begins again. The user can escape giving a legitimate reply only by aborting the command.

Bad Read. If the file cannot be read because of various equipment failure, TECO types an error message and

awaits new commands. Usually one will type Z=;Y (↑D) to remember where the bad character is and attempt to read the rest of the file.

EXAMPLE:

The following is a typical Yank dialog:

```
*;Y(ESC)
INPUT FILE:  HENRI(ESC)  ?
INPUT FILE:  HENRY(ESC).TEXT;3 [Confirm](%)
389 CHARS
```

The user first supplied the name of a non-existent file, then corrected himself and had the designator recognized and completed.

;U *** (output full buffer) "Unget-File"

This command writes from the buffer onto a file. There may or may not be a file open for output at the time this command is executed. The usual case is no file open for output. In this case, the command conducts a dialog with the user to obtain a file designator, opens the designated file for output (clearing it of any previous contents), copies the entire contents of the buffer into the file, closes the output file, and clears the buffer. In short, it outputs the buffer to the designated file and then clears the buffer.

The rare case is file already open for output. This can occur only when commands from the Complicated I/O Group have been used or when an output command has been aborted during the output process. In this case, the command appends characters to the file which is already open for output as follows: first, the contents of the buffer are written out and the buffer is cleared; then, if there is a file open for input, the portion of that file which has not yet been read is copied directly over and appended to the output file. When this output is complete, the output file is closed, assuring its safe storage. In short, the command completes the process of copying the pages of an input file onto a previously designated output file.

In the "usual" case, TECO types "OUTPUT FILE: " to solicit a file designator from the user. The dialog is parallel to that for Yank, but there are quite a few points of difference. Principle features are

-- Explicit Designation. The user can type the full designator of the required file and terminate it

with Escape. The recommended version number for a new file is 1 (one), of course. If Ø (zero) is typed by the user, however, then TECO will use the version number of the highest-numbered version of the designated file which exists. This, of course, will destroy the previous contents of that file. It avoids proliferations of versions, but it can lead to unexpected and irreversible losses. As a third possibility, the user can type everything (including a final semicolon) up to the version number, and let TECO supply a safe version number, as described below.

-- Version Number. If the user permits TECO to type the version number, then the version number supplied will be one greater than that of the highest-numbered version of the designated file which exists in storage.

-- Recognition Designation. When a user wishes to write a new version of an old file, he uses Escape or (↑F) as described under "recognition" for Yank.

-- Same Old Input. If the last file name used for input in the current TECO session happens to be the one now required for output, the user can just type Escape right away, and TECO will type the entire file designator.

-- Confirmation. When the correct file designator has been typed and terminated, the system types "[New file]" or "[New version]" or "[Old version]". The user types Carriage Return or another Escape; and finally, the system performs the output. In the case of an "[Old version]", the designated file is overwritten and, unlike the effect of the executive command DELETE, this loss is not reversible.

-- Abort. The user may decide to abort an Unget Command. The procedure is the same as for Yank; two Delete's should be used. A single Delete followed by, say, Escape is anomalous: if TECO has decided on the file requested, it will go ahead and append the first character of the buffer to the current contents of the output file (but will not close the output file or change the contents of the buffer).

-- Other Devices. This command can be used with any TENEX file device which is available to the user for output.

WARNING:

If this command is aborted after output has begun, the buffer may be partially written out and the output file may be left open. A subsequent input command can produce unexpected results.

ERROR HANDLING:

Incomplete Designation. If the user tries to create a new file designator (not just a new version) and fails to type the name, a period, the extension, and a semicolon, TECO does not abort; instead, it repeats its "OUTPUT FILE: " request and the dialog begins again. The user can answer or abort the command.

Not Found. If the user tries to write a version of an existing file but the file does not exist, TECO responds as for an incomplete designator.

Bad Write. If the file cannot be written out because of various equipment failures, TECO types an error message and awaits new commands.

No Storage. When there is no disk space available, it will be impossible for TENEX to accept the output request. This may be reported as "ERROR IN OUTPUT".

EXAMPLE:

The following is a typical Unget dialog:

```
*;U(ESC)
OUTPUT FILE:  HENRY.(ESC)  ?
OUTPUT FILE:  HENRY.;(ESC)1 [New file](%)
```

On his first try, the user tried to use a new designator but forgot the extension terminator, semicolon; when this was corrected the output was performed.

;S *** (save copy of buffer) "save"

This command is identical to ;U except that the text buffer is not cleared after the output. Frequently one may name LPT: as the output file.

;D *** (date & output full buffer) "Date-and-Unget-File"

This command is a variant of Unget. If a file is not open for output when this command is initiated, the command generates a date line and places it at the beginning of the output. Its action is otherwise identical to ;U. A date line is a

-- comment string (set ↑D), followed by

-- the full designator of the file , followed by

-- the date and time of the output operation, followed by

-- the name of the user who commissioned the output.

Since the date line begins with a comment string (usually ;) it will be ignored when the file is used as a source program. If the Date-and-unget Command is used for successive versions of the same file, the date lines will accumulate at the beginning of the file and provide a history of its development.

↑D *** (declare comment string) "↑D command"

;D unloads the text buffer to a file, prefixing it with date and time informations and the editor's name. ;D looks at the extension of the file being written and picks an appropriate comment character (or string) to shield this information with. The current table includes the following:

EXT.	COM. CHR.	
.MAC	;	
.FAI	;	
.P11	;	
.PAL	/	
.BCP	//	
.BLI	!	
.PPL	...	(3 .'S and a space)
.F4	C	(C and a space)
.F4Ø	C	(C and a space)
.FOR	C	(C and a space)
.F1Ø	C	(C and a space)

If a ;D is done to a file which has an unknown extension, a semicolon will be used.

The default ;D comment character (string) may be overridden by the ;↑D command. Examples: ↑D%$ or ↑D.COMMENT$ or ↑D$. The string following the two characters UP-ARROW and D are stored as the default heading comment string. ↑D .... $ is terminated by an altmode or control-D. A comment string set by ↑D will be used without regard to the file extension. Normal defaulting may be restored by setting the default string to be null (↑D$).

COMPLICATED INPUT/OUTPUT

The commands in this group are required (1) to input, edit, and output a file in parts rather than as a whole or (2) to include non-interactive input-output in a stored TECO program.

The commands in this group were originally designed to contend with the limitations of a user memory of 32K words or less. Although the main justification for the commands is removed by the large capacity of the Tenex TECO buffer (more than a million characters), the following marginal applications remain:

-- Very Large Files. If the user has more than about 60,000 characters to edit, he may want to partition the file in some way. The longer the character string after the site of any deletion or insertion operation in the buffer, the more the operation costs. The user can either distribute his text over several files or divide the single file into TECO pages by inserting (↑L's) (form feeds) into the file. If the user chooses the latter course, he needs the commands in this group.

-- File Manipulation. If the user wishes to merge, split, or rearrange large text files, he may find it convenient to use the page-turning commands in this group to shuffle parts in from several files and out to several other files. But the user should consider the alternative of reading all the files involved into the buffer, one after another, and using a Q-register repeatedly to pick up a piece of one file and put it down into another file.

-- Old Habits. If the user has developed methods of editing, or even actual TECO programs, which are based on the turning of TECO pages, then these commands are available.

Those commands of this group which open files have a second special use. Because they accept a file designator as an ordinary argument and do not require dialog with the user, they can be used in a stored TECO program which will do editing without interaction.

The commands given here fall into three subgroups, as follows:

-- Open. The first step in an input or output process is the opening of the file. The basic commands for this purpose are ;R and ;W for reading and writing, respectively, and their (older) alternative spellings, ER and EW.

-- Transmit. The second step of input or output is the transmission of one or more TECO pages between the buffer and the input-output device. The input Commands are A and Y. The output Command is W, and (when used with two arguments) P. When used with less than two arguments, P is a "leafing" command; that is it writes out a page and then reads the next one in.

-- Close. The third step of input-output is the closing of the file. This step is critical for output, since the new file is not sent to storage unless it is closed. The EF command explicitly closes the output file. No special command is required to close an input file; this is adequately handled by TECO.

With the exception of magnetic tapes, a file must be read from the beginning toward the end. The only way to back up is to re-open the input file and start again from the beginning.

;Rf(↑D) paging (open input file) "Read-Open"

where f is a file designator

This command closes any file which is currently open for input; opens the file f for input; and prepares to begin input at the first character of the file. (The command does not do any actual input, and it leaves the buffer unaffected.)

This command always positions the file specified by f so that input will begin with the first character of the file, even if that file was opened and partially read earlier in the same TECO session. The file opened by this command remains open for input until the user exits permanently from TECO, or until execution of another command which opens an input file.

The user may leave the file designator incomplete in certain ways, as follows:

DESCRIPTOR	DEFAULT POLICY
device name:	May be omitted. DSK: will be assumed
<directory name>	May be omitted. The name of the currently connected directory will be assumed.
file name.	Must be typed in full, although the . (period) may be dropped if extension and version number are omitted.
extension;	May be omitted if the version number is also omitted. A null extension will be assumed.
version number	May be omitted. The version number of the highest-numbered version of the designated file will be assumed.

Note that this policy differs at several points from that used by ;Y and in different ways from that used by the TENEX Executive command EDIT.

ERROR HANDLING:

Not Found. If the designated file does not exist in storage, TECO types an error message and awaits new commands.

EXAMPLE:

Consider the dialog which follows:

```
*;R ALPHA(ESC)
?NOT FOUND
?12
;R ALPHA
*;RALPHA(ESC)
```

This example emphasizes the fact that a blank cannot be used before or after a file name. In his first try, the user preceded the file designator with a blank, so TECO took " ALPHA" as the file designator. In his second try he leaves the blank out.

;Wf(↑D) paging (open output file) "Write-Open"

where f is a file designator

This command closes any file which is currently open; opens the file f for output; and deletes the contents of the file. (The command does not do any actual output, and it leaves the buffer unaffected).

This command always deletes the contents of the file specified by f, even if that file was open earlier in the same TECO session. The file opened by this command remains open until the execution of a command which closes the file because (1) it is opening another file for output or (2) it is intended to close the output file.

The user may leave the file designator incomplete in certain ways. The default policy for open-Write differs at only one point from open-Read, as follows:

DESCRIPTOR	DEFAULT POLICY
version number	May be omitted. the version number assumed will be greater by one than the version number of the highest-numbered version of the designated file.

ERROR HANDLING:

Bad Designator. If f is not in a legal form TECO types an error message and awaits new commands.

EXAMPLE:

Consider the command

*;WBETA.TXT;3(↑D)PW(ESC)

Here the user had to include the the character entered by (↑D) after the file designator because there was another command in the string. Since the user included a version number, that version of the file is opened and cleared, whether it previously existed or not.

ERf(↑D) obsolete (open input file) "Edit-Read"

equivalent to ;Rf(↑D)

EWf(↑D) obsolete (open output file) "Edit-Write"

equivalent to ;Wf(↑D)

EAf (↑D) *** (open output file for appending) "End-Append"

Causes current file output operations to append to the end of the named file rather than writing a new version.

A paging (append input page) "Append-Page"

This command goes to the file which is currently open for input, begins input where any previous input command left off (or at the beginning, if this is the first input); and reads characters in at the end of the buffer until a termination condition is satisfied. (The previous contents of the buffer remain, and the position of the pointer is unchanged. If the buffer was initially empty, the pointer remains at the beginning of the buffer.)

The termination condition for page input is any of the following:

-- The end of the input file is reached;

-- A form feed is read;

-- The buffer is two-thirds full (or is filled within 128 characters of capacity) and a (LF) line feed is read;

-- The buffer is completely filled.

Obviously the last two conditions will rarely apply for TENEX TECO.

ERROR HANDLING:

Bad Read. If the page of the file cannot be read because of equipment failure, TECO types an error message and awaits new commands. See error index for details.

Y paging (clear buffer & input page) "Yank-Page"

As its first action, this command clears the buffer. After this, the action of the command is

equivalent to A

---

m,nW paging (move string to file) "Write-string"

where m and n are integer expressions

HW means Ø,ZW

equivalent to m,nPWm,nK

(This command is analogous to the Extract-String Command; it is to the output file as the latter is to a Q-Register.)

---

nW paging (move lines to file) "Write-lines"

where n is an integer expression

W means 1W

-W means -1W

The command removes a substring from the buffer and places it at the end of the output file. Suppose the line which contains the character just after the pointer is the i-th line of the buffer. Then the nW command outputs the characters between the pointer and the beginning of the (i+n)-th line of the buffer.

If this command specifies lines which are beyond the beginning or end of the buffer, it is not treated as an error. Instead, the command is treated as if it had specified lines up to the beginning or end of the buffer, respectively.

(This command is analogous to the Extract-Lines Command; it is to the output file as the latter is to a Q-Register.)

ERROR HANDLING:

No Output File. If there is no file open for output, TECO types an error message and awaits new commands.

---

m,nP paging (buffer output) "Page-Output"

where m and n are integer expressions.

HP means Ø,ZP

equivalent to m,nPW

---

nP paging (turn page) "Page-Turn"

where n is an integer expression.

P means 1P

equivalent to n<Ø,ZPY>

(The Page-Turn command advances from the page which is currently in the buffer to the n-th page which follows.)

;C paging (close output file) "Close"

If there is a file open for output, this command closes that file. Otherwise, the command does nothing.

This command is necessary because an output file is not guaranteed to be in storage until the file has been closed. Exit from TECO without closing the output file can result in the loss of all or part of the data which was output to that file since it was opened. Such an exit can occur only through a (↑C) or through system failure. All other exits perform a ;Close as part of their action. The command is not required for the commands in the Simple Input/Output Group; those commands close the files they use.

EF obsolete (close output file) "End-File"

equivalent to ;C

DEBUGGING AIDS

TECO can be used effectively for the development of small programs for text editing and string manipulation. TECO has several simple aids for the debugging of such programs. One of these is the Trace Command, "?", which can cause the commands of a program to be typed as they are executed: this command is described here. A second debugging aid is the ;Type-Message Command, which can be used to report as the program passes various points of its execution, this command is described in the Type-Out Group.

?	programs	(trace or untrace execution)	"Trace"

The first time "?" is encountered in the command string, TECO enters trace mode; the next time, TECO leaves trace mode; the next time, TECO enters trace mode; and so on. When TECO is in trace mode, each command is typed out as it is executed.

Appendix A

ERROR MESSAGES

Error numbers range from 1 to 49. Those not used are not shown here.

?n	Meaning
1	TECO attempted to read commands beyond the end of the command string. This diagnostic is caused by (1) an unterminated @I or @S, (2) an unsatisfied O command, (3) a " Command not matched by a closing ', or (4) an ! not matched by a closing !.
2	?ERROR ON OUTPUT DEVICE; FILE CLOSED. An output error has occurred. The output file was closed at the end of the last good block of data.
3	An attempt was made to supply more than two arguments to a command, either by the use of two commas or by an H followed by a comma.
4	There is a right parenthesis not matched by a left parenthesis.
5	No argument given for an = Command.
6	No argument given for a U Command.
7	An illegal Q-register name (i.e., other than A through Z or Ø through 9) is specified for a G, Q, U, X, or % Command.
9	Q-Register contains a coded-integer where a character string is required.
11	In an Ec Command (e.g., ER, EW, EF), c is illegal.
12	?INPUT ERROR...filnam.ext FILE NOT FOUND. Specified input file not found on LOOKUP.
13	?OUTPUT ERROR...ILLEGAL NAME FORMAT. Blank filename specified for output to a directory device.

14 ?INPUT/OUTPUT ERROR...filnam.ext WRONG USER NAME.

15 ?INPUT/OUTPUT ERROR...filnam.ext FILE PROTECT FAILURE. The specified file is read- or write-protected against the user.

16 ?INPUT/OUTPUT ERROR...filnam.ext FILE BEING MODIFIED. The specified file cannot be accessed because it is currently being written.

17 ?INPUT/OUTPUT ERROR...filnam.ext UNDEFINED I/O ERROR. The monitor has returned an undefined I/O error condition on a LOOKUP or ENTER.

18 ?NO DEVICE ASSIGNED. The monitor has returned a no-device error condition on a LOOKUP or ENTER . (If encountered, this error indicates a TECO or monitor bug.)

19 ?DIRECTORY IS FULL. The output file cannot be created because the device directory is full.

20 ?DEVICE dev NOT AVAILABLE. The requested I/O device is not available.

26 Illegal character in filename.

27 Illegal character in user name.

28 ?NO FILE FOR INPUT. An input command has been given without opening a file for input. An ;R command must be given.

29 ?ERROR ON INPUT DEVICE. An error has occurred during input. The input file is released. The user may want to try to read the file again, but if the error persists, the user must return to his backup file.

30 ?NO FILE FOR OUTPUT. An output command has been given without opening a file for output. An ;W Command must be given.

31 Two arguments are supplied for an L Command.

32 Attempt to move the pointer beyond the buffer or to delete characters beyond the buffer.

33 A two-argument command has its second argument less than the first or specifies a character string which is outside the buffer.

34 Attempt to search for too long a character string.

35 ?SEARCH. A Search Command not preceded by the ":"-modifier and not within an iteration failed to find the requested character string. The conditions resulting from such an error are explained

36 A Q-register that does not contain text is referenced by an M Command.

38 There is a right angle bracket not matched by a left angle bracket.

39 Use of a ; when not in an iteration is illegal.

4Ø The " Command is used either without a numeric argument preceding it or without one of the letter G, L, E, N, or C following it.

42 An undefined command character has been given.

44 ?STORAGE CAPACITY EXCEEDED. Insufficient core is available for required expansion. The command requiring this core expansion cannot be executed.

46 No argument should be used with an EX or EG Command.

48 If TTYn is the user's console, or any other attached user's console, device TTYn may not be specified in an ;I or ;W command.

49 If an argument is used preceding an iteration, n<...>, it must be greater than Ø.

Appendix B

CHARACTER CLINIC

This appendix lists the 128 ASCII characters and describes the most convenient methods to insert a character into the buffer, to search for a character, or to identify a character. The main part of the appendix is a table which has seven columns, as follows:

1. The character code is given in decimal. This is the only radix available in TECO.

2. The character code is also given in octal. This is the radix usually used for character codes.

3. The Symbolic Name used in dialogs in this manual is given. For a graphical character, it is the character itself (on the device which printed this document); but for other characters it is a parenthesized mnemonic and does not represent the type-out literally.

4. The English Name used in descriptive text in this manual is given. It is based on DEC usage and the language of the ASCII standard.

5. The Key-In which causes a Model 33 Teletype to generate the character code is given. When a Carriage Return is transmitted, TENEX transforms it into an End-of-Line character. When a Control Command is transmitted, it is intercepted and executed.

6. The Type-Out produced at a Model 33 Teletype when the character code is encountered in the buffer is given. When a control character (except BEL, LF, or CR) or a lower case letter is actually received at a Teletype, nothing happens. However, as the entries in this column show, TENEX and TECO transform many outgoing characters into printed forms. When this transformation fails to identify the character unambiguously, the Access-Code Function can be used.

7. A reference to Insert and Search techniques is given. This entry refers to the "notes" which follow the table. If there is no entry, the character can be used normally in any Insert or Search Command.

Codes: Dec	(Oct)	Names: Symb	English	Teletype Key-In	Type Out	Insert & Search
Ø	(ØØØ)	(NUL)	Null	CTRL-SHIFT-P	↑@	
1	(ØØ1)	(↑A)	Control-A	CTRL-A	none	(↑V)(↑A)
2	(ØØ2)	(↑B)	Control-B	CTRL-B	none	(↑V)(↑B)
3	(ØØ3)	(↑C)	Control-C	CTRL-C	↑C	use 3I
4	(ØØ4)	(↑D)	Control-D	CTRL-D	none	(↑V)(↑D)
5	(ØØ5)	(↑E)	Control-E	CTRL-E	none	use 5I
6	(ØØ6)	(↑F)	Control-F	CTRL-F	↑F	
7	(ØØ7)	(BEL)	Bell	CTRL-G	↑G	
8	(Ø1Ø)	(BS)	Backspace	CTRL-H	↑H	use 8I
9	(Ø11)	(HT)	[Hor] Tab	CTRL-I	spaces	
1Ø	(Ø12)	(LF)	Line Feed	LINE FEED	line feed	
11	(Ø13)	(VT)	Vert. Tab.	CTRL-K	↑K	
12	(Ø14)	(FF)	Form Feed	CTRL-L	↑L	
13	(Ø15)	(CR)	Car. Ret.	RETURN	return	use 13I
14	(Ø16)	(↑N)	Control-N	CTRL-N	↑N	match
15	(Ø17)	(↑O)	Control-O	CTRL-O	↑O	
16	(Ø2Ø)	(↑P)	Control-P	CTRL-P	↑P	
17	(Ø21)	(↑Q)	Control-Q	CTRL-Q	none	match
18	(Ø22)	(↑R)	Control-R	CTRL-R	none	(↑V)(↑R)
19	(Ø23)	(↑S)	Control-S	CTRL-S	↑S	match
2Ø	(Ø24)	(↑T)	Control-T	CTRL-T	↑T	use 2ØI
21	(Ø25)	(↑U)	Control-U	CTRL-U	none	(↑V)(↑U)
22	(Ø26)	(↑V)	Control-V	CTRL-V	none	(↑V)(↑V)
23	(Ø27)	(↑W)	Control-W	CTRL-W	none	(↑V)(↑W)
24	(Ø3Ø)	(↑X)	Control-X	CTRL-X	↑X	match
25	(Ø31)	(↑Y)	Control-Y	CTRL-Y	none	(↑V)(↑Y)
26	(Ø32)	(↑Z)	Control-Z	CTRL-Z	↑Z	
27	(Ø33)	(ESC)	Escape	ALTMODE	↑$	use @
28	(Ø34)	(↑\)	Control-\	CTRL-SHIFT-L	↑\	
29	(Ø35)	(↑])	Control-]	CTRL-SHIFT-M	↑]	
3Ø	(Ø36)	(↑↑)	Control-↑	CTRL-SHIFT-N	↑↑	
or		(^^)	Control-^	CTRL-SHIFT-N	^^	
31	(Ø37)	(%)	End-of-Line	CTRL-SHFT-O	CR & LF	use (CR)
32	(Ø4Ø)	(SP)	Space	(space bar)	space	
33	(Ø41)	!	EXCL. POINT	SHIFT-1	!	
34	(Ø42)	"	Quote	SHIFT-2	"	
35	(Ø43)	#	Number Sign	SHIFT-3	#	
36	(Ø44)	$	Dollar Sign	SHIFT-4	$	
37	(Ø45)	%	Percent	SHIFT-5	%	
38	(Ø46)	&	Ampersand	SHIFT-6	&	
39	(Ø47)	'	Apostrophe	SHIFT-7	'	

Codes: Dec	(Oct)	Names: Symb	English	Teletype Key-In	Type Out	Insert & Search
40	(050)	(	Left Paren.	SHIFT-8	(	
41	(051)	)	Right Paren.	SHIFT-9	)	
42	(052)	*	Asterisk	SHIFT-:	*	
43	(053)	+	Plus	SHIFT-;	+	
44	(054)	,	Comma	,	,	
45	(055)	-	Hyphen	-	-	
46	(056)	.	Period	.	.	
47	(057)	/	Slash	/	/	
48	(060)	0	"0"	0	0	
49	(061)	1	"1"	1	1	
50	(062)	2	"2"	2	2	
51	(063)	3	"3"	3	3	
52	(064)	4	"4"	4	4	
53	(065)	5	"5"	5	5	
54	(066)	6	"6"	6	6	
55	(067)	7	"7"	7	7	
56	(070)	8	"8"	8	8	
57	(071)	9	"9"	9	9	
58	(072)	:	Colon	:	:	
59	(073)	;	Semicolon	;	;	
60	(074)	<	Less Than	SHIFT-,	<	
61	(075)	=	Equals	SHIFT--	=	
62	(076)	>	Greater Than	SHIFT-.	>	
63	(077)	?	Question Mrk	SHIFT-/	?	
64	(100)	@	At Sign	SHIFT-P	@	
65	(101)	A	"A"	A	A	
66	(102)	B	"B"	B	B	
67	(103)	C	"C"	C	C	
68	(104)	D	"D"	D	D	
69	(105)	E	"E"	E	E	
70	(106)	F	"F"	F	F	
71	(107)	G	"G"	G	G	
72	(110)	H	"H"	H	H	
73	(111)	I	"I"	I	I	
74	(112)	J	"J"	J	J	
75	(113)	K	"K"	K	K	
76	(114)	L	"L"	L	L	
77	(115)	M	"M"	M	M	
78	(116)	N	"N"	N	N	
79	(117)	O	"O"	O	O	

Codes: Dec	(Oct)	Names: Symb	English	Teletype Key-In	Type Out	Insert & Search
8Ø	(12Ø)	P	"P"	P	P	
81	(121)	Q	"Q"	Q	Q	
82	(122)	R	"R"	R	R	
83	(123)	S	"S"	S	S	
84	(124)	T	"T"	T	T	
85	(125)	U	"U"	U	U	
86	(126)	V	"V"	V	V	
87	(127)	W	"W"	W	W	
88	(13Ø)	X	"X"	X	X	
89	(131)	Y	"Y"	Y	Y	
9Ø	(132)	Z	"Z"	Z	Z	
91	(133)	[	Left Bracket	SHIFT-K	[	
92	(134)	\	Backslash	SHIFT-L		
93	(135)	]	Right Brackt	SHIFT-M	]	
94	(136)	↑	Up-Arrow	SHIFT-N	↑	
or		^	Caret	SHIFT-N	^	
95	(137)	←	Left Arrow	SHIFT-O	←	
or		_	Underscore	SHIFT-O	_	
96	(14Ø)	`	Grave Accent	none	@	use 96I
97	(141)	a	"a"	none	A	use 97I
98	(142)	b	"b"	none	B	use 98I
99	(143)	c	"c"	none	C	use 99I
1ØØ	(144)	d	"d"	none	D	use 1ØØI
1Ø1	(145)	e	"e"	none	E	use 1Ø1I
1Ø2	(146)	f	"f"	none	F	use 1Ø2I
1Ø3	(147)	g	"g"	none	G	use 1Ø3I
1Ø4	(15Ø)	h	"h"	none	H	use 1Ø4I
1Ø5	(151)	i	"i"	none	I	use 1Ø5I
1Ø6	(152)	j	"j"	none	J	use 1Ø6I
1Ø7	(153)	k	"k"	none	K	use 1Ø7I
1Ø8	(154)	l	"l"	none	L	use 1Ø8I
1Ø9	(155)	m	"m"	none	M	use 1Ø9I
11Ø	(156)	n	"n"	none	N	use 11ØI
111	(157)	o	"o"	none	O	use 111I
112	(16Ø)	p	"p"	none	P	use 112I
113	(161)	q	"q"	none	Q	use 113I
114	(162)	r	"r"	none	R	use 114I
115	(163)	s	"s"	none	S	use 115I
116	(164)	t	"t"	none	T	use 116I
117	(165)	u	"u"	none	U	use 117I
118	(166)	v	"v"	none	V	use 118I
119	(167)	w	"w"	none	W	use 119I

Codes: Dec	(Oct)	Names: Symb	English	Teletype Key-In	Type Out	Insert & Search
12Ø	(17Ø)	x	"x"	none	X	use 12ØI
121	(171)	y	"y"	none	Y	use 121I
122	(172)	z	"z"	none	Z	use 122I
123	(173)	{	Left Brace	none	[	use 123I
124	(174)	|	Vertic1 Line	none	none	use 124I
125	(175)	}	Right Brace	none	none	use 125I
126	(176)	¬	Not	none	none	use 126I
or		~	Tilde	none	none	use 126I
127	(177)	(DEL)	Delete	RUBOUT	none	use 127I

NOTES ON INSERT & SEARCH

(↑V)(↑X) -- This note applies to any Control Command not mentioned in other notes below. To use such a Control Command literally in either an Insert or Search Command, quote it; that is, precede it with a (↑V).

use nI -- This note applies to the Carriage Return and the Interrupt Control Commands. To insert such a character into the buffer, use the Insert-Code Command. To search for such a character, create a Search Command in the buffer as a character string, load it into a Q-Register by an Extract-Command, and execute it by a Macro Command.

match -- This note applies to the Match Control characters, (↑X), (↑S), (↑N), and (↑Q), described in the Search Command Group. To use one literally in an Insert Command, type (↑X), (↑S), (↑N), or (↑V)(↑Q), respectively. (Control-Q must be quoted because it is also a Control Command.) To use one literally in a Search Command, precede each with a quoted Control-Q giving (↑V)(↑Q)(↑X), (↑V)(↑Q)(↑S), (↑V)(↑Q)(↑N), and (↑V)(↑Q)(↑V)(↑Q), respectively.

use @ -- This note applies to an unquoted (↑D) or a quoted (ESC), either of which enters an Escape character into the command register. When such a character is included in an Insert or Search Command, the "@" option of the command must be used.

use (CR) -- This note applies to the End-of-Line character. This important character is normally entered by striking RETURN (which TENEX converts to an End-of-Line) and not, of course, by striking CTRL-SHFT-O.

Appendix C

COMMAND INDEX

The entries given here refer to characters as typed in at a terminal. For example, "(↑A)" means a typed Control-A (which is a Control Command) and not a Control-A which is actually in the Command Register (and which would be an illegal No-Op Command).

An entry with a star (*) is an illegal command which is not detected as an error by TECO and is instead given the interpretation mentioned in the entry. In particular, the entry (↑x) means that any control character which does not have a legal interpretation is treated as a No-Op Command by TECO rather than as an error.

Bolt Beranek and Newman Inc.

Text Editor and Corrector Manual

NIC. No.19937