DEC-10-ETEE-D

decsystem10 TECO
TEXT EDITOR AND CORRECTOR PROGRAM PROGRAMMER'S REFERENCE MANUAL

This manual reflects the software as of Version 23 of TECO.

digital equipment corporation · maynard, massachusetts

1st Printing January 1968
2nd Printing October 1968
3rd Printing August 1969
4th Printing April 1970
5th Printing (Rev) October 1970
6th Printing (Rev) May 1972

The material in this manual is for information purposes and is subject to change without notice.

NEW AND CHANGED INFORMATION

This manual reflects the software as of version 23. It has been revised to include all new and changed material since version 21A of the TECO software. Change bars in the left margin are used to indicate the new and revised information.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

APPENDICES

CONTENTS (Cont)

Page

TABLES

Chapter 1 Introduction

This manual is a complete reference manual for the advanced TECO user. It is not designed to be used as a beginner's text, and people who are learning TECO should not use it as such. Beginners are referred to the tutorial "Introduction to TECO", which appears in Section I of the DECsystem-10 Users Handbook.

TECO is a powerful text editor for use with all DECsystem-10 systems. TECO enables the advanced user to easily edit any ASCII text. Most editing can be accomplished using a few simple commands; or the user can select any of a large set of sophisticated commands, such as character string searching, command repetition, conditional commands, programmed editing, and text block movement. Refer to Appendix C for a summary of the commands available.

TECO editing is normally done on-line, using the terminal. However, the user can also write his editing commands as a TECO command file and have his editing task run by an operator.

TECO is a character-oriented editor; one or more characters in a line can be modified without retyping the rest of the line. Any source document can be edited: programs written in FORTRAN, COBOL, MACRO-10, or any other language, as well as memoranda, specifications, and other types of arbitrarily-formatted text. TECO does not require that line numbers or any other extraneous information be associated with the text. The full ASCII character set, printing and nonprinting characters alike, can be processed.

TECO requires a minimum of 5K of core memory, 3K of which is shared in a reentrant system. TECO takes advantage of any additional core available to expand its buffers, as required.

A single terminal is required for typing in commands. Data can be input or output on any standard I/O device.

Chapter 2
Concepts

2.1 DATA FILES

DECsystem-10 TECO operates on ASCII data files. The input file is the file that the user wishes to change. The output file is the file that receives the newly created or edited data.

Inputting is defined as the process of reading in data that already exists in some computer-readable form (paper tape, disk file, etc.). Data can be input from any device except the user's terminal (or another user's terminal). Inserting is defined as the actual typing in of new data and is done only at the user's terminal.

In the case of such hard-copy devices as the card reader and the paper-tape reader, only the device need be specified to open a file for input or output. For disk and DECtape files, filenames, as well as the device, must be specified. If no device is specified, the device DSK: is assumed. Magnetic tape files are specified by naming the tape drive and by using special TECO commands to position the tape properly.

Any I/O device name acceptable to the monitor can be used. Some examples are:

DSK:	Disk (including drums)
DTAn:	DECtape (n is the number of the drive on which the tape is mounted)
MTAn:	Magnetic tape (n is the number of the drive on which the tape is mounted)
CDR:	Card reader
CDP:	Card punch
PTR:	Paper-tape reader
PTP:	Paper-tape punch
LPT:	Line printer
TTYn:	Terminal number n, usually a terminal having a low-speed reader or punch

NOTE

TTYn: used as an I/O device must be different from the user's terminal and must not be the terminal of any attached user.

Filenames for disk and DECtape files consist of two parts: the first part, the filename proper, consists of from one to six alphanumeric characters; the second part, which is optional, is called the "filename extension." If given, the filename extension consists of from one to three alphanumeric characters and is separated from the filename proper by a period. If the filename extension is not given, it is defined as null and as such is distinctive. In the case of a null filename extension, the period after the filename proper can be omitted.

Examples of filenames:

TECO.21	The source file for TECO version 21
EARNNG .F4	A FORTRAN source program
0015J.CBL	A COBOL source program
GLOB.MAC	A MACRO-10 source program
GLOB.BAK	A backup file
FRMTTR.TEC	A file containing a TECO macro
M20	A filename with null extension
M20.1	A similar filename with non-null extension

2.2 CHARACTER SET

The TECO character set is the full ASCII set. To obtain particular information about individual characters, the user should refer to the table of ASCII characters in Appendix B. This table contains the following:

a. A list of all ASCII characters and the symbols used in this manual to represent them,

b. octal and decimal values of the characters, and,

c. comments concerning any special significance of each character.

In general, the user must be concerned with the character set on two levels: the data level and the command level.

Every ASCII character from control-A (decimal value 01) through rubout (decimal value 127) is legal in TECO data. They can all be input and output, and they can all be inserted. The only character that is not completely legal as data is the null character (decimal value 0). The null character can be inserted and output, but it is ignored on input. Form feed characters (decimal value 12) are completely legal in data but are treated specially on input (see Sections 2.3 and 3.3).

Most of the ASCII characters have some meaning when used as commands. Some are monitor commands. When used as commands, the lower-case characters have the same meaning as their upper-case equivalents. The table in Appendix B tells where in this manual the uses of the various characters as commands are explained.

2.2.1 Special Characters

Because of their use as special immediate-action commands (monitor control commands or erasing commands), certain characters must not be typed in explicitly as alphanumeric arguments. All of them, however, are legal as data (except the null character) and can be inserted using special techniques. The characters to which this restriction applies are referred to in this manual as "special characters." These special characters are listed in Table 2-1.

Table 2-1
Special Characters†

Character		Remarks
↑C	(control-C)	A monitor command
↑G ↑G	(two successive control-G's)	An erasing command (A single control-G is acceptable.)
↑G ␣	(control-G, space)	Immediate editing command (causes current line to be retyped).
↑O	(control-O)	A monitor command
↑U	(control-U)	An erasing command
ESCape or PREfix		Equivalent to ALTmode
ALTmode or ↑[		Standard text argument terminator (Two successive ALTmodes terminate a command string.)
Rubout		An erasing command

†In monitors preceding the 5.02 monitor the characters ↑B , ↑F , and ↑P are also monitor commands and must be included in the above list for these systems.

2.2.2 Control Characters

Control characters are characters that are typed by holding down the CTRL key while striking a character key. The control characters have decimal values 0 through 31. When TECO is printing text, a control character is printed as an up-arrow, followed by the character which is typed to produce the control character. For example, control-A prints as " ↑A".

In many cases the control character commands can be typed into command strings by using an alternate procedure to the standard method of holding down the CTRL key while striking the desired character.

Instead, the user can first type up-arrow and then type the desired character without depressing the CTRL key. For example, when used as a command, the two-character sequence up-arrow, H (denoted by ↑H) is equivalent to the single character control-H (denoted by (↑H)). This method can be used only when the control character is typed as a command, not when it is typed as text or as an alphanumeric argument. Control characters appearing as text arguments must be preceded by a (↑R) . Exceptions are noted at appropriate places throughout the manual.

2.2.3 Carriage Control Functions

A few of the control characters are the special terminal functions: bell, tab, line feed, vertical tab, form feed, and carriage return. All of these characters echo by performing their particular function; they also perform this function when TECO is printing out text from the buffer.

When a carriage return is typed in, the monitor automatically generates a line feed following it. The echo to the carriage return type-in is a carriage return followed by a line feed. If the carriage return is typed as an insert, a line feed is automatically inserted immediately after the carriage return.

Altmode (or escape or prefix) echoes and prints out as a dollar sign.

2.2.4 Symbols

In the examples in this manual, some special symbols are used to clearly indicate what the user must type. These special symbols are listed in Table 2-2.

In all examples containing both characters typed by the monitor or TECO and characters typed by the user, the characters typed by the monitor or TECO are underlined. Carriage control characters (carriage return, form feed, etc.) typed by the user are indicated through use of the special symbols.

Table 2-2
Special Symbols

Symbol	Character
⇥	tab
↓	line feed
(VT)	vertical tab
(FORM)	form feed
↵	carriage return
␣	space
($)	altmode
(RO)	rubout
(↑A)	control-A
↑A	up-arrow followed by A

(Other control characters similarly denoted)

2.3 DATA FORMAT

TECO is capable of editing text written in any format. There are, however, features in TECO that make use of the concept of a line and the concept of a page. Therefore, the user must know how these concepts are defined in TECO.

Lines can be of any length. The characters that define the end of a line are the line feed, vertical tab, and form feed. The end of the editing buffer also counts as an end-of-line character if there is no other end-of-line character at the end of the buffer. When TECO counts lines, it does so by counting these end-of-line characters. An end-of-line character is considered to belong to the line that it terminates.

Examples:

The following text comprises three lines of text as defined by TECO:

```
LINE ONE↓
          LINE TWO↵ ↓
LINE THREE↵ ↓
```

The following text is considered to be two lines:

BEGINNING↵ OVERPRINT VT CONTINUATION ↵↓

The first line is terminated by the VT character and the second by the ↓ character.

Text to be edited by TECO does not have to contain end-of-line characters; however, if it does not contain them, those features of TECO that count lines will not be useful.

NOTE

If the EO value has been set to 1, the only end-of-line character is the line feed (refer to Paragraph 3.17.3 for a description of the EO value).

Pages are defined in TECO by form feed characters, which act as page separators. They are not considered to belong to either of the two pages that they separate. Two consecutive form feed characters delimit a null page. A form feed charater at the beginning of a file delimits a null page at the beginning of the file. A form feed character at the end of a file has no effect in TECO. It can be omitted.

Examples:

The following file consists of two pages:

```
LINE ONE ↵↓
LINE TWO ↵↓
FORM LINE THREE ↵↓
LINE FOUR ↵↓
```

The following consists of four pages; the first and third pages are null:

```
FORM LINE ONE ↵↓
LINE TWO ↵↓
FORM FORM LINE THREE ↵↓
LINE FOUR ↵↓
```

TECO operates most efficiently with files that are divided into pages of approximately fifty or fewer lines. Files with longer pages or files containing no form feed characters can be edited with TECO; but, this process requires either additional core storage or more care when editing.

The processing of form feed characters by TECO must be thoroughly understood by the user. The page concept is further discussed in relation to the size of the editing buffer in Section 2.4, and the relation of form feed characters to input and output commands is discussed in Sections 3.3, 3.9, 3.10, and 3.11.

TECO may be used to edit files containing the special line-sequence numbers produced by BASIC, the PIP /S switch, LINED, and several other editors, but TECO does not need these numbers and makes no special use of them (nor does it destroy them). See Section 3.2.6 for an explanation of how these numbers may be processed.

2.4 EDITING BUFFER

Editing is accomplished by:

a. Reading text into the editing buffer
b. Making changes to the text in this buffer
c. Writing the modified text out to a new file

The editing buffer is a block of core memory within TECO. Data is put in the editing buffer when it is read in or inserted; it is kept in the editing buffer while it is being modified.

Text is packed in the editing buffer with five 7-bit ASCII characters per 36-bit word. When TECO is running in the minimum 5K of core, the editing buffer holds approximately 3600 characters. Each additional 1K of core assigned to TECO increases the size of the editing buffer by 5120 characters.

TECO normally passes data into and out of the editing buffer a page at a time. Pages are delineated by form feed characters (see Sections 2.3 and 3.3).

2.5 BUFFER POINTER

TECO is a character-oriented editor, therefore, the concept of the buffer pointer must be understood by the user. The position of the buffer pointer determines the effect of many editing commands. For example, insertion and deletion always take place at the current position of the buffer pointer.

The buffer pointer is a movable position indicator. It is always positioned between two characters in the editing buffer, or before the first character in the buffer, or after the last character in the buffer. It is never positioned exactly <u>on</u> a particular character; it is positioned either immediately <u>before or after</u> the character.

The pointer can be moved forward or backward over any number of characters. It cannot be moved beyond the boundaries of the buffer; i.e., it cannot be moved further back than the position immediately prior to the first character in the buffer, and it cannot be moved further ahead than the position immediately after the last character in the buffer.

In the examples in this manual showing text in the editing buffer, the position of the buffer pointer is shown by a caret (^) directly under the line of text.

Example:

```
TEXT IN THE EDITING BUFFER
              ^
```

When discussing text in the editing buffer in terms of lines, the phrase "current line" is frequently used. The current line is the line at which the buffer pointer is currently directed. The pointer can be positioned either at the beginning of the line or in the interior of the line.

2.6 GENERAL COMMAND STRING SYNTAX

Commands are given to TECO by typing a command string; command strings are formed by writing a series of commands, one immediately after the other, and concluding with two consecutive altmodes (refer to Appendix C for a summary of commands).

A command string may be typed after TECO indicates that it is ready by printing an asterisk. An example of a command string is as follows:

```
*YIHEADING $ 2K4DNTAG $ 2LT $$
```

Execution of the command string begins only after the two consecutive altmodes have been typed. TECO then indicates that it is beginning execution of the command string by typing a carriage return-line feed. At that point, each command in the string is executed in turn, starting at the left. When all commands in the string have been executed, TECO prints another asterisk indicating it is ready to accept another command string.

If a command in the string cannot be executed due to a command error, execution of the command string stops at that point, and an error message is printed. Commands preceding the command in error are executed. The erroneous command and the commands following it are not executed. Errors, error messages, and recovery techniques are fully discussed in Chapter 5.

There are exceptions to the general rule that commands are not executed until the end of the command string has been indicated by two consecutive altmodes. These exceptions are the commands listed in Table 2-1 in Section 2.2.

2.7 ARGUMENTS

2.7.1 Alphanumeric Arguments

Most alphanumeric arguments are text arguments that are interpreted as ASCII data by TECO. Some examples of text arguments are: data to be inserted in the buffer, search character strings, and command string tags. Other types of alphanumeric arguments are device and filenames and Q-register names.

An alphanumeric argument always follows the command to which it applies. As a rule, most commands that take text arguments require that the argument be terminated by an altmode; however, there are exceptions to this rule which are explained at appropriate places in the manual.

An altmode used to terminate an alphanumeric argument can also function as one of the two altmodes necessary to terminate a command string.

Example:

```
*ITEXT $ STEXT2 $$
*
```

The alphanumeric argument, "TEXT", is terminated by an altmode. The second argument, "TEXT2", is also terminated by an altmode, but this altmode is also used as one of the altmodes terminating the command string.

Any printable ASCII character is legal in an alphanumeric argument with the exception of the special characters listed in Table 2-1, Section 2.2. In addition, non-printing characters are legal when they are preceded by a ↑R.

2.7.2 Numeric Arguments

Numeric arguments always precede the command to which they apply. In some cases, only a single numeric argument is required; in others, a pair of numeric arguments is required.

When two numeric arguments are used, they are separated by a comma. In most cases, numeric arguments must be positive; however, some commands allow a numeric argument to be negative or zero. The number and type of numeric arguments allowed by each command are stated in the section in which that command is explained.

Where a numeric argument is used to specify a buffer position, the number used is the number of characters in the buffer to the left of that position. Thus, n means the position to the right of the nth character in the buffer (between the nth and n+ 1st characters).

Numeric arguments used in pairs are always buffer position arguments. Such a pair specifies all the characters in the buffer that lie between the two buffer positions represented by the two arguments. This definition is precise because the term "buffer position" always indicates a position before or after a given character, not "on" or "at" the character.

Example:

12,20	This argument pair specifies the thirteenth (13th) through the twentieth (20th) characters in the buffer. These characters are specified because the 12 indicates the position between the 12th and 13th characters, and the 20 indicates the position between the 20th and 21st characters.

Numeric arguments can be used in arithmetic/logical combinations. The characters shown in Table 2-3 are used as operators.

Table 2-3
Numeric Operators

Operator	Function	Example
+	Ignored, if used before the first term in a string.	+2=2
+	Addition, if used between two terms.	5+6=11
space	Equivalent to +.	␣2=2 5␣6=11
-	Negation, if used before the first term in a string.	-2=-2

Table 2-3 (Cont)
Numeric Operators

Operator	Function	Example
-	Subtraction, if used between terms.	8-2=6
*	Multiply. (Used between two terms.)	8*2=16
/	Integer Divide (and drop the remainder). (Used between two terms.)	8/2=4 8/3=2
&	Bitwise logical AND of the binary representations of two terms, if used between the terms.	12 &10=8
#	Bitwise logical OR of the binary representations of two terms, if used between the terms.	12# 10=14

When more than one arithmetic/logical operator is used in a single numeric argument, the operations are performed from left to right. This sequence can be overridden through use of parentheses (). All operations within parentheses are performed before those outside parentheses. Parentheses can be nested.

In TECO, numbers are ordinarily assumed to be decimal integers. Preceding a number with ↑O (uparrow-O, not control-O) causes the number to be read in octal radix.

Example:

```
↑O177 is equivalent to 127.
```

Examples:

```
3*↑O10=24
2+3 * 4=20
2+(3 * 4)=14
2+(3 * (16/(3-1) ) /2 +(2 * 5) ) =24
2&(3#5) # 16=18
-( (2+ (3 * 4) - 1 &(6 +8) ) /2) =-6
```

The arithmetic/logical operators and parentheses can be used to form one or both of the numeric arguments in a pair.

Example:

```
260 - (3 * 42), 250 + (77/3)
```

2.7.3 Commands That Return a Value

Generally speaking, there are two main categories of TECO commands: 1) those that perform some operation, such as inserting text, and 2) those that "return" a value, such as the number of characters in the editing buffer. (There are also some commands that do both.)

A command is said to "return" a value if the command causes the current value of some quantity to be calculated, and then the command takes on this value, becoming itself a numeric argument that may be used by another command. Using such a command is equivalent to typing the particular number that the command returns as a value, except that the value is not usually known in advance. This value can then be used as an argument by the next command in the command string, provided that the command is one that can take a numeric argument. Otherwise, it is ignored.

An example of a command that returns a value is the Z command (see Section 3.4). The Z command returns a value equal to the number of characters in the buffer. It has no other function. Thus, in order to be useful, Z must be used as a numeric argument preceding another command.

Commands that return values may be used in arithmetic/logical combinations with each other and with explicit numbers. All the same rules apply. Each command that returns a value has all the properties of a number that has been explicitly typed in.

If commands that return values are concatenated with each other or with digits, the value returned is that of the last command or number in the string. An operator preceding such a string continues to apply.

Examples:

```
ZZ = Z
Z48 = 48
-2Z = -Z
3+ZZ = 3+Z
```

2.8 Q-REGISTERS

Q-registers are data storage registers that are available to the TECO user. Q-registers give a great amount of editing power to the user by enabling programmed editing and text block movement. Data stored in Q-registers is not disturbed by the flow of data into and out of the editing buffer. It can be preserved throughout an entire TECO job, and it is available for retrieval or change at any time.

There are 36 Q-registers; each Q-register has a single character name, which is either one of the digits 0 through 9, or one of the letters A through Z. Also, there is a Q-register pushdown stack that effectively makes available an additional 32 Q-registers for certain applications.[1]

[1] The number of entries in the pushdown stack can be increased by changing the parameter LPF in TECO.MAC and reassembling TECO.

Two types of data can be stored in Q-registers: decimal integers or alphanumeric character strings.

For numeric storage, a Q-register can be used to hold a single positive, negative, or zero decimal integer in the range $-2^{35} + \leq n \leq 2^{35} -1$. Numbers stored in Q-registers can be incremented, tested, or recalled. Hence, Q-registers can be used as switches and counters, as well as for simple data-save functions.

For text storage, a Q-register can be used to hold a character string of any length. Two types of character strings can be stored: ordinary text and TECO command strings. Ordinary textual data stored in a Q-register is copied into the Q-register from the editing buffer without destroying the copy in the editing buffer. Storing text in a Q-register is useful for functions such as making many copies of a given segment of text throughout a file without retyping it each time, for moving a block of text from one position to another in a file, and for moving a block of text to another file.

Textual data in the form of TECO command strings can also be stored in Q-registers. Such a command string can be executed over and over throughout an editing job, much like calling a subroutine. This feature also enables an editing job to be typed up off-line and then executed by an operator at a later time. Such command strings can be edited just as any other text.

2.9 CORE EXPANSION

The minimum 5K of core memory is allocated within TECO in the following manner. The executable code is allocated 3K of core memory; this code is pure and is shared in a reentrant system. The other 2K of core memory is allocated to the data segment. Part of the data segment is used for program variables and fixed-length I/O buffers, while the rest is used for three variable-length storage areas:

a. The editing buffer,

b. the command string buffer, and

c. the storage area for Q-registers containing text.

When TECO is initialized, the three variable-length storage areas are assigned a specific amount of space. After a command string is executed, the command string buffer is cleared. When text is deleted from the editing buffer, the formerly occupied space is reclaimed. However, during a TECO job, conditions can arise where the available space is not sufficient for the three variable-length storage areas. For example, a command string having a single insert command with many lines of text to be inserted may overflow the command string buffer. In such a case, TECO attempts to obtain the required space from one of the other variable-length storage areas. If, however, all three areas are filled to such an extent that the total amount of space allotted to all three is insufficient, TECO automatically requests another 1K of core memory from the monitor.

If the request for more core is successful, operation continues normally. TECO prints a message of the form "[nK CORE]" (where n is the new number of 1K segments of (low) core allocated to the user) to inform the user that his core has been expanded to the specified amount. (This message is suppressed while the user is typing a command string.) If the request for more core is unsuccessful, TECO stops execution of the command string at this point and prints the error message ?COR Storage Capacity Exceeded.

Chapter 3
Commands

3.1 INITIALIZATION COMMANDS

TECO is called by giving one of three different initialization commands to the monitor. An initialization command can be given whenever the monitor has typed a period to indicate that it is waiting for a new command.

3.1.1 R TECO Command

The general TECO initialization command is the command:

```
.R TECO↵
*
```

This command calls TECO into core and initializes the program for general use. It does not automatically initialize any particular devices or files for input or output.

When initialization is complete, an asterisk is typed to indicate that TECO is ready to receive a command. This state, in which TECO waits for command string type in, is called command mode or the idle state.

The R TECO command can be given with an argument:

```
.R TECO n↵
```

where n is a decimal integer. The argument is used to request more than the minimum of 5K of core memory for the TECO job. If n is greater than 5, the monitor initializes the user's TECO job with nK of core, if possible. If n is not greater than 5, it has no effect.

3.1.2 MAKE Command

The two main uses of TECO are (1) to create a new file, and (2) to edit an existing file. These two uses are so common that there are special monitor commands to initialize TECO for executing them.

The command:

```
.MAKE dev:filnam.ext[proj,prog]↵
```

is used to initialize TECO for creating a new file. Filnam.ext is the name that the user, using this command, gives to the new file. Dev: is the device on which the file is to be created; it can be any output device. If dev: is omitted, DSK: is assumed. If the output device is a disk device, [proj,prog] is used to specify the user area in which the file is to be created; if [proj,prog] is omitted and the device is DSK:, the file will be created in the user's own disk area. For a more precise explanation of file specifications (dev:filnam.ext[proj,prog]), see Section 3.2.1.

The MAKE command opens a new file to receive output from TECO and gives it the name specified. Once the file has been opened, it is then actually created by using the insert and output commands.

Care should be used in the choice of the filename used with a MAKE command. If there is already a file on the system device with the name specified, the MAKE command will cause the old file to be overwritten and TECO will output the warning message %SUPERSEDING EXISTING FILE. If the user does not wish to supersede the file, he should type ↑C to return to the monitor. If no filename is used with a MAKE command, the name of the last ASCII file used in a MAKE command or any other edit-class command (MAKE, TECO, EDIT, or CREATE) is used. If no filename is given in a MAKE command and no edit-class command was previously given, the error message "COMMAND ERROR" is typed.

When initialization is completed, TECO types an asterisk to indicate its readiness to receive a command string. Usually the first command following a MAKE command is an insert command.

```
.MAKE dev:filnam.ext[proj,prog] ↵
```

is equivalent to

```
.R TECO ↵
*EWdev:filnam.ext[proj,prog] $ $
```

3.1.3 TECO Command

The command

```
.TECO dev:filnam.ext[proj,prog] ↵
```

is used to initialize TECO for editing an existing file on disk or DECtape. The file specifications dev:filnam.ext[proj,prog] are interpreted in the same way as for the MAKE command, except that the device must be a directory-structured device (disk or DECtape).

The filename and filename extension must be exactly the same as those of the file that is to be edited.

The TECO command opens the specified file for input and reads in the first page of that file. It also opens a new file, with a temporary name, for output of the edited version. The temporary name is of the form nnnTEC.TMP, where nnn is the user's job number, including leading zeros. When output of the new version is completed, the original (input) version of the file is automatically renamed filnam.BAK, and the new version is given the name of the original file. This operation is identical to that used for the EB command (see Section 3.2.5).

If no filename is specified in a TECO command, the name of the ASCII file last referenced in any edit-class command is assumed. If no filename is specified and no edit-class command has previously been given, the error message "COMMAND ERROR" is typed. The TECO command cannot be used with a file having the filename extension .BAK, nor with a file name nnnTEC.TMP, where nnn is the user's job number.

When initialization is completed, TECO types an asterisk to indicate its readiness to receive a command string.

The command

```
        .TECO dev:filnam.ext[proj,prog]↵
```

is equivalent to

```
        .R TECO
        *EBdev:filnam.ext[proj,prog] $ Y $$
```

If the project-programmer number specified in a TECO filnam.ext[proj,prog] command is different from the user's project-programmer number, the action of the TECO command is somewhat different from that of the standard TECO command explained above. In this case the named file is taken for input from the specified project-programmer area, but the output file is written in the user's own disk area with the same name as the input file. This operation is identical to that used for the EB command (see Section 3.2.5).

If [proj,prog] is not the user's project-programmer number, the command:

```
        .TECO filnam.ext [proj,prog]↵
```

is equivalent to

```
        .R TECO↵
        *EBfilnam.ext[proj,prog] $Y$$
```

or to

```
        .R TECO↵
        *ERfilnam.ext [proj,prog] $EWfilnam.ext$Y$$
```

and the input file is not renamed to filnam.BAK.

NOTE

The R TECO command must be used for jobs involving editing a file on a device other than disk or DECtape, or for editing a file named nnnTEC.TMP, or a file with the filename extension .BAK. The R TECO command is also preferred with complex editing jobs, where user errors are likely, because of the greater control it gives over the input and output files. The R TECO command requires the use of file selection commands (see Section 3.2), whereas the MAKE and TECO commands do not.

3.1.4 Examples of the Use of Initialization Commands

```
.MAKE EARNNG.F4↵
*
```

This command initializes TECO for creation of a FORTRAN file named EARNNG.F4.

```
.TECO LIB40.MAC ↵
```

This command initializes TECO for editing the existing file LIB40.MAC. At the completion of editing, TECO automatically changes the name of the original version of LIB40.MAC to LIB40.BAK and gives the name LIB40.MAC to the new version.

```
.TECO ↵
```

This initializes TECO for editing the disk file last referenced in an edit-class command (MAKE, TECO, EDIT, or CREATE).

```
.R TECO ↵
*
```

This is the command to initialize TECO for general-purpose editing. FILE selection commands (see Section 3.2) should follow.

3.2 FILE SELECTION COMMANDS

File selection is the specification of the device from which input is to be taken and the device to which output is to go. In the case of magnetic tape, file selection also involves positioning the tape. In the case of directory-structured devices, disk and DECtape, a filename must be specified in addition to the device.

If the user wants only to create a file, or to edit an existing disk or DECtape file, file selection can be done by using either of the previously described initialization commands.

```
.MAKE dev:filnam.ext[proj,prog]↵
        or
.TECO dev:filnam.ext[proj,prog]↵
```

In all other cases, and in particular if the user initializes TECO with the R TECO command, one or more of the file selection commands described in this section must be used.

3.2.1 ER Command

The ER command is used to select a file for input. The general form is

```
        *ERdev:filnam.ext [proj,prog] ($)
```

where

a. dev: is the device name, which can be any name acceptable to the monitor. The device name must be followed by a colon. If dev: is omitted, the default value DSK: is assumed.

b. [proj,prog] is ignored when used with a device other than disk. proj is the project number and prog is the programmer number of the disk area where the specified file resides or, in the case of output, is to be written. If [proj,prog] is omitted and the device is a disk, the user's project-programmer number is assumed.

c. filnam.ext need be used only if the device is a directory device, i.e., disk or DECtape. filnam is the one-to-six character filename, and ext is the one-to three character filename extension conforming to the rules stated in Section 2.1. If the device is a disk or DECtape, filnam must not be omitted; .ext must not be omitted unless the null extension is explicitly intended.

d. The ($) (altmode) functions as the argument terminator.

The ER command terminates input from any file that may have been previously opened for input, and then opens the specified file for input.

The user may open one file for input, read only part of that file, and then, with another ER command, release the first file and open a new file for input. It is not necessary to read to the end of a file before opening another. However, opening the second file does end input from the first. There is never more than one input file active. In Section 4.4, an example is given showing how to use multiple ER commands to merge parts of several files. Data cannot be input without first giving an ER, or equivalent, command.

3.2.2 EM Command

EM commands are used to position a magnetic tape for input or output. However, EM command apply only to the magnetic tape that is currently open for input (i.e., opened by the latest ERMTAn: ($) command). To position a magnetic tape for output, it is necessary to first initialize the tape for input, then do the desired EM function, and then reopen the device for output.

The function of an EM command is determined by the value of a single numeric argument preceding the EM. The various EM commands are shown in Table 3-1.

3.2.3 EW Command

The EW command is used to select a file for output. The general form is

```
        *EWdev:filnam.ext[proj,prog] ($)
```

The EW command opens the specified file for output. If any output file is already active, a new EW command closes that file before opening the new file. Only one output file can be active at any one time. If a previously active output file is closed by an EW command, that closed file contains all and only that data supplied to it by output commands preceding the new EW command.

If there is already an output file with the name specified, the EW command causes the old file to be overwritten and TECO outputs the warning message %SUPERSEDING EXISTING FILE.

Multiple EW commands may be used without changing the input file. In Section 4.3, an example is given showing how to use this technique in order to split a single input file into several parts.

The MAKE filnam.ext initialization command causes an automatic EWDSK:filnam.ext Ⓢ command to be executed. Output may not be done without first giving an EW, or equivalent, command.

Table 3-1
EM Commands

Command	Function
EM or 1EM	Rewind the currently-selected input magnetic tape to load point.
3EM	Write an end-of-file record on the input tape.
6EM	Skip ahead one record.
7EM	Back up one record.
8EM	Skip ahead to logical end-of-tape (defined by two successive end-of-file marks). The 8EM command leaves the tape positioned between the two end-of file marks so that successive output correctly overwrites the second EOF.
9EM	Rewind and unload.
11EM	Write 3 in. of blank tape.
14EM	Advance tape one file. This leaves the tape positioned so that the next item read will be the first record of the next file (or the second end-of-file mark at the logical end-of-tape).
15EM	Backspace tape one file. This leaves the tape positioned so that the next item read will be the end-of-file mark preceding the file backspaced over (unless the file is the first on the tape).

NOTE

The EM commands do not clear the internal input buffers. It is best to reinitialize with a new ER command before doing an EM command.

3.2.4 EZ Command

The EZ command is used only with disk, DECtape, or magnetic tape. Its function is equivalent to that of the EW command except that before opening the specified output file it zeros the output device directory if the device is a disk or DECtape, or it rewinds the tape if the device is a magnetic tape. For other devices, it is treated exactly like an EW. The form is

```
*EZdev:filnam.ext[proj,prog] ($)
```

3.2.5 EB Command

The EB command is used to open a file for editing in a manner similar to the initialization command TECO dev:filnam.ext[proj,prog]↵ . It can be used only for files on a disk or DECtape. The general form of the command is

```
*EBdev:filnam.ext[proj,prog] ($)
```

The exact operation of the EB command is as follows:

First, the EB command executes an automatic ERdev:filnam.ext ($) command, opening the specified file for input and releasing any previously opened input file. Then, it opens a temporary file to receive the output of the edited version of the input file. This temporary file is named nnnTEC.TMP, where nnn is the user's job number with leading zeros. This action is equivalent to executing the command EWdev:nnnTEC.TMP ($) . The output device is the same as the input device. Finally, the EB command sets an internal flag indicating that special action must be taken when the EB file is closed (by an EF, EX, or EG command - see Sections 3.9 and 3.10). It also prohibits any further EW, EZ, or EB commands until the file is closed.

When the EB file is closed, the following action takes place. First, if there already exists on the device a file with the name filnam.BAK, it is deleted. Then, the input file filnam.ext is renamed filnam.BAK. Finally, the output file, nnnTEC.TMP, is renamed filnam.ext.

The effect of using the EB command is analogous to editing a file in place, to itself, and converting the original version into a backup file. It updates the specified file and keeps the most recent previous version as a backup file.

If the project-programmer number specified in an EBfilnam.ext[proj,prog] ($) command is different from the user's, then the input file is taken from the specified area, but the output file is written in the user's own area with the same name as the input file. In other words, if [proj,prog] is not the user's project-programmer number,

```
*EBfilnam.ext [proj,prog] ($)
```

is equivalent to

```
*ERfilnam.ext [proj,prog] ($) EWfilnam.ext ($)
```

The EB command cannot be used with a file having the filename extension .BAK nor with a file named nnnTEC.TMP. The TECO dev:filnam.ext[proj,prog]↵ initialization command causes an automatic EBdev:filnam.ext[proj,prog] ($) to be executed (followed by an automatic Y command).

3.2.6 Editing Line-Sequence Numbered Files

Some ASCII files, e.g., those created by BASIC, PIP with the /S and /O switches, and LINED, have a special type of line number at the beginning of each line. These "line-sequence numbers" conform to certain rules so that they may be ignored or treated specially by compilers and other programs. The standards for line-sequence numbers are given in the LINED Program Reference Manual.

TECO does not need line-sequence numbers for operation, but TECO can be used to edit files containing them. If such a file is edited with TECO the line-sequence numbers are, in the normal case, simply preserved as additional text at the beginning of each line. The line-sequence numbers may be deleted, edited, and inserted exactly like any other text. On output the line-sequence numbers are output according to the standard, except that the tab after the number is output only if it is already there. Leading zeros are added as necessary. If a line without a line-sequence number is encountered, a line-sequence number word of five spaces is placed at the beginning of the line.

The following switches are available for use with line-sequence-numbered files. These switches are merely added to the appropriate file selection command.

```
ERdev:filnam.ext[proj,prog]/SUPLSN $ $
EBdev:filnam.ext[proj,prog]/SUPLSN $ $
```

causes line-sequence numbers to be suppressed at input time. The numbers will not be read into the editing buffer. Also, the tabs following the line-sequence numbers, if they exist, will be suppressed.

```
EWdev:filnam.ext[proj,prog]/SUPLSN $ $
```

causes the line-sequence numbers to be suppressed at output time. Tabs following the line-sequence numbers will also be suppressed if they exist.

```
EWdev:filnam.ext[proj,prog]/GENLSN $ $
EBdev:filnam.ext[proj,prog]/GENLSN $ $
```

causes line-sequence numbers to be generated for the output file if they did not already exist in the input file. Generated line-sequence numbers begin at 00010 and continue with increments of 10 for each line.

Note that these switches are needed only if a change is to be made in the format of the file being edited. If no switches are specified, a file is output in the same form as it was input.

3.2.7 Examples of the Use of File Selection Commands

```
*ERDTA2:CREF.2 $ EWDSK:CREF.3 $ $
*
```

This command string selects the DECtape file CREF .2 on DECtape drive 2 for input and opens a file called CREF.3 on the disk for output. If there is a file named CREF.3 already on the disk, it will be overwritten.

```
* ERCDR: ($) EWPTP: ($)($)
*
```

Select the card reader for input and the paper tape punch for output.

```
*ERMTA1: ($) EM14EM14EMEZDTA5:PROFIT.CBL ($) ($)
```

This command string selects the tape on magnetic tape drive 1 for input, then positions the tape at the beginning of the third file on that tape, and finally zeros the directory of the DECtape on drive 5 and opens an output ffle named PROFIT.CBL on it.

```
*ERPULSE.F4[]1,141] ($)($)
*
```

Select the file PULSE.F4 in project-programmer area [11,14] on the disk for input. If this file is read-protected against the current user, an error message results.

```
* EZMTA3: ($)($)
*
```

Rewind the magnetic tape on drive 3 and select it for output.

```
*ERMTA1: ($) 8EMEWMTA1: ($)
```

To position a magnetic tape for output (other than just a rewind), the user must first select the tape for input, then use EM commands to position the tape, and finally select the tape for output. In this example, the 8EM command positions the tape at the end of data that had previously been written on the tape. This enables new output to the tape without overwriting any of the previous data.

```
* EB22.F4 ($)($)
*
```

This command selects the disk file 22.F4 for editing. When the editing is completed, the file 22.F4 is the new version. The old version is changed to the backup file 22.BAK, and any previous backup file 22.BAK is deleted.

```
* n<14EM> ($)($)
*
```

Advance magnetic tape n files.

```
*EBCHESS.MAC[1,4] ($)($)
```

This command opens the file CHESS.MAC on the [1, 4] disk area for input, and opens a file CHESS.MAC on the user's own disk area for output (assuming the user's project-programmer number is not [1,4]).

3.3 INPUT COMMANDS

Input commands are used to read data from the input file, which must previously have been opened, into the editing buffer. Input commands can be used only after an ER command (or the equivalent) has been given. Input always begins at the beginning of the selected input file. Successive input commands then read successive segments of data from the input file.

The amount of data read on an input command depends on the buffer size, the particular input command used, and the data itself, as explained in the paragraphs below.

3.3.1 Y Command

The Y (yank) command first clears the editing buffer and then reads text into the buffer until one of the following conditions is met:

a. The end of the input file is reached;

b. A form feed character is read;

c. the buffer is two-thirds full and a line feed is read (or filled to within 128 characters of capacity);

d. the buffer is completely filled.

The usual effect of the Y command is to clear the editing buffer and then read the next page of the input file into it. Less than the entire next page is read in only if that page is too large to fit within two-thirds of the buffer's capacity. If the cleared buffer is not large enough to accommodate at least 3000 characters, TECO automatically expands its buffer by 1K, if possible, before beginning to input. The user is notified of the buffer expansion by a message of the form [nK CORE], where n is the new number of 1K segments of (low) core allocated to the user.

If the end of the input file has previously been read, the Y command only clears the buffer.

If a form feed is read (i.e., if input stops because of condition b), the form feed flag (↑E) is set to -1. The form feed itself is not packed in the buffer with the rest of the text. A succeeding input command begins input at the character following the form feed. If a form feed is not read, the form feed flag is set to 0, and the next input command begins input at the character following the last character previously read in. The form feed flag may be tested by the user (see Section 3.16), but ordinarily this is not necessary.

A single Y command is automatically executed by the TECO filnam.ext initialization command causing the first page of the input file to be read into the buffer before TECO prints the first asterisk.

The Y command sets the buffer pointer to the position preceding the first character in the buffer.

The Y command does not accept a numeric argument. If multiple Y commands are desired, n<Y> (where n is the number of pages to be ignored) can be typed.

3.3.2 A Command

The A (append) command reads in the next page of the input file without clearing the current contents of the editing buffer. The new input data is appended to that which is already in the buffer (at the end of that data). The position of the buffer pointer is not changed. If there was a form feed character in the input file separating the data already in the buffer and the data read in, it is removed. Thus, the A command can be used to combine several pages of a file.

If the editing buffer does not have sufficient space to accommodate 3000 more characters, TECO automatically expands its buffer by 1K, if possible, and then completes execution of the A command. The user is notified of the buffer expansion by a message of the form [nK CORE].

Input begun by an A command is terminated by any of the same four conditions that terminate a Y command. The A command processes form feeds and the form feed flags in the same manner as the Y command.

The A command does not accept a numeric argument. If multiple appends are desired, the user can type n<A> where n is the number of pages to be appended to the buffer. Note that nA is a different command (refer to Paragraph 3.16).

If the end of the input file was previously read, the A command has no effect.

3.3.3 Examples of the Use of Input Commands

*ERREPORT.CBL $ Y $$ *	This command string opens the disk file REPORT.CBL for input and reads in the first page of that file.
*YA $$ *	This deletes the page of text currently in the editing buffer, reads in the next two pages of the current input file, appending the second page to the first.
*A $$ [3K CORE] *	This inputs the next page of the file, appending it to the data already in the buffer. The previous contents of the buffer are not altered and the pointer is not moved.
	The buffer is expanded automatically, as required by the A command. In most cases, this message is of no concern to the user. It is important only if the system is nearly overloaded.
*ERDTA6:DATA.DOC $ YYY $$	This command string reads in and discards the first two pages of the DECtape file DATA.DOC, and then reads in the third page of that file.

3.4 SPECIAL CHARACTERS AS BUFFER POSITION NUMERIC ARGUMENTS

In many cases, numeric arguments are used to specify buffer positions. Because such arguments tend to be large and not easily determined by counting, the buffer positions commonly used as arguments are represented by special characters. These special characters are shown in Table 3-2.

Table 3-2
Special Buffer Position Arguments

Character	Value
B	Equivalent to 0. It represents the position at the beginning of the buffer, i.e., preceding the first character in the buffer.
Z	Equals the total number of characters in the buffer. Thus, Z represents the position at the end of the buffer, immediately after the last character in the buffer.
. (period)	Equals the number of characters to the left of the current position of the buffer pointer, and hence represents the buffer pointer position itself.
H	Equivalent to the numeric argument pair B, Z. Thus, in those commands that take two numeric buffer position arguments, H represents the whole of the buffer. This letter is particularly useful with type-out and output commands.

The characters B, Z and . can be used in arithmetic expressions.

3.5 BUFFER POINTER POSITIONING COMMANDS

This section describes the most elementary commands for moving the buffer pointer. In addition to these elementary commands, the search commands make up an entire set of powerful pointer-positioning commands. The search commands are described in Section 3.11.

3.5.1 J Command

The nJ command moves the buffer pointer to the position immediately after the nth character in the buffer. The command 0J moves the pointer to the beginning of the buffer, i.e., to the position immediately preceding the first character in the buffer. The command J, not preceded by an argument, is equivalent to 0J.

3.5.2 C Command

If n ≥ 0, nC moves the pointer forward over n characters in the buffer. If n <0, nC moves the pointer backward over n characters. The nC command is equivalent to . +nJ. The command C is equivalent to 1C; -C is equivalent to -1C.

3.5.3 R Command

The R command is equivalent to -C. The nR command is equivalent to -nC. If n ≥0, nR moves the pointer backward over n characters in the buffer. If n <0, nR moves the pointer forward over n characters. The nR command is equivalent to .-nJ. The command R is equivalent to 1R; -R is equivalent to -1R.

3.5.4 L Command

The L command is used to move the buffer pointer over entire lines. The use of the L command with various arguments is shown in Table 3-3.

Table 3-3
L Commands

Command	Argument	Function
L	1 assumed	Advances the pointer to the beginning of the line following the current line.
nL	$n > 0$	Advances the pointer to the beginning of the nth line following the current line.
0L	0	Moves the pointer back to the beginning of the current line.
-L	-1 assumed	Moves the pointer back to the beginning of the line preceding the current line.
nL	$n < 0$	Moves the pointer back to the beginning of the nth line preceding the current line.

If the user attempts to move the buffer pointer backward beyond the position immediately prior to the first character in the buffer, or forward beyond the position immediately after the last character in the buffer with a C, R, or J command, an error message is printed, and the pointer is not moved from the position it had before the illegal command was given. With the L command no such error message results, but the pointer will be moved beyond the boundary of the buffer.

3.5.5 Examples of the Use of Buffer Pointer Positioning Commands

```
*J3L ($)($)
*
```

The J command moves the pointer to the beginning of the first line in the buffer. The 3L command then moves it to the beginning of the fourth line.

```
*ZJ-2L ($)($)
*
```

The ZJ command moves the pointer to the end of the last line in the buffer. Then the -2L command moves the pointer to the beginning of the next to last line in the buffer (assuming that the last line is terminated by a line feed).

```
*L4C ($)($)
*
```

Advance the pointer to the position following the fourth character in the next line.

`*0L2R $ $`
`*`
	The 0L command moves the pointer back to the beginning of the current line. Then the 2R command moves it back past the last two characters in the preceding line (the second of which must be a line terminator).

`*J-L $ $`	The J command moves the pointer to the beginning of the buffer, and the -L command then has no effect and therefore does not return an error message.
`*ZJC $ $`	The ZJ command moves the pointer to the end of the buffer, and the C command then causes the error message.
`?POP`	Attempt to move pointer off the page with the C command.

3.6 TEXT TYPE-OUT COMMANDS

3.6.1 T Command

Any part of the text in the editing buffer can be typed out for examination. This is accomplished by using the T commands. The text typed out depends on the position of the buffer pointer and the argument(s) given. The T commands never move the buffer pointer.

When used with a single numeric argument, T is a line-oriented type-out command; when used with a pair of numeric arguments, T is a character-oriented type-out command. The various T commands are described in Table 3-4.

3.6.2 ↑O Command

During the execution of any T command, the user can stop the terminal output by typing the special monitor control-character ↑O . The ↑O command causes TECO to finish execution of the command string omitting all further type-outs. The effect of the ↑O command does not carry over to the next command string. (This command may only be typed as a control character. The combination ↑O (uparrow, O) does not have the same effect.) Occasionally the asterisk output by TECO when a command is finished is also suppressed by ↑O . If this occurs, the user can type ↑U . TECO will respond with an asterisk if it is waiting for a command.

Table 3-4
T Commands

Command	Argument	Function
T	1 assumed	Types out everything from the buffer pointer through the next line terminator. If the pointer is at the beginning of a line, T causes the entire line to be typed out. If the pointer is in the middle of a line, T causes that portion of the line following the pointer to be typed out.

Table 3-4 (Cont)
T Commands

Command	Argument	Function
nT	n >0	Types out everything from the buffer pointer through the nth line terminator following it. If the pointer is at the beginning of a line, this command types out the next n lines (including the current line).
0T	0	Types out everything from the beginning of the current line up to the pointer. This command is especially useful for determining the position of the buffer pointer.
-T	-1 assumed	Types out everything in the line preceding the current line, plus everything in the current line up to the pointer.
nT	n < 0	Types out everything in the n lines preceding the current line, plus everything in the current line up to the pointer.
m,nT	m < n	Types the m+1st through the nth characters in the buffer.
.,.+nT	n > 0	Types the n characters immediately following the buffer pointer.
.-n,.T	n >0	Types the n characters immediately preceding the buffer pointer.
HT	H = B,Z	Types out the entire contents of the buffer.

3.6.3 ↑L Command

If a form feed character, (↑L) or ↑L, is included in a command string as a command, it causes a form feed to be printed on the terminal when TECO reaches that point in execution of the command string. This feature is useful for obtaining a clean printout of the text in the buffer.

3.6.4 nET Command

In normal typeout mode, most control characters print in the up-arrow form and altmodes print as dollar signs. For the benefit of users with special terminal equipment, this feature can be suppressed. The command 1ET (any nonzero argument has the same effect as 1) changes the typeout commands so that every ASCII character is delivered to the typeout device literally, i.e., with its own octal mode. This is called literal type-out mode.

When TECO is in literal type-out mode, it can be restored to normal type-out mode, i.e., with substitutions for control characters and altmodes, by using the command 0ET.

The ET command (with no argument) returns the value (0 or 1) of the current setting of the type-out mode switch. See Section 3.16 for an explanation of this command.

3.6.5 Case Flagging On Type-out

TECO has three text type-out case-flagging modes: (1) lower case flagging, (2) upper case flagging, and (3) no case flagging. In lower case flagging mode, all characters in the range octal 140 to 177. are preceded by ' (apostrophe) when typed out. In upper case flagging mode characters in the range octal 100 to 137 are flagged with a preceding '. TECO is initially set for lower case flagging.

The case flagging mode may be set as follows:

nEU (n >0)	sets the typeout mode to flag upper case characters,
0EU	sets the mode to lower case flagging (standard),
nEU (n < 0)	sets the mode to no flagging,
EU	(without argument) returns the value of the current case flagging mode.

If TTY LC is on (i.e., the user's terminal handles lower case) or if the ET flag is on, no case flagging ever occurs regardless of the EU setting.

3.6.6 Examples of the User Text Typeout Commands

The following examples assume the buffer contains the text shown at the right, with the buffer pointer positioned between the M and the N

```
ABCDE↵ ↓
FGHIJ↵ ↓
KLMNO↵ ↓
  ^
PQRST↵ ↓
UVWXY↵ ↓
Z↵ ↓
```

Examples:

```
*T$$
NO
*
*3T $ $
NO
PQRST
UVWXY
*
*0T $ $
KLM*
```

Note that no carriage return-line feed exists between the beginning of the line the pointer is on and the pointer itself, therefore, none are typed. The second asterisk indicates that TECO is ready for the next command.

```
*0TT $$
KLMNO
*
```

This pair of commands causes the entire current line to be typed out without moving the pointer.

```
*-2T $$
ABCDE
FGHIJ
KLM*
```

```
*.,.+6T $$
NO
PQ*
```

The six characters typed are NO↲ ↓PQ.

```
*.-2,.T $$
LM*
```

```
*0LT $$
KLMNO
*
```

This pair of commands types out the entire current line and leaves the pointer at the beginning of this line.

```
*HT $$
ABCDE
FG ↑O
*
```

The user requests type-out of the whole buffer, but stops it with a ↑O immediately after the G is typed.

```
*↑LHT↑L $$


ABCDE
FGHIJ
KLMNO
PQRST
UVWXY
Z

*
```

This command string causes the entire contents of the buffer to be typed out, with a form feed printed before and after the text is printed.

```
*0ETHT1ETHT$ $
X↑AY$Z
XYZ
*
```

If the buffer contains the text X ↑A Y $ Z ↲↓, this command string causes it to be typed out in both normal and literal modes, as shown. In the first line typed out, the control-A and altmode are typed in normal mode as up-arrow, A and dollar sign. In the second line, typed in literal mode, ↑A and $ do not appear because they are delivered to the console device in their true values, which are nonprinting characters on most terminals.

```
*T $$
TECO M'A'N'U'A'L
```

The appearance of apostrophes in the typed text indicates that "anual" is lower case.

```
*1EUT $$
'T'E'C'O 'MANUAL
```

1EU changes TECO so that upper case characters are flagged.

```
*-1EUT $$
TECO MANUAL
*
```

-1EU stops case flagging.

3.7 DELETION COMMANDS

The K and D commands are used to delete characters from the editing buffer. The K command used with a single numeric argument is a line-oriented deletion command. The D command and the K command used with a pair of numeric arguments are character-oriented deletion commands.

3.7.1 K Command

The various K commands are described in Table 3-5.

Table 3-5
K Commands

Command	Argument	Function
K	1 assumed	Deletes everything from the buffer pointer through the next line terminator. If the pointer is at the beginning of a line, the K command causes the entire line to be deleted. If the pointer is in the middle of a line, the K command deletes only the portion of the line following the pointer (including the line terminator).
nK	n >0	Deletes everything from the buffer pointer through the nth line terminator following it.
0K	0	Deletes everything from the pointer back to the beginning of the current line.
-K	-1 assumed	Deletes everything from the pointer back to the beginning of the line preceding the current line.
nK	n < 0	Deletes everything from the pointer back to the beginning of the nth line preceding the current line.
m,nK	m<n	Deletes the m+1st through the nth characters in the buffer and positions the pointer at the point of deletion (that is, the pointer is set equal to m).

3.7.2 D Command

Using the D command, characters can be deleted individually and in short strings. The nD command, where n ≥0, deletes the n characters immediately following the buffer pointer. If the argument n is omitted, n = 1 is assumed. The command nD, where n <0, deletes the n characters immediately preceding the pointer; -D is equivalent to -1D.

At the conclusion of any K or D command, the buffer pointer is positioned between the characters that preceded and followed the deletion.

3.7.3 Examples of the Use of Deletion Commands

The following examples assume that the buffer contains the text shown at the right; the buffer pointer is positioned between the M and the N.

```
ABCDE↲↓
FGHIJ↲↓
KLM^NO↲↓
PQRST↲↓
UVWXY↲↓
Z↲↓
```

Examples:

*6D ($)($) *	Deletes NO↲↓PQ, changing the third and fourth lines to KLMRST↲↓.
*-D ($)($) *	Deletes M.
*-5D ($)($) *	Deletes↲↓KLM, changing the second and third lines to FGHIJNO↲↓.
*-2D2D ($)($) *	Deletes LMNO, changing the third line to K↲↓.
*HK ($)($) *	Deletes everything in the buffer, but does not delete the form feed marking the end of the page (if there is one).
*0,.K ($)($) *	Deletes everything from A through M.
*.,ZK ($)($) *	Deletes everything from N through Z↲↓.
*K ($)($) *	Deletes NO↲↓ changing the third and fourth lines to KLMPQRST↲↓.
*0LK ($)($) *	Deletes the entire third line.
*L3K ($)($) *	Deletes the last three lines (everything from P through Z↲↓).
*KD ($)($) *	Deletes NO↲↓P, changing the third and fourth lines to KLMQRST↲↓.
*0K ($)($) *	Deletes KLM.
*-K ($)($) *	Deletes FGHIJ↲↓KLM.

3.8 INSERTION COMMANDS

The insertion commands are used to insert characters into the editing buffer from the user's terminal.

3.8.1 I Command

The basic text insertion command is the I command used with the desired text as its argument. The text argument is terminated by an altmode. The general form is

```
*Itext ($)
```

This command inserts the ASCII text string, "text", into the editing buffer just ahead of the buffer pointer. After the insertion, the buffer pointer is positioned immediately after the last inserted character. The altmode terminating the text argument is not inserted. The text to be inserted may contain any character except the special characters (see Table 2-1), but control characters must be treated specially (see Section 3.8.8).

3.8.2 Tab Command

The tab command is equivalent to the I command, except that the tab command causes the tab itself as well as all the following text up to the altmode to be inserted. In other words, if the first character of a text string to be inserted by an I command is a tab, the I may be omitted. The general form of the tab command is

```
*→|text ($)
```

3.8.3 @I Command

The @I command is slightly more powerful than the I command. This command enables the user to insert single (but not double) altmode characters in addition to the characters that can be inserted with the I command. (To insert a double altmode, the second altmode must be preceded by a (↑R) .) The @I command is useful for inserting TECO command strings into the editing buffer. The general form is

```
*@I/text/
```

In this form, "text" is the text string to be inserted. The text argument must be immediately delimited, both before and after by any single character which is not itself a part of the text to be inserted. In this example, the delimiting character is the slash character. Altmode is not required to terminate the text string; the second occurrence of the delimiting character terminates the text string. The text is inserted immediately preceding the buffer pointer, as it is with the I command. The delimiting character is not inserted.

3.8.4 nI($)Command

Any ASCII character can be inserted into the buffer using the nI($) command. This includes all characters that the I and @I commands cannot insert. However, the nI command inserts only one character at a time. The command nI($) inserts the character with the ASCII value n (decimal) into the buffer immediately preceding the pointer.

Unless the EO value has been set to 1, the nI command must be followed by an altmode (refer to Paragraph 3.17 for a description of the EO value).

3.8.5 n\Command

The n\command is used to insert the ASCII representation of a decimal number n into the buffer. For example, 349\inserts the ASCII characters 3, 4 and 9 into the buffer immediately preceding the pointer. Note that n does not have to be a number typed in by the user. It can be a value returned by some other TECO command. Note that the n\command always inserts the decimal representation of n.

3.8.6 Examples of the Use of Insertion Commands

The following examples assume that the buffer contains ABCD∧EF↵ ↓ with the pointer positioned between D and E.

```
*IXYZ $ $                    Produces ABCDXYZ∧EF↵ ↓
*

*I↵                          Produces ABCD↵ ↓
$ $                                   ∧EF↵ ↓
*

*I↓                          Produces ABCD↓
 $ $                                   ∧EF↵ ↓
*

*3RI␣$ 4CI␣$ $               Produces A␣BCDE␣F↵ ↓
*

*→|XYZ $ $                   Produces ABCD →|XYZ∧EF↵ ↓
*
*@I#IA $ SA $ PW# $ $        Produces ABCDIA $ SA $ PW∧EF↵ ↓
*↑O33I $ $                   Produces ABCD$∧EF↵ ↓
*10I $ 10I $ $               Produces ABCD↓
*                                     ↓
                                     ∧EF↵ ↓
*Z\ $ $                      Produces ABCD8∧EF↵ ↓ because Z has
                             the value 8.

*Z\Z\Z\ $ $                  Produces ABCD8910∧EF↵ ↓ because Z
*                            successively returns the values 8, 9, and 10.

*I FORM                      This command is used to separate the page
*$ $                         in the editing buffer into two pages. Both
                             pages, however, remain in the editing buffer.

*12I $ $                     This is equivalent to the command in preceding
*                            example. It is convenient because it avoids
                             the form feed echo.
```

```
*JILINE ONE↵
LINE TWO↵
LINE THREE↵
$$
*
```

This example shows insertion of several lines of text at the beginning of the buffer. Note that line feeds are inserted automatically as the user types the carriage returns.

```
*KI↵
$$
*
```

This command string is used to delete the tail of a line without rei ioving the carriage return-line feed at the end of the line. If the buffer contains

```
AB∧CD↵↓
EFGH↵↓
```

this command produces

```
AB↵↓
∧EFGH↵↓
```

```
*I↵
RO
$$
*
```

This is used to insert a carriage return without a line feed following it. The single rubout deletes the line feed but not the carriage return. (See Section 5.1 for an explanation of rubout.)

```
*@I%TEXT $ x RO $ %$$
*
```

This is a convenient method for inserting multiple altmodes when using the @I command. The sequence x RO , where "x" is any character except altmode, is typed between the successive altmodes.

```
*↑O777\$$
```

This is used to insert the ASCII characters 511 at the current pointer position.

3.8.7 Case Control with Insert Commands

With the I, @I, and tab insert commands TECO ordinarily inserts text in the same case in which it appears in the command string. The user may, however, alter the case of text being inserted by use of the special case control commands described in this section.

3.8.7.1 Alphabetic Case Control - The features described in this section provide the method by which alphabetic characters in the upper case range can be converted to the equivalent characters in the lower case range, and vice-versa. Alphabetic case conversion is done by use of two control-character commands,

↑V is used for translation to lower case,

↑W is used for translation to upper case.

These two commands may be used within insert text arguments to cause case conversion on a temporary basis for that text argument, or as independent commands to cause case conversion in all insert and search text arguments.

Note that ↑V and ↑W affect only alphabetic characters. They have no effect on non-alphabetic characters.

(1) ↑V ↑V and ↑W ↑W used within text arguments.

When used inside an insert text argument, two successive ↑V or ↑W commands cause translation, to the specified case, of all following alphabetic characters in that text argument.

Example:

```
*IF ↑V↑V OR USERS OF ↑W ↑W TECO.$$
```

The above command inserts "For users of TECO." with the initial "F" and "TECO" capitalized, and all the other letters in lower case.

(2) Single ↑V and ↑W used within text arguments.

When used inside an insert text argument, a single ↑V or ↑W command causes translation of the next single character (if it is alphabetic) to the specified case. The single ↑V or ↑W in a text argument takes precedence over the case conversion mode defined by double ↑V or ↑W commands.

Example:

```
*I ↑V↑V USER ↑W PROGRAM$$
```

The above command causes the string "user Program" with the "P" in upper case, and all the other letters in lower case to be inserted.

(3) Independent ↑V and ↑W Commands.

As explained above, when ↑V and ↑W commands are used inside a text argument, they affect only that particular text string. When used as independent commands, however, ↑V and ↑W set TECO to a prevailing case conversion mode that affects all insert and search text arguments (except as specified by ↑V and ↑W commands within the text arguments). The independent command ↑V or ↑V (or n ↑V, where n does not equal 0) sets the prevailing case conversion mode so that all upper case alphabetic characters in insert and search text arguments are translated to lower case, except where ↑W commands within individual text arguments override the independent ↑V.

Example:

```
*↑V$$
*I↑W FOR USERS OF ↑W ↑W TECO.$$
*IEXAMPLE $$
```

The above commands cause "For users of TECO." and "example" to be inserted with all letters lower case except the "F" and "TECO". Likewise, the independent command ↑W or ↑W (or n ↑W, where n does not equal 0) sets the prevailing case conversion mode so that all lower case alphabetic characters in insert and search text arguments are translated to upper case, except where ↑V commands within individual text arguments override the independent ↑W .

The independent ↑W command has the use explained above, obviously, only when the user TTY has lower case capability and TTY LC is on. Otherwise the ↑W command serves merely to turn off the ↑V command.

(4) 0 ↑V and 0 ↑W

The independent 0 ↑V and 0 ↑W commands both have the same effect, namely, to restore TECO to the default condition where neither case of alphabetic characters are translated to the opposite case, except by ↑V and ↑W commands within text arguments.

TECO is initially set for no prevailing case conversion.

Note that the prevailing case conversion mode can have one, and only one, setting at any one time. The possible settings are:

↑V	convert upper case to lower case
↑W	convert lower case to upper case
0 ↑V or 0 ↑W	no prevailing conversion

When any of these prevailing modes is put into effect, it cancels any of the others that were in effect.

The order of precedence of the case conversion commands is as follows:

Highest:	single ↑V and ↑W inside text
Next:	double ↑V and ↑W inside text
Lowest:	independent ↑V and ↑W

NOTE

If the EO value has been set to 1, ↑W and ↑V have no special effect when used inside text arguments (refer to Paragraph 3.17 for a description of the EO value).

3.8.7.2 Special "Lower Case" Characters - When used <u>inside</u> an insert text argument, the control command ↑↑ causes the immediately following character (if it is one of the special characters @, [, \,], ↑, or ←) to be converted to the equivalent character in the lower case ASCII range (i.e., octal 140 or octal 173-177). That is,

↑↑@ becomes	`	ASCII 140
↑↑ [becomes	{	ASCII 173
↑↑ \becomes	|	ASCII 174
↑↑] becomes	}	ASCII 175
↑↑ ↑ becomes	~	ASCII 176
↑↑ ← becomes	RO	ASCII 177

↑↑ has no special effect within text arguments if the EO value has been set to 1.

Examples:

```
*↑VI ↑W EXAMPLES FOR THE                 Inserts "Examples for the
↑W ↑W TECO M ↑W ↑W ANUAL.                TECO Manual.
$ $                                      EXAMPLE 1.
*0↑VIEXAMPLE 1.                          nI Command.".
↑V NI C↑V ↑V OMMAND.
$ $
*

*I ↑↑ [$ $                               Inserts a right brace ({)
```

3:8.8 Inserting Control Characters

As of version 22 of TECO all of the control characters ↑A - ↑G , ↑N - ↑Z , and ↑\ , ↑] , ↑↑ , and ↑← have been reserved as inside-text-commands (some as yet undefined). In order to insert these characters, the user must employ either the ↑R or ↑T command.

↑R when used inside an insert text argument causes the next single character to be interpreted as text rather than as a command, and accordingly to be inserted in the buffer. This applies to all control characters including ↑R itself. It also applies to Altmode. (It does not, however, apply to ↑C , ↑O , ↑U , or RUBOUT.)

↑T when used inside an insert text argument causes all succeeding instances of the above mentioned control characters except ↑R and ↑T itself to be interpreted as text rather than as commands. ↑T does not affect altmodes. A second instance of ↑T in the same text argument nullifies the effect of the first.

If the EO value has been set to 1, ↑R and ↑T have no special effect when used inside text arguments, and all control characters can be inserted with no special treatment (refer to Paragraph 3.17 for a description of the EO value).

NOTE

The clever way to create a TECO macro is simply to type the macro as a long command string just as if it were to be executed immediately, but instead of typing $ $ at the end, type ↑G ↑G. Then type *i to place the command string in Q-register i. (This stores the macro, ready for execution, in Q-register i. (Refer to Paragraph 3.14.3 for the description of the *i command.)

Examples:

```
*I ↑R ↑A TEXT ↑R ↑A $ $                  Inserts the text " ↑A TEXT ↑A ".
*
```

```
*INSTRING ↑R $ $ $                    Inserts "NSTRING $".
*
*I ↑T ↑A SEARCH ↑A                    Inserts " ↑A SEARCH ↑A
NSTRING ↑R $                                  NSTRING $
I ↑V ↑V TEXT ↑R $ ↑T                          I ↑V ↑V TEXT $
!E ↑V ↑V XAMPLE! $ $                          !Example!".
*
```

3.9 OUTPUT COMMANDS

Output commands are used to transfer data from the editing buffer to the output file.

3.9.1 PW Command

The PW command is the basic output command. It does nothing but output. Depending on the argument used with it, the PW command outputs all or any part of the data in the editing buffer. It does not, however, delete any data from the buffer, and it never moves the buffer pointer.

The PW command outputs the entire contents of the buffer and always appends a form feed to it. The nPW command (n >0) outputs n copies of the text in the buffer, appending a form feed to each copy.

3.9.2 P Command

The P command is a combination command; when used with a single numeric argument (or no argument), the P command does both output and input. The various functions of the P command are described in Table 3-6.

Note that the P command (with a single argument) always clears the editing buffer before it inputs the next page, and it leaves the pointer at the beginning of the new page. If a P command is executed after the end of the input file has already been reached or when there is no input file, the buffer is simply cleared. No data is read in.

Unlike the PW command, the P command does not always cause a form feed to be output at the end of the data output from the editing buffer. The P command outputs a form feed at the end of the data only if a form feed was encountered to terminate the last input command.

Table 3-6
P Commands

Command	Argument	Function
P	1 assumed	Similar to PWY. Outputs the entire contents of the buffer, then clears the buffer and reads in the next page of input. The buffer pointer is left at the beginning of the page that is read in. If there is no input file, or no more data in the input file, the buffer is left cleared. A form feed character is appended to the end of the data that is output only if the last input command was terminated by a form feed.
nP	n >0	Executes the P command n times. This command can be used to skip over several pages of text when no editing is required. The nP command causes the n pages of the input file, starting with the page currently in the editing buffer, to be output, and then the nth page after the current page to be yanked in.
m,nP	m<n	When used with a pair of numeric arguments, the P command does output only; it does not clear any data from the buffer, it does not input any more data, and it does not move the buffer pointer. Also, the m,nP command never causes a form feed to be appended to output[1]. The only action of m,nP is to output the m+1st through the nth characters in the buffer. (m,nP and m,nPW are equivalent.)
HP	H = B,Z	Outputs the entire contents of the buffer without appending a form feed to it; the buffer is not cleared, and no new data is read in. (HP and HPW are equivalent.)

[1] However, if a form feed character has been inserted in the buffer between the mth and nth characters, it will be output.

The PW command does not clear the buffer and does not move the buffer pointer. The same is true of a P command used with two arguments.

Note also that when a PW command is used, a form feed character is always automatically sent to the output file immediately following the data from the buffer. (Recall that when the page was read into the buffer, the form feed character that terminated it, if any, was discarded and not read into the buffer.) The form feed character is appended to the outgoing data regardless of whether or not a form feed character was encountered when the data was read in, i.e., regardless of the setting of the form feed flag. This is not true of the P command.

NOTE

If the EO value has been set to 1, the P command behaves like the PW command with regard to form feeds.

When a P or PW command is used with a double numeric argument (including an H argument), a form feed character is never appended to the output data. This is true regardless of whether or not a form feed character was encountered when the data was read in.

NOTE

> The discussion in this section does not apply to the form feed characters that the user has inserted into the editing buffer using 12I $ or I FORM $ commands. Form feed characters in the buffer are output exactly as other characters in the buffer.

If the editing buffer is empty when a P or PW command is executed, no output of any kind takes place. No form feed character is output. If the user wants to create a blank page, an example of the procedure is shown below.

As shown in the discussion above, the nP command can be used to skip over several pages to get to the next page where editing is required. The nP command can also be used with a very large argument, e.g., 10000, in order to skip to the end of the input file without doing any more editing. The N and EX commands are other commands which can be used for this purpose.

3.9.3 EF Command

The EF command is the output file closing command. The EF command, or an equivalent command, must be used to close the output file after all output to it is complete. The EF command is normally used after the P command which outputs the last page of a file. The special exit commands EX and EG (see Section 3.10) automatically cause an EF to be executed. Also, a new EW command causes an EF to be executed on the previous output file, if any, before opening the new output file. Note that if an EF command is executed in the middle of the file, all succeeding pages of that file are lost.

3.9.4 Examples of the Use of Output Commands

```
*PT $ $
FIRST LINE OF NEXT PAGE
```

Output the current page, clear the buffer, read in the next page, then type out the first line of the new page.

```
*PEF $ $
*
```

Output the current page to the output file, and then close the output file. This command string is used to close a file (after writing the last page) when it is not desirable to exit from TECO.

```
*PWEF $ $
```

Equivalent to the preceding example, except that the buffer is not altered.

```
*.,ZP0,.P $ $
*
```

This command string outputs the entire contents of the buffer, but it rearranges the data as it is output. The part of the page that follows the buffer pointer is output first by the .,ZP command. Then that part of the data which precedes the pointer is output by the 0,.P command. No form feed character is appended to either section of the output.

```
*.,ZP12I $ 0,.P $ $
*
```

This performs the same function as the preceding command string except that it does append a form feed character to that part of the page that is output last.

```
*HK12I $ HP $ $
*
```

This command string produces a single blank page.

```
*HK12I $ PW $ $
*
```

This produces two successive blank pages.

```
*8P $ $
*
```

If page 6 of a file is in the editing buffer, this command causes pages 6 through 13 of the file to be output one after the other, and then reads in page 14.

```
*300PW$ $
*
```

This outputs 300 copies of the current page.

```
*PWJKIJ.DOE $ PW $ $
```

This outputs the current buffer, the modifies the first line and outputs the buffer again.

```
.MAKE FILE
*Ipage of text $ $
*PI2nd page of text $ $
*PIlast page of text $ EX $ $
```

This is the usual method for creating a text file.

3.10 EXIT COMMANDS

Exit commands are used to terminate a TECO job and return to the monitor. There are four exit commands: EX, EG, ↑Z , and ↑C .

3.10.1 EX Command

The EX command is used to bring an editing job to a satisfactory conclusion with a minimum of effort. Its use is shown in the example below.

The user is editing a 30-page file and that the last actual change to the file is made on page 10. At this point the user gives the command:

```
*EX $ $
EXIT
↑C
.
```

In this case, the action performed by TECO is equivalent to the command string 21PEF, with an automatic exit to the monitor at the end. Thus, the action of TECO is (1) to rapidly move all the rest of the input file, including the page currently in the buffer, on to the output file; (2) to close the output file; and (3) to return control to the monitor.

The EX command is the easiest method of finishing an editing job, with the latter part of the input file being properly output and the output file closed.

The EX command performs both input and output functions.

The EX command causes a form feed character to be output after the output of the buffer, only if a form feed was encountered when that buffer of text was read in. In this way, the EX command maintains existing page sizes.

3.10.2 EG Command

The EG command first performs exactly the same functions as the EX command, and then causes the last compile-class command (COMPILE, EXECUTE, LOAD, or DEBUG) attempted before TECO was called, to be re-executed (with the same arguments). Generally, the EG command is used only to exit from an editing job that was initialized by an EB command or a TECO filnam.ext command.

As an example, suppose the user gives the command

```
.COMPILE PLOT.F4↵
```

to request compilation of a FORTRAN source program, but the compiler encounters errors in the code. The user then calls TECO to correct these errors with the command:

```
.TECO PLOT.F4↵
*
```

When all the errors are edited, the user exits from TECO with the command

```
*EG $$
```

This command causes (1) the rest of the file PLOT.F4 to be output and closed, and (2) the command COMPILE PLOT.F4 to be re-executed automatically.

3.10.3 ↑Z and ↑C Commands

The ↑Z and ↑C commands do not perform any input or output. They are used strictly for exiting to the monitor.

The command ↑Z (or ↑Z) is the simple exit command that can be entered into command strings. It allows any I/O commands that have already been given to be completed, then closes the output file, and then returns the user to the monitor.

Example:

```
*PWEF ↑Z $ $
EXIT
↑C
.
```

The ↑Z is executed as a regular command in the command string when its turn comes.

NOTE

If the EO value has been set to 1 (refer to Paragraph 3.17.3), a single ↑G is equivalent to ↑Z .

The ↑C command is a monitor command that is used to immediately exit to the monitor. The ↑C command can be typed at any time, while typing a command string or while a command string is being executed, and it will override everything else. It cannot be entered in the up-arrow, C form. If there are any input/output functions in progress when ↑C is typed, a single ↑C will allow them to be completed before exiting to the monitor. Double ↑C (↑C ↑C) interrupts everything, even I/O in progress, and exits to the monitor immediately. The ↑C command does not cause the output file to be closed.

Both ↑Z and ↑C are abortive exit commands. However, when they are used, it is possible to return to the TECO job provided no other program has been called into core over the TECO job. Simple monitor commands such as ASSIGN, or PJOB, can be executed without damaging the TECO job.

After an exit to monitor level, even if the exit was caused not by a user ↑C , or ↑Z , but instead by some problem detected by the monitor itself, the user can return to his TECO job by using either the CONTINUE or the REENTER command.

The command CONT causes TECO to begin operations exactly where it left off. Even I/O can be interrupted and then continued.

Example:

```
*ERPTR: $ EWLPT: $ Y3P $ $
DEVICE LPT OK?
.CONT↵
*
```

Here the monitor causes an exit to monitor level because of a device problem. After the user corrects the problem, he continues the job and the current command string executes to completion.

REENTER causes the TECO job to be reentered with the contents of the editing buffer (when the exit occurred) intact. After reentry by a REENTER, TECO reinitializes itself for a new command string. Any previous commands still unexecuted at the time of the exit are lost. If a command string was being executed when the exit occurred, the part of the string that was not executed before the exit will not be executed after the REENTER command. The user must determine how much of the command string was executed. If I/O is interrupted, some portion of the input or output files is frequently either lost or duplicated.

Examples:

```
*ICOMME ↑C
.DEASSIGN LPT ↵
.DAYTIME ↵
14-APR-70     10:34
```

Before finishing a command string the user exits to perform a monitor command.

```
.REE↵
*ICOMMENTS $ $
*
```

He then reenters TECO. The command string must be retyped, but the buffer is still intact.

```
*<SFOO $ 0L > $ $
 ↑C ↑C
.REE↵
*
```

This is an infinite loop (if FOO is in the buffer). ↑C ↑C stops execution and returns the user to the monitor. REE restarts TECO with the editing buffer intact and the command buffer empty.

```
*50P $ $
 ↑C
 ↑C
.REE↵
*
```

This is an example of what should not be done. Interrupting execution of an I/O command does not permit reentry. In this case, some of the output file will almost certainly be duplicated.

The contents of any Q-registers (refer to Paragraph 2.8) remain intact after a ↑C , CONT or ↑C , REENTER command sequence.

3.11 SEARCH COMMANDS

In many cases the simplest way to reposition the buffer pointer is by using a character string search. A search command causes TECO to scan through the text until a specified string of characters is found, and then to position the pointer at the end of this string.

The string of characters to be searched for is supplied as a text argument with the search command. The search string can be from 1 to 36 character positions in length or up to 80 characters including all control commands.

If an exact match for the search string is found in the text, the buffer pointer is positioned immediately after the last character in this match. If the string is not found, TECO positions the pointer at the beginning of the buffer and notifies the user of the failure. The failure notice may take one of two forms, depending on the type of search command used. For further explanation see the paragraph below.

All searches begin at the current position of the buffer pointer.

If no text argument is provided with a search command, e.g., S $ $ or @N//, the search is executed using the last previous search command argument.

3.11.1 S Command

The S Command is used to search for a character string within the current editing buffer. If the string is not found between the current buffer pointer position and the end of the buffer, the search fails. After an unsuccessful S search, the buffer pointer is reset to the beginning of the buffer, and, unless the : modifier (explained below) was used or the search is within an iteration (see Section 3.12), an error message is printed.

The general form of the S command is

```
*Sstring $
```

For the standard S command, the search string is provided as a normal alphanumeric argument following the S and terminated by an altmode. "string" can contain any character except the special characters listed in Table 2-1.

The S command may be used with a single numeric argument. The command nS causes a search for the nth occurrence of the specified search string. When n is omitted, n=1 is assumed. n must be greater than 0.

3.11.2 FS Command

The FS command is used to search for a character string within the current editing buffer (function of the S command) and replace it with another string. If the string to be replaced is not found after the current buffer pointer position and before the end of the buffer, the search fails and no replacement is made.

The general form of the FS command is

```
*FSstring1 $ string2 $
```

where string 1 is the string to be deleted and string2 is the string to be inserted in its place. If string 2 is omitted, string 1 is deleted without any string replacing it. However, even when string2 is omitted, its terminating altmode must be present as shown in the form:

```
*FSstring1 $ $
```

3.11.3 N Command

The N command combines the S command with input/output functions. The N command is used to search for a character string in a page of the input file which may not yet have been read into the buffer. The N command has the same form as the S command.

The N command functions exactly like the S command except that an N search does not terminate at the end of the page currently in the buffer. If no match for the search string is found between the current buffer pointer position and the end of the buffer, the current page is output, the buffer is cleared, the next page is read in, and the search starts over at the beginning of the new page. This process continues until a match is found or the input file is exhausted.

If an N search fails, the entire input file has been passed through the buffer and delivered to the output file, and the buffer cleared. The output file is not closed. Unless the : modifier was used or the search is within an iteration, an error message is typed to notify the user that the search has failed.

An N search will not detect a match when the matching characters are split across two buffer loads.

The output function of the N command is exactly like the P command and the EX command. If a form feed character was encountered when a given page was read in, a form feed character is appended to that page when it is output; otherwise, no form feed character is output.

The N command can be used with a single numeric argument. The command nN causes a search for the nth occurrence of the specified search string. When n is omitted, n=1 is assumed. n must be greater than 0.

3.11.4 FN Command

The FN command is used to search for a character string in a page of the input file which may not yet have been read into the buffer (function of the N command) and to replace it with another string. The FN command operates like the N command when searching for the string. If the search fails, no replacement occurs.

The general form of the FN command is

```
*FNstring1 ($) string2 ($)
```

where string1 is the string to be deleted and string2 is the string to be inserted in its place. If string2 is omitted, string1 is deleted without any string replacing it. However, even when string2 is omitted, its terminating altmode must be present as shown in the form

```
*FNstring1 ($)($)
```

3.11.5 Backarrow Command

The backarrow command is identical to the N command except that a backarrow search generates no output. Generally, where the N command executes a P, the backarrow command executes a Y. The backarrow search is used for examination functions and for discarding parts of a file. The general form of the backarrow command is

```
*←string ($)($)
```

The backarrow command can also be used with a single numeric argument. The command n←causes a search for the nth occurrence of the specified search string. When n is omitted, n=1 is assumed. n must be greater than 0.

3.11.6 Search Command Modifiers

3.11.6.1 @ Modifier – There are two search command modifiers. The @ modifier is used to alter the method which TECO reads the search command's text argument from the command string. The general form of a @ search command is the same for S, FS, N, FN, and backarrow. It is

```
*@nS/string/
```

The @ modifier is placed before the S, FS, N, FN, or backarrow, and before the numeric argument, if any. When the @ modifier is used, the search string argument is delimited, not by the search command and an altmode, but by the first character typed after the search command and the next recurrence of this character. In the example above, the delimiting character is a slash. The delimiting character may be any character except a character that appears in the search string itself. With the @ modifier, single (but not double) altmodes can be used in the search string. The @ modifier can be used in an FS or FN command to separate the strings with a delimiting character other than altmode. This is useful in cases where a double altmode cannot terminate the command. A double altmode terminates an FS or FN command when the replacement string is omitted to allow deletion of the string for which the search is made. Use of the @ search commands is similar to the use of the @I insert command (refer to Paragraph 3.8.3).

3.11.6.2 Colon Modifier – The colon modifier is used to alter the execution of a search command in the event the search fails. Without the colon modifier, a search that fails causes an error message to be printed; if the colon modifier is used, no error message is printed. Instead, every colon search command executed returns a numeric value that can be printed out, stored in a Q-register, or tested by a conditional branch. A colon search command returns the value -1 if the search is successful, and the value 0 if the search fails.

The general form of a colon search command is the same for S, FS, N, FN, and backarrow searches:

```
*:nSstring $
```

The colon precedes the search command letter and its numeric argument, if any. Both the colon and @ modifiers may be used on a search command, in either order.

The concept of a command returning a value is explained in Section 2.7.3. Just as the Z command takes on a value that may be used as a numeric argument, so also the command :Sstring $ takes on a value of 0 or -1 after it is executed. If this is the last command in a command string, or if the command following it does not take a numeric argument, the value returned by the colon search is discarded. Hence, a colon search should be followed by a command that takes a numeric argument.

The colon search commands reposition the buffer pointer in the same manner as other search commands, regardless of whether or not the returned value is used.

The colon searches are used primarily in programmed editing and are usually followed by a conditional command. Examples of the uses of colon searches are given in Sections 3.13 and 3.14.

3.11.7 Automatic Typeout After Searches

The ES command allows the user to specify automatic typeout of the line where a successful search has terminated. The search cannot be in an iteration, nor can the search command be preceded by a colon. When the FS or FN command is used, the typeout occurs after the insertion has taken place. The user can also specify in the ES command that either a line feed or a character be inserted into the typeout to indicate the position of the pointer. Unless the ES value is set, the default is that no automatic typeout after searches will be performed.

The user can set the ES value in the following manner:

0ES — Restore TECO to the default of no automatic typeout.

-1ES — Set the ES value to cause automatic typeout of a line on which a successful search has terminated.

nES(n >0) — Set the ES value to n. If n is in the range 1 through 31, a single line feed character is included in the typeout at the position of the pointer. If n is 32 or greater, the character with the ASCII value specified by n is included in the typeout at the position of the pointer.

ES — Examine the setting of the ES flag.

3.11.8 Case Control in Searches

When searching for alphabetic characters TECO will normally accept either upper or lower case characters as a match. This is called "either-case mode". TECO may, however, be forced to execute any or all searches in "exact mode". In exact mode TECO will accept an alphabetic character or a search match only if it has the same case as the corresponding character given by the user in the text argument.

Before the techniques for match mode control can be explained, we must first explain the various techniques for case control. Match mode control is explained in Section 3.11.8.4.

3.11.8.1 Alphabetic Case Control in Search Arguments - The case of alphabetic characters in search text argument is controlled by the same set of commands used to control case in insert text arguments.

The features described in this section provide the method by which alphabetic characters in the upper case range can be converted to the equivalent characters in the lower case range, and vice-versa.

Alphabetic case conversion is done by use of two control-character commands.

↑V is used for translation to lower case.
↑W is used for translation to upper case.

These two commands may be used within search text arguments to cause case conversion on a temporary basis for that text argument, or as independent commands to cause case conversion in all insert and search text arguments.

Note that ↑V and ↑W affect only alphabetic characters. They have no effect on non-alphabetic characters.

(1) ↑V ↑V and ↑W ↑W used within text arguments.

When used inside a search text argument, two successive ↑V or ↑W commands cause translation, to the specified case, of all following alphabetic characters in that text argument.

Example:

```
*SF ↑V ↑V OR USERS OF ↑W ↑W TECO. $ $
```

The above command searches for "For users of TECO." with the initial "F" and "TECO" capitalized, and all the other letters in lower case.

(2) Single ↑V and ↑W used within text arguments.

When used inside a search text argument, a single ↑V or ↑W command causes translation of the next single character (if it is alphabetic) to the specified case. The single ↑V or ↑W in a text argument take precedence over the case conversion mode defined by double ↑V or ↑W commands.

Example:

```
*S ↑V ↑V USER ↑W PROGRAM $ $
```

The above command causes a search for the string "user Program" with the "P" in upper case, and all the other letters in lower case.

(3) Independent ↑V and ↑W commands.

As explained above, when ↑V and ↑W commands are used inside a text argument, they affect only that particular text string. When used as independent commands, however, ↑V and ↑W set TECO to a prevailing case conversion mode that affects all insert and search text arguments (except as specified by ↑V and ↑W commands within the text arguments).

The independent command (↑V) or ↑V (or n (↑V) where n does not equal 0) sets the prevailing case conversion mode so that all upper case alphabetic characters in insert and search text arguments are translated to lower case, except where (↑W) commands within individual text arguments override the independent (↑V).

Likewise, the independent command (↑W), or ↑W (or n (↑W), where n does not equal 0) sets the prevailing case conversion mode so that all lower case alphabetic characters in insert and search text arguments are translated to upper case, except where (↑V) commands within individual text arguments override the independent (↑W) .

The independent (↑W) command has the use explained above, obviously, only when the user TTY has lower case capability and TTY LC is on. Otherwise the (↑W) command serves merely to turn off the (↑V) command.

(4) 0 (↑V) and 0 (↑W)

The independent 0 (↑V) and 0 (↑W) commands both have the same effect, namely, to restore TECO to the default condition where neither case of alphabetic characters are translated to the opposite case, except by (↑V) and (↑W) commands within text arguments.

TECO is initially set for no prevailing case conversion.

Note that the prevailing case conversion mode can have one, and only one, setting at any one time. The possible settings are:

↑V	convert upper case to lower case
↑W	convert lower case to upper case
0 ↑V or 0 ↑W	no prevailing conversion

When any of these prevailing modes is put into effect, it cancels any of the others that were in effect.

The order of precedence of the case conversion commands is as follows:

Highest:	single (↑V) and (↑W) inside text
Next:	double (↑V) and (↑W) inside text
Lowest:	independent (↑V) and (↑W)

NOTE

If the EO Value has been set to 1 (refer to Paragraph 3.17.3), (↑W) and (↑V) have no special effect when encountered inside text arguments.

3.11.8.2 Special "Lower Case" Characters - When used inside a search text argument, the control command ↑↑ causes the immediately following character (if it is one of the special characters @, [, \ ,], ↑, or ←) to be converted to the equivalent character in the lower case ASCII range (i.e., octal 140 or octal 173 to 177). ↑↑ has no special effect within text arguments if the EO value has been set to 1. Refer to Paragraph 3.8.7.2 for examples.

3.11.8.3 Control Characters in Search Arguments - As of version 22 of TECO all of the control characters ↑A - ↑G , ↑N - ↑Z , and ↑\, ↑] , ↑↑ , and ↑← have been reserved as inside-text-commands (some as yet undefined). In order to search for these characters, the user must employ either the ↑R or ↑T command.

↑R when used inside a search text argument causes the next single character to be interpreted as text rather than as a command. This applies to all control characters including ↑R itself. It also applies to altmode. (It does not, however, apply to ↑C , ↑O , ↑U , or RUBOUT.)

↑T when used inside a search text argument causes all succeeding instances of the above mentioned control characters except ↑R and ↑T itself to be interpreted as text rather than as commands. ↑T does not affect altmodes. A second instance of ↑T in the same text argument nullifies the effect of the first.

If the EO value has been set to 1, ↑R and ↑T have no special effect when used inside text arguments, and all control characters (except the special characters) can be searched for with no special treatment.

3.11.8.4 Case Match Mode Control in Searches - Unless special action is taken all searches are executed in "either-case mode". This means that regardless of the setting of the prevailing case mode by an independent ↑V or ↑W command, a search for an alphabetic character will accept either the corresponding upper or lower case character as a match.

However, if ↑V or ↑W case control commands are used within a search text argument, it is assumed that the user desires an exact mode search, and a match will be accepted only for the corresponding characters in the exact case specified by the user.

If the user desires a search to be executed partly with exact mode and partly with either-case mode, he should bracket the characters to be taken in either case with ↑\ characters. (The ↑\ character is entered by simultaneously depressing the CTRL, SHIFT, and L keys.)

For example, S ↑V ↑V ABC ↑\ DEF ↑\ $ will be successful only with strings containing lower case abc, but it will accept either upper or lower case def as a match for the last 3 characters.

NOTE

If EO=1, all searches are executed in exact mode and ↑\ has no special effect in text arguments.

The search mode can be forced to exact mode for all searches by use of the independent command n ↑X, where n does not equal 0. 0 ↑X resets the search mode to 'either' mode. ↑X without an argument returns the value of the search mode flag.

3.11.9 Special Match Control Characters

There are five special control characters that can be used in search character string arguments. These characters alter the usual character-matching process that goes on when a search is in progress. They actually reside in the search string and are interpreted by the search routine itself.

The presence of a ↑X command in a search string is a signal that this particular character position in the string is unimportant and that any character is to be accepted as a match for it. The ↑X command is a free variable in the search string. To find a match, some character must be present in the position occupied by the ↑X command; however, it does not matter what this character is.

The ↑S command in a search string is a restricted variable. Its presence indicates that any separator character is to be accepted as a match in its position. A separator character in any character except a letter, a digit, a period, a dollar sign, or a percent sign; i.e., any character except a character that is commonly used in symbols. ↑S also accepts the beginning of the editing buffer as a match.

The ↑N command is another restricted variable. It must be followed by a single character argument: ↑N x. The ↑N command signals that, in the position occupied by the ↑N and its argument, any character is to be accepted as a match except the argument.

The ↑R command is used in a search string to indicate that the character following the ↑R is to be interpreted literally rather than as a command, even if this character is one of the special match control characters. The ↑Q command has the same function as ↑R, but it is better to use ↑R because ↑Q will not allow insertion of $ as a text character while ↑R will.

The ↑E command when used with an argument in a search string indicates particular groups of characters to be accepted as a match. Depending on the argument, this command matches on the first occurrence of one of the following groups.

↑E A	any alphabetic character.
↑E D	any digit.
↑E L	any end of line character (or end of buffer character in the absence of an end of line character).

↑E S — any string of spaces and/or tabs.

↑E V — any lower case alphabetic character.

↑E W — any upper case alphabetic character.

↑E <nnn> — the ASCII character whose octal value is nnn.

↑E [a,b,c,..] — any one of the characters a,b,c,... (a,b,c,.. can be any symbols that represent single characters).

Since the five commands ↑X , ↑S , ↑N , ↑R , and ↑E are used in the middle of ASCII search strings, they cannot be entered in the up-arrow, character form allowable for some control character commands. They must be typed as a single control character.

3.11.10 Examples of the Use of Search Commands

Examples:

```
*SA→|B $ $
*
```

This causes the pointer to be positioned immediately after the B, in the first occurrence of the string A→|B after the current position of the pointer.

```
*SNIX $ $
?SRH CANNOT FIND "NIX"
*
```

The string NIX is not found between the current pointer position and the end of the buffer. The error message is typed and the pointer moved to the beginning of the buffer. The user may have typed an incorrect search string, the pointer may have been positioned somewhere in the buffer after the N, or the string NIX may not have been read into the current buffer.

```
*NDIGITAL $ $
*
```

If page 5 of the text is currently in the buffer and the string DIGITAL does not occur until page 15, this command causes pages 5 through 14 to be output and page 15 to be read in. The pointer will be set immediately after the L.

```
*NLAST LIN PG1
TST LIN PG2
$ $
?SRH CANNOT FIND "LAST LIN PG1
1ST LIN PG 2
"
*
```

If this string actually exists in the file but the two lines are not read into the same buffer load, the N search will fail.

```
*12FSOF $ FOR $ $
```

This command causes TECO to search the current buffer for the 12th occurrence of the string "OF" and replace it with the string "FOR".

```
*5←VERSION88 $ $
```

This command can be used to determine if the string VERSION88 occurs in the input file five times. If it does, the pointer is positioned immediately after the fifth occurrence, and everything in the input file, preceding the page on which the fifth occurrence is located, is discarded.

```
*-1ESSWORD $ $
60     FORMAT ('WORD')
*
```

The ES value is set to -1 to cause the line where the search ended to be typed. This makes certain that the search has actually found the right occurrence of the string. It is easy to overlook an occurrence of a string preceding the one which the user desires.

```
*5FSINTEREST $ $
```

This command causes TECO to search the current page for the fifth occurrence of the string "INTEREST" and delete it. Two $'s must be present following the string to be deleted; the first delimits the string to be searched for and the second tells TECO that there is no replacement string.

```
*NMASSACHUSETS $ $
?SRH CANNOT FIND "MASSACHUSETS
*EF $ $
*EBOUTPUT.FIL $ Y $ $
*NMASSACHUSETTS $ $
*
```

An N search should not be used where an S search would suffice, because user errors with the N command, such as the spelling error shown here, can cause considerable delay. In this example, the user's error caused him to have to pass over the entire file twice instead of just once.

```
*@3S+ $ +IEF $ $
*
```

The command @ 3S + $ + searches for the third occurrence of the altmode character following the buffer pointer. When this altmode is found, the characters EF are inserted immediately after it. The plus characters serve as the delimiters for the one-character search string $. The plus characters are not part of the search string.

```
*@FN/WRITE#/PRINT#/ $ $
```

This command causes TECO to search for the string "WRITE#" and replace it with the string "PRINT#." Each page of the text is searched until the string is found.

```
*NA ↑X B ↑S C ↑N .D ↑R ↑X $ $
*
```

Any of the following three strings of characters would serve as a match for this N search:

```
A6B-C?D ↑X
A␣B ⇥C␣D ↑X
AAB,C $ D ↑X
```

None of the following four strings would serve as a match:

```
AJB C-D3
A.B.C.D. ↑X
AABBCCD ↑X
AXB␣CAX ↑X
```

```
*1ESSFOUR $ $
FOUR
     SCORE AND SEVEN YEARS AGO
```

Because the ES value was set to 1, automatic typeout of the line occurs after the string "FOUR" was found. A line feed was inserted at the pointer position in the line to allow the user to easily locate the pointer.

```
*1ESFSI/O $ I-O $ $
I-O
   CONTROL
```

This command string causes TECO to search for the string "I/O" on the current page and replace it with the string "I-O". The line is then typed with a line feed at the position of the pointer.

3.12 ITERATION COMMANDS

3.12.1 Angle Bracket (<...>)

The user can cause a group of command to be iterated (repeatedly executed) any number of times by placing these commands within angle brackets. The left angle bracket marks the beginning of a command string loop and the right angle bracket marks the end of the loop. These command string loops can be nested in the same manner as arithmetic expressions are nested within parentheses. Loops should be nested to no more than approximately 20 levels; otherwise, pushdown list overflow may occur.

A numeric argument can be used to specify the number of times a given loop is executed. The argument is placed before the left angle bracket in the form n<...>. This causes the group of commands within the brackets to be iterated n times. In a command of the form n <...>, if the argument n is less than or equal to zero, the commands contained within the angle brackets are skipped. If no argument is given, the number of iterations is assumed to be infinite (2^{35}).

Example:

```
*J8< ⇥$ L> $ $
*
```

This command string inserts a tab at the beginning of the first eight lines in the buffer and leaves the pointer positioned at the beginning of the ninth line. The J command starts the pointer off at the beginning of the first line. The first command in the loop, ⇥$ inserts a tab. Then the next command, L, moves the pointer to the next line to prepare for the next iteration of the loop.

3.12.2 Semicolon Command

Iteration of a command string loop can be terminated before the iteration count is satisfied by using the conditional iteration exit command, semicolon. The semicolon command can be used only within angle brackets. It can be used with or without a numeric argument.

When used without a numeric argument, the semicolon command evaluates the outcome of the last search (of any kind) that was executed before the semicolon command was encountered. If this search was successful, command execution continues within the loop, as if no semicolon were present. If, however, the most recent search failed, the semicolon command causes all those commands that follow the semicolon in the loop to be skipped over, and command execution to pass on to the first command following the right angle bracket which closes the innermost loop that the semicolon is in.

NOTE

> Within a command loop, all searches are colon searches. They do not generate error messages when a failure occurs, instead they return a value of -1 if successful and 0 if unsuccessful.

The semicolon command can also be used with a numeric argument. The command n; is ignored if n<0. However, if n ≥ 0, the command n; causes command execution to exit from the loop just as the semicolon command exits from the loop when a search fails.

Examples:

```
*J<0LIJAN $ FS1969 $ 70 $;>HT $$
JAN REPORT
DEPT:

JAN 1970    SALES
            WHOLESALE:
            RETAIL:
JAN 1970    EXPENSES:
            OVERHEAD:
            ADVERTISING:
            COMMISSIONS:
JAN 1970    RETURNS:
JAN 1970    INVENTORY:

*
```

This command string inserts JAN at the beginning of the first line in the buffer and at the beginning of each line that contains the string 1970. It also changes the 69 in every occurrence of 1969 to 70. The action is as follows: The J command starts the operation at the beginning of the buffer. The first execution of the 0L does nothing. IJAN $ then inserts JAN at the beginning of the first line. Now, a search is made for 1969. When 1969 is found, FS1969 $ 70 $ changes the 69 to 70. This completes the first iteration; execution loops back to the <. 0L moves the pointer to the beginning of the line where the 1969 was found. Here JAN is inserted and then a search is begun for the next 1969. This continues until the search command fails to find another 1969. When the search fails, the pointer is moved to the beginning of the buffer. HT is the next command which is executed. (It is assumed that no line contains more than one "1969.")

```
*<S1969 $ ;0LIDEC $>$$
[6K CORE]
[7K CORE]
[8K CORE]
 ↑C ↑C
.REE↵
*
```

This command puts TECO into an infinite loop because the 0L causes the search command to keep finding the same 1969 over and over again. If left to run long enough the IDEC $ command will eventually exhaust available core and stop execution. In this example, the user has stopped the loop with ↑C ↑C , and then REEntered.

```
*Y<NEXAMPLES: $;<S↵
$; ⇥$ L >>$ $
*
```

This is an example of nested loops. The main loop searches for pages in a file that contain the heading EXAMPLES:. When this is found, execution enters the secondary loop, which inserts a tab at the beginning of all the succeeding lines on that page (i.e., after every ↵↓ on that page). When the second semicolon causes an exit from the inner loop, execution loops back to the N search. Finally, when the N search fails, execution is completed.

```
*EBfilnam.ext $ 50000<YHP>EX $ $
```

This example shows how to remove all form feeds from a file.

```
*<FSREAD $ WRITE $ ;>
```

This command causes a search of the current page for all occurrences of the string "READ" and replacement of them with the string "WRITE".

```
*<@FN/ERROR//;>
```

This command causes TECO to search all the following pages for the string "ERROR" and delete every occurrence of it. The @ construction must be used in this case because it allows the user to specify a delimiting character other than $. The delimiting character (in this case /) must be specified twice after the string; the first to end the string and the second to indicate that a replacement string is not present. If $ were used as the delimiter, a double $ would be present which would cause an erroneous result.

Only the methods described in this section should be used to exit from a loop. Specifically, the flow control commands described in Section 3.13 should not be used. Some violations of this rule may be successful, but generally they will not succeed.

Matching pairs of angle brackets defining loops within the loop may, however, occur following the semicolon.

3.13 FLOW CONTROL COMMANDS

TECO contains commands that enable the user to write editing programs capable of solving most complex editing problems. The iteration commands discussed in Section 3.12 are a specialized example. In addition to these, TECO has an unconditional branch command and a set of conditional execution commands that can be used to create any kind of conditional branch or conditional skip.

3.13.1 Command String Tags

To have branching in a command string, there must be a method of naming locations in the command string. Location tags in the general form

!tag!

may be placed anywhere in a command string (except in text arguments). A tag is delimited before and after by an exclamation point and may contain any number of any ASCII characters except the special characters listed in Table 2-1 and exclamation points.

Command string tags are also the recommended method for putting comments in TECO macros; they need not be referenced.

3.13.2 O Command

The unconditional branch command is the O command. The general form is

*Otag ($)

The text argument following the O command and delimited by an altmode is the tag naming the destination of the branch. The tag location itself may be either before or after the O command in the command string. The O command causes the command string execution pointer to be moved to the first character following the exclamation point that terminates the tag, and command execution continues from that point.

Tags are ignored except when an O command forces TECO to scan the command string for them.

3.13.3 Conditional Execution Commands

All conditional execution commands have the following general form:

*n''x...'

In this form, n is the numeric argument on which the decision to execute or not to execute is based. The quotation mark ('') is the first character of all conditional execution commands. The letter x represents the second character of the conditional execution command. The letter x may be any one of several letters depending on which conditional execution command is intended. The two command characters, ''x, may be followed by any string of commands terminated by an apostrophe('). If the condition specified by x is satisfied by the argument n, all the commands between ''x and ' are executed in the usual manner. If there is no branch command within the range ''x...', then after the last command in the range is executed, command execution falls through the apostrophe and executes the next command following it. If n does not satisfy the condition specified by x, then all the commands between ''x and the matching ' are skipped, and command execution continues with the first command following the apostrophe.

The commands ''x and ' must be used in matching pairs and they may be nested in the same manner that parentheses surrounding arithmetic expressions may be nested.

The individual conditional execution commands are shown in Table 3-7.

Table 3-7
Conditional Execution Commands

Command	Function
n"G	Execute the commands that follow if n >0; otherwise, skip to the matching apostrophe on the right.
n"L	Execute the commands that follow if n<0; otherwise, skip to the matching apostrophe on the right.
n"E	Execute the commands that follow if n=0; otherwise, skip to the matching apostrophe on the right.
n"N	Execute the commands that follow if n≠0; otherwise, skip to the matching apostrophe on the right.
n"C	Execute the commands that follow if n is the decimal value of an ASCII symbol constituent character (a letter, digit, $, ., or %); otherwise, skip to the matching apostrophe on the right.
n-1"L	Execute the commands that follow if n≤0; otherwise, skip to the matching apostrophe on the right.
n+1"G	Execute the commands that follow if n>0; otherwise, skip to the matching apostrophe on the right.
n"D	Execute the commands that follow if n is in the digit range (octal 60 to 71).
n"A	Execute the commands that follow if n is in the alphabetic range (octal 101 to 132 or 141 to 172).
n"V	Execute the commands that follow if n is in the lower case alphabetic range (octal 141-172).
n"W	Execute the commands that follow if n is in the upper case alphabetic range (octal 101 to 132).
n"T	Execute the commands that follow if n is 'true' (flag is on) (i.e., if n<0).
n"F	Execute the commands that follow if n is 'false' (flag is off) (i.e., if n=0).
n"S	Execute the commands that follow if n is 'successful' (i.e., if n<0).
n"U	Execute the commands that follow if n is 'unsuccessful' (i.e., if n=0).

3.13.4 Examples of the Use of Flow Control Commands

```
*!START! J→| →|PDP-10 TECO↵
$                                       !INSERT PAGE HEADING!
<S 5K $ ;R-DI6 $>                       !CHANGE 5K TO 6K!
<SWAR $ ;-3DILOVE $ >                   !CHANGE WAR TO LOVE!
PZ"NOSTART $ '                          !GET NEXT PAGE AND!
EF $ $                                  !RESTART IF NOT NULL!
```

This small editing program contains an example of the O command, i.e., the OSTART ($) command which causes a jump back to !START!. It also contains examples of command string tags used purely for documentation, e.g., !INSERT PAGE HEADING!. Normally, comments would be used only for lengthy and complex macros that the user expects to maintain.

This example also shows how a conditional execution command may be combined with an O command to produce a conditional branch. When all three of the editing functions have been performed on the page, the P command is executed to output this page and read in the next. The program then tests Z (the number of characters in the buffer) to determine if any data was read in. If Z≠0, data was read in, therefore a branch is taken to restart the program. When finally Z=0, the command OSTART ($) is skipped, and execution branches to the concluding EF command. This technique fails when a file contains null pages (consecutive form feed characters). Therefore, the (↑N) end-of-file test is preferred.

```
*YZ"N!##!Z-4000+1"G4000J0L12I ($) 0,.P0,.KO## ($) 'ZJA.-Z"NO## ($) ''PEF ($) ($)
*
```

This slightly more complex command string shows how conditional execution commands may be nested. If the first Y command produces no data, the "N command sends execution to the matching apostrophe on the right. This is the last apostrophe, immediately prior to the PEF. Otherwise, the commands following the "N are executed.

The function of this command string is to convert a file with pages of arbitrary lengths to one with pages of approximately 4000 characters each.

The command string operates as follows: Z-4000 + 1"G means if Z≥4000, i.e., there are at least 4000 characters on the current page, execute the following commands; otherwise, skip to the matching apostrophe (between ($) and Z). If Z≥4000, 4000J0L moves the pointer to the end of last complete line before the 4000th character in the buffer. Then, 12I ($) 0,.P outputs this much of the buffer with a form feed character after it, and 0,.K deletes that which has been output. Now, go back to !##! and test Z again. Stay in this loop until Z<4000. Execution then skips to the apostrophe. ZJ moves the pointer to the end of the current buffer. A appends another page, but leaves the pointer (.) at the end of the previous page. .-Z"N checks to determine if any data was actually read in. If so, the loop is reentered at !##!; otherwise the end of the file has been reached. When .-Z=0, execution skips to the matching apostrophe and then falls through the next apostrophe to the PEF that closes the output file.

```
*<NSIN ($) ;:SCOS ($) "S-3DITAN ($) 'ZJ> ($) ($)
```

This example shows how the value returned by a colon search can be used as the argument for a conditional execution command. The N command searches through the file for the first occurrence of SIN on any page. When SIN is found, the command :SCOS ($) checks for an occurrence of COS following SIN on the same page. The colon search command returns the value -1 if the search is successful, and 0 if there is no COS following SIN on the page. This value is then used as the numeric argument for the "S command. If :SCOS Ⓢ has a value of -1, the occurrence of COS that was found is replaced by TAN. If :SCOS Ⓢ has a value of 0, the commands -3DITAN Ⓢ are skipped. We then jump to the end of this page, ignoring all further occurrences of SIN and COS on it, and continue the iteration process.

3.14 Q-REGISTER COMMANDS

Q-registers are a powerful feature of TECO with many different uses. The general concept of Q-registers is explained in Section 2.8. Section 3.14 explains the TECO commands that enable the use of Q-registers.

The 36 Q-registers have the single character names A, B, C,..., Z, and 0, 1, 2,..., 9. In this section, the letter i is used to represent the name of an arbitrary Q-register.

3.14.1 Commands for Storing Integers

The following commands enable the use of Q-registers for storing single 36-bit integers.

3.14.1.1 U Command - The command nUi stores the decimal integer n in Q-register i. n may be any integer in the range $-2^{35} + \leq n \leq 2^{35} -1$. If anything was previously in Q-register i, it is destroyed.

3.14.1.2 Q Command - The command Qi is used to read the numeric value in Q-register i. Qi has no function other than returning the value in the specified Q-register as a numeric argument. It does not alter the value in the Q-register. In order to be useful, Qi must be used as a numeric argument for another command. Qi is often used in conjunction with conditional commands.

3.14.1.3 % Command - The command %i adds 1 to the integer in Q-register i and then returns the new value in the same manner as a Qi command. If the user wants to increment the value in Q-register i, but does not want the returned value to be used as an argument for the next command, he should type an altmode after the %i command.

3.14.2 Commands for Storing Character Strings

The following commands enable the user to store character strings of any length consistent with the amount of core available.

3.14.2.1 X Command - The X command copies characters from the editing buffer into a Q-register. These characters are not removed from the editing buffer. Any data previously in the Q-register is destroyed.

The various uses of the X command are as follows:

a. m,nXi (m<n) copies the m + 1st through the nth characters in the buffer into Q-register i.

b. If n>0, nXi copies everything from the current buffer pointer position through the nth following line terminator character into Q-register i. Xi is equivalent to 1Xi.

c. 0Xi copies everything from the beginning of the current line up to the buffer pointer into Q-register i.

d. If n<0, nXi copies everything from the beginning of the nth line preceding the current line up to the buffer pointer into Q-register i. -Xi is equivalent to -1Xi.

An X command may require more core space for storage than is available. If so, TECO automatically tries to expand its core. If successful, TECO prints a message in the form [nK CORE] to show the new amount of core being used. If unsuccessful, TECO prints an error message and does not execute the X command.

3.14.2.2 G Command - The command Gi fetches a copy of the entire character string stored in Q-register i and inserts it into the editing buffer at the current position of the buffer pointer. The contents of Q-register i are not changed. The buffer pointer is positioned at the right end of the character string that was inserted by the G command.

3.14.2.3 M Command - TECO command strings are basically ASCII character strings and, as such, can be inserted or read into the editing buffer just like any other text. When a command string is in the editing buffer, it can be edited but it cannot be executed, because at that point it appears to be data to TECO. However, if the user copies a command string from the editing buffer into a Q-register (using an X command), then this command string can be executed. The command that accomplishes this is the Mi command.

The command Mi executes the text in Q-register i just as if this text had been typed in the command string instead of Mi. Using an Mi command is analogous to calling a subroutine. Any TECO commands may be included in the command string or "macro" which is stored in and executed from the Q-register. Even double altmodes can be included if there are conditions under which the user wants execution to stop. The only restriction is that the commands must all be complete within the macro in the Q-register. For example, a command and its argument must not be split apart, one in the main command string with the Mi command and the other in the Q-register. Iterations and conditional execution strings, if included, must be complete within the Q-register. If an O command is used in the Q-register macro, the tag to which it branches must be in the Q-register also. M commands may be nested up to approximately 10 levels, depending on the contents of the internal pushdown list.

3.14.3 Saving the Previous Command String

After a command string has completed execution or if it has been aborted by means of the ↑G ↑G command, it may be stored in a Q-register. This is done by using an *i command as the first command in the next command string.

*i causes the entire previous command string, less one of the two concluding altmodes, to be stored in Q-register i. If the command string was aborted by ↑G ↑G, neither ↑G is stored with the command string. The previous contents of Q-register i are lost. The asterisk has this function only when used as the first command in a command string. At any other position in a command string, asterisk has its usual meaning of multiplication (see Section 2.7.2).

If the user intended to use *i as the first command but typed some other command first instead, he may recover the ability to use *i as the first command by typing enough rubouts to cause TECO to respond with a carriage return/line feed and a new asterisk. This technique will not work perfectly if some of the characters typed before the *i command were break characters (altmode, carriage return, etc.). In this case some of the leading characters of the preceding command string will be overwritten.

The *i command is especially useful when an error occurs in a long command string. See the example in Section 3.14.5.

3.14.4 Q-Register Pushdown List

An additional Q-register feature is the Q-register pushdown list, which may be used for temporary storage during the execution of a command string.

The command [i pushes the contents of Q-register i onto the stack. It does not change the contents of i.

The command]i pops the last pushed entry from the top of the pushdown list into Q-register i. The previous contents of Q-register i are lost; the entry which was popped off the pushdown list is erased from the top of the list.

NOTE

> The Q-register pushdown list is cleared after the execution of each complete command string (i.e., every time TECO types an * to indicate readiness to accept a new command string).

The maximum depth of the Q-register pushdown list is 32 entries. (This number can be changed by redefining LPF in TECO.MAC and reassembling TECO.)

3.14.5 Examples of the Use of Q-Register Commands

*QR-3UR $ $	This command subtracts 3 from the value in Q-register R

```
*Y!ST!0UC!ST + 1!:S↓
                  $ "S%C-50"LOST + 1 $'12I $ 0,.P0,.KOST $'↲
ZUEAQE-Z"NQEJOST + 1 $' PWEF $ $
```

This command string arranges a file into pages of 50 lines each. The Y command starts operation at the beginning of the file. At !ST! the command 0UC sets the value 0 in Q-register C. At !ST+1! search begins for a line feed. The command :S↓ $

returns a value of -1 if a line feed is found, in which case "S causes the following commands to be executed. The %C command increments Q-register C by 1 and returns the new value in C. If %C<50, jump back to !ST+1! and search for another line feed. However, if %C=50, proceed as follows: (1) insert a form feed character because the output command used does not output one automatically, (2) output everything from the beginning of the buffer through the form feed character, then (3) delete everything that was output and (4) go back to !ST! where the counter is reinitialized and start over.

If the search command fails to find another line, with the value in Q-register C less than 50, it returns the value 0, therefore the "S command causes a skip to the apostrophe at the end of the second line. The carriage return is ignored (see Section 3.18). The ZUE command stores the number of characters currently in the buffer in Q-register E. The A command reads in more data without moving the buffer pointer, while QE-Z"N checks the old value of Z with the new value to see if any data was actually read. If data was read, QEJ sets the pointer at the end of the old data and before the new data, then continue the line count at !ST+1!. If not, output the last page and close the file.

```
*0,.X10,.KZJG1 $$
```

This command string moves everything to the left of the pointer from its position at the beginning of the page to the end of the page. The 0,.X1 command puts everything from the top of the page to the pointer in Q-register 1. The 0.,K command deletes this data from its present position. The ZJ command moves the pointer to the end of the page. At this point the command G1 copies the contents of Q-register 1 into the buffer at the position of the pointer.

```
*ZJ-5XAJ8LGA $ $
```

This command string puts a copy of the last five lines of the page into Q-register A and then puts a copy of these five lines immediately after the eighth line in the page. It does not delete the five lines from their position at the end of the page.

```
*HK@I#J<SREAD $ ;-4DIACCEPT $ >#HXS $ $
*Y4PMS6PMS2PMSEX $ $
EXIT
↑C
.
```

In this example, the @I command inserts a short macro into the buffer. The # character is used to delimit the insertion. The HXS command stores this macro in Q-register S. In the second command string, the MS command executes the stored macro on pages 5, 11, and 13 of the input file. Note that the initial Y command clears the macro from the buffer before the first page is read in. The EX command copies all remaining pages, closes the output file, and returns to the monitor.

```
*J16<[DSDIMENSION $ 0L1XDK >J4L16<GD]> $ $
*
```

The 16 <[DSDIMENSION $ 0L1XDK> command locates the first 16 lines on the current page that have the word DIMENSION in them, stores them on the Q-register pushdown list, and then deletes them from their present positions. Then the J4L16<GD]D> command brings these 16 lines back onto the page immediately after the fourth line from the top.

```
*A LOT OF TEXT $ $
?NFI NO FILE FOR INPUT
```

Assume the user meant to insert "A LOT OF TEXT" but forgot the "I" at the beginning. The following technique illustrates the simple way to recover from this common error.

```
** Z $ $
```

Move the entire command string (with just one altmode at the end) into Q-register Z.

```
*GZ $ $
```

Move the command string from Q-register Z into the editing buffer at the current pointer position.

```
*-D $ $
*
```

Delete the altmode at the end of the command string. The rest of the command string is the text that was to be inserted, and it is now inserted.

```
*5DITITLE $ NLONG STRING $
-8DIA LOT OF TEXT $ $
?NFO No File for Output
```

An error is encountered early in a long command string. (The N-search failed because it could not output the page in the editing buffer. The commands preceding the N-search have been executed.)

```
**Z $ $
```

Save that entire command string in Q-register Z.

```
*.UP $ $
*JGZ $ $
```

Save the current pointer position. Move the pointer to the beginning of the buffer (a convenient place to edit the command string), and get the string back from Q-register Z.

```
*J9D $ $
```

Delete the commands "5DITITLE $" that have already been executed.

*EWOUT.FIL $ $	Correct the error.
*STEXT $ D $ $	Get back to the end of the command string. The D command deletes the $ at the end of the command string.
*0,.XZ $ $	Put the corrected string back into Q-register Z.
*0,.K $ $	Delete the command string from the editing buffer.
*QPJ $ $	Move the pointer back to its previous position. (In this particular case this step is not actually necessary.)
*MZ $ $ *	Execute the corrected command string.
*W<SDIVIS $;S= $RINOT ␣$ L1X1KLG1> $ $?ILL Illegal Command W	This example shows a simple technique for creating a TECO macro. The user purposely begins the command string with an illegal command. The rest of the command string is the TECO macro the user wishes to create.
**ZHKGZ $ $	When the expected error occurs, move the command string to Q-register Z, then move it into the editing buffer.
*JDHXZ $ $	Delete the W from the beginning of the macro, then save the correct macro in Q-register Z.

3.15 NUMERIC TYPEOUT COMMAND

The numeric typeout command is n=, where n is the numeric value to be typed in decimal radix. If a double = sign is used, the numeric value is typed in octal radix.

Example:

*YZ = $ $ 2529 *	This reads in a page and then types out the (decimal) number of characters in the page.
*IA== $ $ 40	This types the octal representation of the next character in the buffer.

3.16 SPECIAL NUMERIC VALUES

Several TECO commands, which have no other purpose than to return some particular numeric value, have already been discussed in this manual. These commands are B, Z, ., and Qi. Some commands that execute a function while returning a numeric value have also been discussed. These commands are %i, colon searches, and all searches within iterations. The concept of a command returning a numeric value is explained in Section 3.11.

All of these commands can be used as numeric arguments for commands that take a numeric argument, e.g., nI, n=, n;, nD, nUi, etc. To perform this function place the command, which returns a numeric value, in the position of n immediately before the command that takes a numeric argument.

There are several other commands that return numeric values; these commands are listed below.

> The nA command (where n can be any numeric value, and serves only to differentiate this command from the A (append) command) is equivalent to the ASCII value of the character immediately to the right of the buffer pointer. The nA command equals 0, if the pointer is at the end of the buffer. The nA command is used primarily with conditional commands where one is checking for a particular character or range of characters.
>
> The ↑E (or ↑E) command returns the value of the form feed flag. If, on the last input command (Y or A), input was terminated because a form feed character was encountered, E equals -1; otherwise, E equals 0. For further discussion of the form feed flag, see Sections 2.4, 3.3, 3.9, 3.10, 3.11 and 4.2.
>
> The ↑N (or ↑N) command returns the value of the end-of-file flag. If the end of the input file was seen on the last input command (Y or A), ↑N = -1; otherwise, ↑N=0. When ↑N is set to -1, it will remain -1 until cleared by an ER or EB command. When ↑N is first set to -1, new data may or may not have been read into the editing buffer. Consequently, the ↑N flag should usually be tested after processing the input data.
>
> The ↑F (or ↑F)[1] command is equivalent to the value of the console data switches.
>
> The ↑H (or ↑H) command is equivalent to the time of day in 60th's of a second (50th's where 50 Hz power is used).
>
> The ET command (without a numeric argument) returns the value of the ET flag. The ET command equals -1 if the flag is on and equals 0 if the flag is off. The significance of this flag is discussed in Section 3.6 When the ET flag is on, the T command delivers all characters, including altmodes and control characters, to the terminal in their exact form rather than substituting other characters.
>
> The EU command returns the value of the case flag. The EU value is 1 if upper case characters are flagged on typeout; 0, if lower case characters are flagged on typeout (default); and -1, if no case flagging is being performed. Refer to Section 3.6.
>
> The EH command returns the value of the error message flag. The EH value is 1 if only the error is typed; 2, if the error code plus one line is typed (default); and 3 if the full error message is typed. Refer to Section 5.2.
>
> The EO command returns the value of the version number flag. The EO value is 1 for version 21A of TECO and 2 for versions 22 and 23 of TECO. Refer to Section 3.17.
>
> The ES command returns the value of the automatic typeout flag. The ES value is -1 for automatic typeout after successful searches, 1 through 31 for automatic typeout with a line feed to indicate the pointer position, a decimal number greater than 31 for automatic typeout with the character equal to the ASCII value of the decimal number indicating the pointer position, and 0 for no automatic typeout (default). Refer to Section 3.11.

[1] When using TECO with monitors prior to the 5.02 monitor, the ↑F TECO command must be entered in the up-arrow, F form because control-F is interpreted as a special monitor command (see Section 3.18).

The ↑↑ (or ↑↑) command, followed by an arbitrary character x, is equivalent to the ASCII value of the character that immediately follows the ↑↑ in the command string. For example, in the command ↑↑ A, the character A is an argument for ↑↑ and is not interpreted as a command. (↑↑ A equals 65.)

The backslash(\ ) command (without a numeric argument) is equivalent to the decimal value of the digit string (optionally preceded by a + or - sign) immediately following the current position of the buffer pointer. The value is terminated by the first nondigit character encountered. If there is no digit string immediately following the buffer pointer, backslash equals 0. The backslash command moves the buffer pointer to the right end of the digit string and assumes the value of the digit string.

The ↑T (or ↑T) command is used to enable type-in of characters while the command string is being executed. When the ↑T command is enountered in a command string, execution of the command string stops and waits for the user to type any single character. When this character is typed, the ↑T command assumes the value of this character. Hence, the ↑T command is useful only as a numeric argument for another command, e.g., the command ↑TUC puts the ASCII value of the typed character into Q-register C.

The ↑T command is most often used with a ↑A message string preceding it (see Section 3.17). The message string is used to inform the user that TECO is waiting for a character to be typed in.

3.16.1 Examples of the Use of the Special Numeric Arguments

```
*J3C1A== $ $
71
*
```

If the fourth character in the buffer is 9, the command string returns the indicated result.

```
*J!A!1A-97"G1A-123"L1A-32UCDQCI $ OB $ ''
C!B!2A"NOA $ ' $ $
*
```

This command string converts all lower case alphabetic characters in the buffer to upper case. Starting at the beginning of the buffer (J), if the next character has a decimal ASCII value between 97 and 122 inclusive (1A-97"G1A-123"L), store the upper case value of this character in Q-register C (1A-32UC), delete the character (D) and replace it with the value in Q-register C(QCI $). Then TECO skips to !B! (OB $); otherwise, it advances to the next character (C). In either case, at !B! TECO checks to determine if there is another character in the buffer (2A"N) and if so, returns to !A! (OA $). When 2A equals 0, execution stops.

```
*P<-1- ↑E ;A> $ $
[3K CORE]
[4K CORE]
*
```

This command string outputs the current page, and then continues input until a form feed character is detected. This command string could be used on a file that is not divided into pages of a reasonable size. The A command is repeatedly executed until ↑E equals -1. When ↑E equals -1, the semicolon command causes an exit from the loop.

```
*↑F= ↑H=ET= ($)($)
23094886497
1823373
-1
*
```

This command string causes the (decimal) value of the console data switches, the time of day in 60th's of a second, and the value of the ET flag to be typed out. At this execution, the console switches were set to octal 254064000141, the time was 08:26:29:33, and the ET flag was on.

```
*↑↑MU0 ($)($)
*
```

This command string stores the ASCII value of the letter M (77) in Q-register 0.

```
*YNCHAPTER ␣($)\= ($)($)
T6
*
```

This command string searches for the next chapter heading and then types out the number of the chapter. The buffer pointer is positioned immediately following the 6, after the command in this example has been executed.

```
*<SFUNCTION ␣($) ; (↑A)↵
FUNCTION LETTER (↑A) (↑T) I($)>($)($)
FUNCTION LETTER M
FUNCTION LETTER K
FUNCTION LETTER C
*
```

Here, the (↑T) command is used as the argument for an ni ($) insert command. This command string inserts the letter which is typed in following each occurrence of the string FUNCTION that is found by the search command.

```
*<YITITLE
($) PW↑N; >($)($)
*
```

This command string inserts "TITLE" at the top of each page of a file.

3.17 TECO PROGRAMMING AIDS

Bugs can occur in editing macros written in TECO language as in any other program; therefore, TECO provides the following debugging aids for the TECO user.

3.17.1 (↑A) Command

The user can cause a statement to be typed out at any point in the execution of a command string. The (↑A) command is used to perform this function. The general form of this command is

(↑A) text (↑A)

The first (↑A) is the actual command. It can be entered either as (↑A) or ↑A. The string "text" is the character string that TECO types out when the (↑A) command is encountered. The second (↑A) command marks the end of the text to be typed and must be entered as (↑A) . The text string can contain any characters except (↑A) and the special characters listed in Table 2-1.

Example:

```
*Y!ST! ↑A NEW PAGE
 ↑A OUC!ST+1!:S↓
                 $ ''N%C-50''LOST+1 $ '12 I$0,. P0,.KOST $' ↵
ZUEAQE-Z''NOST=1 $ ' ↑A END↵
 ↑A PWEF $ $
NEW PAGE
NEW PAGE
NEW PAGE
NEW PAGE
NEW PAGE
END
*
```

This command string is identical to an example used in Section 3.14; however two ↑A commands have been added.

3.17.2 Question Mark (?) Command

The question mark command has two uses in TECO. When question mark is the first character typed by the user after TECO has typed out an error message, it has the special function described in Section 5.2. However, at any other time the question mark can be entered in a command string exactly like any other command. This use of the question mark command causes TECO to enter trace mode. In trace mode, TECO types out each command as it is executed. A second question mark command takes TECO out of trace mode.

Example:

```
*JHT?!L!1A-9''N!M!1A-58''NCOM $ 'CD ⇥ $ 'LOL $ $
AB:  LINE1
          LINE 2
C:   LINE 3
          LINE4
!L!1A-9''N!M!1A-58''NCOM$1A-58''NCO!M!1A-58''NCD   $'LOL$1A-9''NLO!L!1A-9''N!
 M!1A-58''NCO!M!1A-58''NCD       $'LO!L!1A-9''NLO!L!1A-9''N!M!1A-58''NC?POP
*J?HT $ $
J?
AB:  LINE1
     LINE2
C:   LINE3
     LINE4
*
```

After the first question mark command, TECO begins typing out each command as it is executed. This enables the user to see exactly what the command string is doing. The ?POP error message is caused by the attempt to move the pointer beyond the end of the fourth (and last) line (the end of the buffer) with the C command.

The second question mark command turns off the trace feature so that the "HT" following it is not printed.

3.17.3 The EO Value

The EO (Edit Old) feature enables TECO users to protect existing TECO macros from future changes to the TECO specifications. In most cases when features are added to TECO, the changes merely involve additional commands whose existence in no way affects old TECO macros. The EO feature does not apply to changes such as these. Occasionally, however, a new feature would cause old macros not to run properly. The EO feature is designed to protect old macros from such changes.

Every version of TECO has an EO value. For all versions of TECO up through version 21A, the EO value is 1. For TECO versions 22 and 23, and all succeeding versions until the next specification change that would affect old macros, the EO value is 2.

The EO value is always initially set to the maximum value for the version of TECO being run. This enables all new features.

By using the EO command the EO value can be set to a lower value so as to disable features of TECO that were implemented since the macro was written and which would cause the macro not to function properly. The EO command does not disable all new features, but only those that affect old macros.

0EO or	resets the EO value to the maximum (standard)
nEO (n<0)	for the version of TECO in use.
nE0 (0<n<=max)	sets the EO value to n.
EO (no argument)	returns the current value of the EO flag.

All TECO macros written before version 22 should be edited by putting "1EO" at the beginning and "0EO" at the end. All macros written with version 22 should have "2EO" at the beginning and "0EO" at the end, etc.

Table 3-8
Features Enabled by EO Values Greater Than 1

EO=1	Base value.
EO=2	(1) Standard altmode changed from ASCII 175 to 033. (2) All control characters within text arguments reserved as commands, instead of only ↑N , ↑Q , ↑S , ↑X in search strings. (3) Standard searches accept either upper or lower case alphabetic characters as a match. (4) Vertical tab and form feed recognized as end-of-line characters in addition to line feed. (5) The P command does not create form feeds. (6) Command string jumps will not accept instances of the target characters occurring within text arguments. (7) Because of (6) comments should be enclosed only by !...!. (8) The nI command must be followed by altmode. (9) The ↑G exit command is changed to ↑Z .

Examples:

```
*EO= $ $                    Initial setting is EO=2.
2
*1EOEO== $ $                Set EO value to 1.
1
*0EOEO== $ $                Revert back to EO=maximum.
2
*
```

3.18 COMMAND STRING TYPE-IN CONTROL COMMANDS

The use of two successive altmodes as the command string terminator has already been discussed in Section 2.6. The use of rubout, ↑U , and double ↑G as command string erasing commands is discussed in Section 5.1. There are other characters, however, that are useful in the creation of command strings.

3.18.1 Carriage Return, Line Feed, and Spaces

Except as text arguments, the characters carriage return and line feed are ignored in command strings. Spaces are also ignored except (1) when used in text arguments, and (2) when used between two numeric arguments as a + (see Section 2.7.2). Hence, these characters can be employed by the user when formatting command strings. The carriage return (and the monitor-supplied line feed following it) is used to enable the user to conveniently type command strings much longer than a single line. Spaces are used to lend clarity to more complicated macros.

Chapter 4
Techniques

4.1 CREATION, EXECUTION, AND EDITING OF A FORTRAN PROGRAM

This section demonstrates the use of TECO's multi-purpose commands to simplify the creation and editing of programs.

The following example shows the creation and immediate execution of a FORTRAN program.

```
.MAKE ATEST.F4↵                         Give the command to create the
                                         disk file ATEST.F4 using TECO.

*→|TYPE 1↵                               Begin insertion with the TAB command.

1    FORMAT ('COMPILER
     ARITHMETIC TEST')↵
     J=3↵
     K=7↵
     X.5  (RO)  5  (RO)  .=.5            Rub out erroneous .5.
($)  -T ($)($)                           Stop insertion and use the -T command
                                         to verify last line inserted.
     X=.5

* →|II=K/J*(X*1.E+2-K*K/                 Continue insertion.
     (3.*J))↵
     R=10.6↵
     S=3.5↵
     I=5↵
     J=2↵
     N=7↵
     Z=R+S*I/J*N/3↵
     TYPE 2,II,Z↵
1    FORMAT (I8,F20.12)↵
     END↵
($)  EX ($) ($)                          End insertion, and then use the EX
                                         command to output and close the file.

EXIT
↑C
```

```
.EXECUTE ATEST)                              Give the command to compile and
FORTRAN:  ATEST.F4                           execute ATEST.F4.
UNDEFINED LBLS                               The FORTRAN compiler discovers
2                                            errors in the program.
MULTIPLY DEFINED LBLS
1
MAIN. ERRORS DETECTED:  2
?TOTAL ERRORS DETECTED:  2
LOADING
LOADER 4K CORE
?EXECUTION DELETED
EXIT
↑C
.TECO)                                       Call TECO to edit ATEST.F4.
*SF20. ($) 0LDI2 ($) 0TT ($)($)              Change the second label 1 to 2,
2            FORMAT (I8,F20.12)              and then verify the change.
*EG ($)($)                                   Output the new version and auto-
FORTRAN:  ATEST.F4                           matically cause a repeat of the ex-
LOADING                                      ecution by using the EG command.
LOADER 4K CORE
EXECUTION
COMPILER ARITHMETIC TEST
    103       21.683333118562
EXIT
↑C                                           Success.
.
```

NOTES

a. The command MAKE ATEST.F4) is equivalent to the following sequence of commands:

```
.R TECO)
*EWATEST.F4 ($)($)
*
```

b. The -T command does not move the buffer pointer, therefore, the user can continue insertion from the point he left off.

c. In this example, the EX command is equivalent to PWEF (↑G) .

d. No filename is given with the command TECO, therefore the name of the file used in the most recent edit-class command (i.e., MAKE, or TECO command) is assumed. In the example, the command TECO) is equivalent to

```
.R TECO)
*EBATEST.F4 ($) Y ($)($)
*
```

NOTES (Cont)

e. The command SF20. ($) moves the pointer to the line the user wishes to correct. The 0L command positions the pointer immediately prior to the bad character 1. The D command deletes the 1; the I2 ($) command inserts 2 in its place. The 0TT command types out this entire line.

f. The command EG ($) ($) is equivalent (in this example) to

```
*PWEF (↑G) ($)($)
EXIT
↑C
.EXECUTE ATEST.F4↵
```

4.2 REARRANGING A FILE

In Section 3.14, an example shows the use of a Q-register in moving a segment of text from one place on a page to another place on the same page. This section describes how to move blocks of text, or whole pages, to entirely differenf places in a file.

Example:

The user has a file named PGM.MAC on the disk and this file contains data in the following form:

AB (FORM) CD (FORM) EF (FORM) GH (FORM) IJ (FORM) KL (FORM) MN (FORM) OP

where each of the letters A, B, C... represents 20 lines of text.

The user intends to rearrange the file, as shown in the following example:

AOB (FORM) D (FORM) MN (FORM) EF (FORM) ICJ (FORM) KL (FORM) P (FORM) GH

The following commands achieve this rearrangement.

.R TECO 6↵	Call TECO with extra core.
*EBPGM.MAC ($) Y ($) ($)	Specify the file and get the first page.
*NC ($) ($)	Output AB (FORM) and input CD.
*J20X1 ($) ($)	Save all of C in Q-register 1.
*20K ($) ($)	Delete C from its position in the editing buffer.
*NG ($) ($)	Output D (FORM) and input EF. Output EF (FORM) and input GH.
*HX2 ($) ($)	Save all of GH in Q-register 2.
*Y ($) ($)	Delete GH and input IJ.
*20L ($) ($)	Move the pointer to the beginning of J.

*G1 $ $	Bring in all of C from Q-register 1.
*NM $ $	Output ICJ FORM input KL, output KL FORM , and input MN.
*HX1 $ $	Save all of MN in Q-register 1 (thereby discarding the previous contents).
*Y $ $	Delete MN and input OP.
*J20X3 $ $	Save all of O in Q-register 3.
*20K $ $	Delete O from the editing buffer.
*P $ $	Output P FORM and clear the editing buffer.
*G2 $ $	Bring GH into the buffer from Q-register 2.
*HPEF $ $	Output GH, close the output file (now called nnnTEC.TMP), rename the input file PGM.BAK, and then rename the output file PGM.MAC.
*EBPGM.MAC $ Y $ $	Now edit the partially revised file just output. Loop around to the beginning of the file.
*20L $ $	Move the pointer to the beginning of B.
*G3 $ $	Bring in all of O from Q-register 3.
*ND $ $	Output AOB FORM and input D.
*PWHK $ $	Output D FORM , and then clear the buffer.
*G1 $ $	Bring in all of MN from Q-register 1.
*EX $ $	Output MN FORM and continue the input/output sequence until GH has been output. Then close the output file (called nnnTEC.TMP), delete the previous PGM.BAK, rename the input file PGM.BAK, and then rename the new output file PGM.MAC. Finally, exit to the monitor.
EXIT	
↑C	
.	

4.3 SPLITTING AND MERGING FILES

This section demonstrates the procedure to split a file into several smaller files and the procedure to merge parts of several files.

Example 1: Splitting a File

Assume the user has a file named FILE.CBL on the disk; this file contains data in the following form:

AB FORM CD FORM EF FORM GH FORM IJ FORM KL FORM MN FORM OP where each of the letters A, B, C,... represents 20 lines of text. The user wants to separate FILE.CBL into two files:

a. FILE.1 containing AB FORM CD and
b. FILE.2 containing KL FORM M

And to discard the rest of the data. To accomplish this proceed as follows.

```
.R TECO↵                         Call TECO.
*ERFILE.CBL $ EWFILE.1 $ $       Open the input file and the first
                                 output file.
*Y $ $                           Input AB.
*P $ $                           Output AB FORM and input CD.
*HPEF $ $                        Output CD and then close the out-
                                 put file FILE.1.
*←K $ $                          Clear the buffer (deleting CD from
                                 it) and continue inputting pages of
                                 the file and searching for K. If K
                                 is not found on a given page, clear
                                 the buffer, and read in the next
                                 page. The ← command does not
                                 perform output. Thus EF, GH, and
                                 IJ are all read in and then deleted.
                                 When KL is read in, the search stops.
*EWFILE.2 $ $                    Open the second output file.
*P $ $                           Output KL FORM and input MN.
*20L $ $                         Position pointer at the end of M.
*0,.PEF $ $                      Output M and then close the output
                                 file FILE.2.
* ↑C                             Exit to the monitor with the job
                                 completed.

.
```

Example 2: Merging Files

Assumed the user has two files:

a. MATH.BAK containing

    AB FORM CD FORM EF FORM GH FORM IJ FORM KL

b. MATH.F4 containing

    A'B' FORM C'D' FORM E'F'

Where A, B, C,... each represents 20 lines of text, and A', B',... represent updated versions of A, B,....

The user wants to merge MATH.F4 with the latter half of MATH.BAK to produce:

MATH.NEW containing

A'B (FORM) C'D' (FORM) E'F' (FORM) GH (FORM) IJ (FORM) KL

He proceeds as follows.

.R TECO	Call TECO.
*ERMATH.F4 ($) EWMATH.NEW ($)($)	Open the first input file and the output file.
*Y ($)($)	Input A'B'.
*NF' ($)($)	Output A'B' (FORM) , input C'D', output C'D' (FORM) , and input E'F'.
*PW ($)($)	Output E'F' (FORM) .
*ERMATH.BAK ($)($)	Close input from MATH.F4, and open MATH.BAK for input.
*Y ($)($)	Delete E'F' from the buffer and input AB.
*←G ($)($)	Delete AB, input and delete CD and EF, then input GH.
*NL ($)($)	Output GH (FORM) , input and then output IJ (FORM) , then input KL.
*HPEF (↑G) ($)($) EXIT ↑C .	Output KL, close the output file MATH.NEW, and then exit to the monitor with the job completed.

The technique shown in Example 2 illustrates the best method for recovering from the error indicated by the error message:

?OUT-200000 Output Error 200000 - Output File 018TEC.TMP Closed

If this error occurs during an editing job initialized by the TECO filnam.ext↩ command or an EB command, the incomplete output file has a temporary name of the form nnnTEC.TMP (see Section 3.2); otherwise, the incomplete output file will have the name specified by the user. (Refer to Appendix A for a list of error messages and their meanings.)

Example 3 is more explicit illustration of recovery from the foregoing error.

Example 3: Recovery from an Output Error

```
.TECO FIL.DOC)
*edit a few pages ($)($)
*P ($)($)
?OUT-200000   Output Error 200000 - Output File 018TEC.TMP Closed
*ER018TEC.TMP ($) EWFIL.NEW ($) Y ($)($)
*Nlast page edited and successfully output ($)($)
*PW ($)($)
*ERFIL.DOC ($) Y ($)($)
*←last page edited and successfully output ($)($)
*Y and edit next page ($)($)
*Nnext place to edit ($)($)
*finish editing normally ($)($)
*EX ($)($)
EXIT
↑C
.RENAME FIL.DOC=FIL.NEW)
```

4.4 EXAMPLE OF AN ADVANCED TECO MACRO

This section demonstrates a TECO macro for formatting DECsystem-10 Macro assembly language programs.

The procedure for executing this macro is as follows:

```
.R TECO 6)                        Call TECO with enough core to cov-
                                  er the maximum page size.

*ERDTA7:PGMFMT.TEC ($)($)         Open the file containing the macro
                                  itself for input.

*YHX1 ($)($)                      Input the macro and save it in Q-
                                  register 1.

*EBPROGRM.MAC ($)($)              Open for editing the file that is to
                                  be formatted by the macro.

*Y ($)($)                         Read in the first page of the file.
*M1 ($)($)                        Execute the macro.
* (↑C)                            Exit with job completed.
.
```

Formatting Macro (PGMFMT.TEC)

```
1EO!START!0UL<S↓
$;%L>ZJR1A-10"N%L'                                        !COUNT LINES ON PAGE!
!LOOP!JQL<0UC                               !EXECUTE LOOP ONCE FOR EACH LINE!
!FSTCH!1A"COTAG $'
!FSTCH2!1A-9"ECOOP $ '1A-32"NOZ $'
!FSTCH3!%C-8"GOZ $ 'C1A-32"EOFSTCH3$'1A-9"EQC-7"GOZ $ 'COFSTCH4 $'
QC-8"GOZ $'
!FSTCH4!0US                                  !CHANGE LEADING SPACES TO A TAB!
!FSTCH5!-D%S-QC"LOFSTCH5 $'  ⇥$OOP $
!TAG!%C-6"GOZ $ 'C1A"COTAG $ '1A-58"NOZ $'
!COLON!0USC1A-9"ECOOP $ '1A-32"NOZ $'                     !LOOK FOR A COLON!
!COLON2!%S $ C1A-32"EOCOLON2 $ 'QC+QS-7"GOZ $ 'QC+QS-7"EOCOLON3
$'1A-9"NOZ $ 'D
!COLON3!R1A-32"EDOCOLON3 $
'C ⇥$                                      !CHG SPACES AFTER COLON TO TAB!
!OP!1A-90"GOZ $ '1A-65"LOZ $ '0UC
!OP2!%C $C1A-90"GOZ $'1A-64"GOOP2 $'1A-57"GOZ $ '1A-47"GOOP2$'
  1A-9"EC1A-32"EOZ $ '1A-9"EOZ $                   !GIVE UP IF NO OPERANDS!
  1A-32"NOZ $ 'QC-7"GOZ $ 'C1A-32"EOZ $ '1A-9"EOZ $'
  -D ⇥ $                   !IF A SINGLE SPACE FOLLOWS OP, CHANGE IT TO A TAB!
!OP3!0UC
!EOL!%C $                              !LOOK FOR END OF LINE OR SEMI-COLON!
!EOL2!1A-9"EOEOL $ '1A-13"G1A-59"NOEOL $ '0US
!SEMI!R1A-32"N1A-9"NOSEMI2 $ ' '%S $                     !LINE UP COMMENTS!
!SEMI2!QS"NC ⇥$ QC-QS-8"L ⇥$ ' ' '
!Z!L>
PZ"NOSTART $ '0UC                                         !LOOK FOR NEXT PAGE!
!GET!YZ"NOSTART $ '%C-10"NOGET $ 'EF0EO !QUIT WHEN 10YANKS YIELD NO DATA!
```

An explanation of the macro follows.

1EO	The 1EO command enables only those features found in versions prior to 21A for which this macro was written.
!START!	It is assumed that the pointer is at the beginning of the first page of the file.
0UL	Initialize line counter.
<S↓ $;%L>	Count the line feed characters on the page.
ZJR1A-10"N%L'	If the last character on the page is not a line feed, count those characters following the last line feed character as one more line.

!COUNT LINES ON PAGE!	This is the standard technique for including comments in TECO macros.
JQL<	Execute everything which follows, down to the > character on the second to the last line, once for each line on the page.
0UC	Initialize first character counter for the line.
1A-90"GOZ $'	If the first character in the line is greater than Z (decimal 90) in the ASCII set, skip this line by jumping to !Z!.
1A"COTAG $'	If the first character is alphabetic or period, or %, or a dollar sign (i.e., legal as the first character of a Macro language symbol), go to !TAG!. Otherwise, go to !FSTCH2!.
!FSTCH2!1A-9"ECOOP $'	If the first character is a tab, move the pointer past the tab, then go to !OP!.
1A-32"NOZ $'	If the first character is a space, continue on to !FSTCH3!; otherwise, skip this line.
!FSTCH3!%C-8"GOZ $'C	Increment the character counter (counting leading spaces), and if the new total is more than eight spaces, skip to the next line; otherwise, move the pointer to the next character.
1A-32"EOFSTCH3 $'	If the next character is another space, go back to !FSTCH3!.
1A-9"EQC-7"GOZ $' COFSTCH4 $ 'QC-8"GOZ $'	If the character is neither a tab nor a space, and if more than eight spaces preceded this character, skip to the next line. If the character is a tab, but more than seven spaces preceded this tab, skip to the next line. Otherwise, go to !FSTCH4!.
!FSTCH4!0US	Initialize space deleted counter.
!FSTCH5!-D	Delete last space seen.
%S-QC"LOFSTCH5 $'	Increment space deleted counter. Then, if the new value of this counter is still less than the number of characters (spaces) counted on the line, go back to !FSTCH5!.
⇥$ OOP $	When the count of spaces deleted reaches the number of spaces there were, insert a tab and then go to !OP!.
!TAG!%C-6"GOZ $ 'C	Increment the character counter (counting characters in the tag), and if the new total is more than six spaces, skip to the next line. Otherwise, move the pointer to the next character.
1A"COTAG $'	If the next character is a symbol constituent, go back to !TAG!.
1A-58"NOZ $ 'OCOLON $	If the character is a colon, go on to !COLON!; otherwise, skip to the next line.
!COLON!0USC	Initialize counter of spaces following the colon, and move the pointer to the next character.

`1A-9"ECOOP $'`	If the character after the colon is a tab, move the pointer to the next character and go to !OP!.
`1A-32"NOZ $'`	If the character is not a space either, skip to the next line. Otherwise, continue on to !COLON2!.
`!COLON2!%S $ C`	Increment the space-following-colon counter, and then move the pointer to the next character. The altmode following %S prevents the value returned by the %S command from being used as an argument for the following C command.
`1A-32"EOCOLON2 $'`	If the next character is another space, go back to !COLON2!.
`QC+QS-7"GOZ $'`	If the total count of the symbol characters before the colon and the spaces after the colon is more than seven, skip to the next line.
`QC+QS-7"EOCOLON3 $'`	If the count mentioned above exactly equals seven, go to !COLON3!.
`1A-9"NOZ $ 'D`	With the count mentioned above less than seven, if the next character is not a tab, skip to the next line. If this character is a tab, delete it and continue to !COLON3!.
`!COLON3!R`	Move pointer back one character (i.e., back past the next space or the colon).
`1A-32"EDOCOLON3 $'`	If the character passed over is a space delete it and go back to !COLON3!.
`C ⇥$ OOP $`	Otherwise, the pointer is now in front of the colon. Move it forward over the colon and then insert a tab to replace the deleted spaces. Then go to !OP!.

`!OP!1A-90"GOZ $'`
`1A-65"LOZ $ '0UC`
	If the first character in the operator field is not alphabetic, skip to the next line. Otherwise, initialize the op field character counter.

`!OP2!%C $ C`	Increment operator field character counter and then move pointer to the next character.

`1A-90"GOZ $'`
`1A-64"GOOP2 $'`
	If the next character is above Z in the ASCII set, skip to the next line. If it is alphabetic, go back to !OP2!.

`1A-57"GOZ $'`
`1A-47"GOOP2 $'`
	If the character is greater than the digit nine in the ASCII set, skip to the next line. If it is a digit, go back to !OP2!.

`1A-9"E`	If the character is not a tab, skip to the ' following the comment "GIVE UP IF NO OPERANDS". The leading spaces are for appearance only and are ignored. (A tab could not be used for this purpose.)

`C1A-32"EOZ $'`
`1A-9"EOZ $ 'OOP3 $`
	If it is a tab, move the pointer to the next character. If this character is a tab or a space, skip to the next line. If the character is anything else, go to !OP3!.

`1A-32"NOZ $'`	If the letter following the last letter or digit of the operator is anything but a space (or the tab that was processed above), skip to the next line.
`QC-7"GOZ $ 'C`	If the operator is more than seven characters long, skip to the next line. Otherwise, move the pointer to the character after the space following the operator.
`1A-32"EOZ $ 1A-9"EOZ $'`	If this character is another space or a tab, skip to the next line.
`-D ⇥ $`	Delete the space between operator and operand and insert a tab in its place.
`!OP3!0UC`	Initialize operand character counter.
`!EOL!%C $ C`	Increment operand character counter and move pointer to the next character.
`1A-9"EOEOL $'`	If the character is a tab, go back to !EOL!.
`1A-13"G`	If the character is equal to or below carriage return in the ASCII set, skip to the next line by skipping to the last ' in the line starting with !SEMI2!.
`1A-59"NOEOL $'`	If the character is not a semicolon, go back to !EOL!.
`0US`	Initialize the counter for spaces and tabs preceding the semicolon.

`!SEMI!R1A-32"N1A-9"`
`NOSEMI2 $ ''`
	Move the pointer back one more character from the semicolon. If this character is not a space or tab, go to !SEMI2!.

`%S $ DOSEMI $`	Count the space or tab, then delete it and go back to !SEMI!.
`!SEMI2!QS"N`	If there are no spaces or tabs preceding the semicolon, skip to the next line by skipping to the next to the last ' in this line. This check prevents most cases of inserting tabs before semicolons that occur in SIXBIT or ASCIZ fields.
`C ⇥ $`	Move pointer forward over the last character seen, and then insert a tab before the semicolon.
`QC-QS-8"L ⇥ $ '''`	If the number of characters in the operand field, not counting the spaces and tabs preceding the semicolon, is less than eight, insert a second tab. Otherwise, skip to the next line.
`!Z!L>`	Move pointer to the next line, and then go back to the beginning of the loop.
`P`	When every line on the page has been edited by the loop, output this page, clear the buffer, and then yank in the next page.
`Z"NOSTART $'`	If the yank produces any new data, go back to !START!.

0UC	Otherwise, initialize the yank counter.
!GET!YZ"NOSTART $'	Try another yank. If this produces any new data, go back to !START!.
%C-10"NOGET $ '	Increment the yank counter, and if it is still less than 10, try again.
EF	When a total of 10 straight yanks after the P command fails to produce any new data, close the output file.
0EO	The 0EO command re-enables TECO commands to the current version.

Chapter 5
User Errors

This chapter describes two types of errors: (1) typing errors discovered by the user before a command string is completed, and (2) command errors detected by TECO. The user should realize, however, that there is a third class of error. Because TECO interprets almost every character as a command, there can be cases where, if the user fails to notice a command string typing error, TECO executes a command that the user did not intend. For example, if the user meant to type the command

```
*INAME ($)($)
```

but forgot to type the "I", then TECO is forced to interpret the command as an N-search for "AME" and act accordingly. There is no way to protect the user from errors of this type.

5.1 ERASING COMMANDS

If the user makes an error while typing a command string and discovers the error before terminating the command string (with a double altmode), the error can be corrected using one of three erasing commands described below. All of these must be typed before the double altmode that terminates the command string.

5.1.1 Rubout Command

Rubout is used to erase typed-in characters one at a time starting with the last character typed in.

Example

After typing the portion of the command string shown below, the user discovers that he has mispelled the name "Ericson".

```
*3LKILEIF ERICXON
```

To nullify the error, he types three successive rubouts. As he does this, TECO responds by retyping the characters which are being rubbed out.

```
*LKILEIF ERICXON (RO) N (RO) O (RO) X
```

The actual function of the rubout character is to delete the last typed character in the command string. Consequently, if the incorrect character is not the last in the string, all characters back to that point must also be rubbed out.

Rubout is a nonprinting character; consequently, the actual line appears as follows:

```
*LKILEIF ERICXONNOX
```

When the user has rubbed out the incorrect character, he continues the command string from the last correct character.

```
*3LKLEIF ERICXONNOXSON $ 0TT $ $
```

Two successive rubouts are required to erase a carriage return and the monitor-generated line feed following it.

5.1.2 Double ↑G Command

The command ↑G ↑G (two successive control-Gs) is used to erase an entire command string.

In the following example the user has decided, after typing the "N", to quit and start over. He does this by typing two successive control-Gs. (Control-G echoes visibly as "↑G" and audibly as a bell ring.)

```
*3LKILEEF ERIXON ↑G ↑G
*
```

↑G ↑G cannot be typed in the alternate up-arrow, character form described in Section 2.2.

5.1.3 ↑U Command

The ↑U command is another erasing command available to the TECO user. The ↑U command erases everything in a command string back to the last carriage-return/end-of-line character pair. It does not erase the carriage-return nor end-of-line character. The end-of-line characters are line feed, vertical tab and form feed.

In monitors previous to 5.02B, control-U is intercepted by the monitor and erases only back to the most recent break character (carriage-return, linefeed, formfeed, altmode).

Example 1:

```
*ILINE ONE↵
LINE TWO↵
LINE THREE↵
KINE FOUR ↑U
LINE FOUR↵
$$
*
```

The user makes an error typing the fourth line and uses the ↑U command to erase the entire line. The ↑U command causes a carriage return-line feed to be echoed but the carriage return and line feed are not inserted.

Example 2:

```
*ILINE ONE ↵
LINE TWO ↵
KINE THREE ↵
LINE FOUR  ↑U
 RO
  ↑U
LINE THREE ↵
LINE FOUR ↵
$$
*
```

The user makes an error on the third line but does not notice *it until* he is on the fourth line. In order to erase back to his error without erasing the entire command string, he types control-U, rubout, control-U. The first ↑U erases "LINE FOUR". The rubout erases the line feed that marks the end of the third line, and the second ↑U erases "LINE THREE" and the carriage-return at its end.

5.1.4 Bell-Space Command

The bell-space command is not actually an erasing command, but it is usually used in conjunction with the erasing commands. Its function is to cause the current line of the command string to be retyped. It is used when the user has typed so many rubouts on a line that he cannot tell exactly what has been typed.

Specifically, if the user types ↑G and space in succession, everything in the command string back to, but not including, the last carriage return line feed pair is immediately retyped on the next line. The user may then continue the command string just as if bell-space had not been typed. The bell-space is not stored in the command string. Neither does it remove anything from the command string.

Example:

```
*ISTAET: RO  :  RO  T  RO  ERT:→|TRZE →|SW,  ↑G␣
START: TRZE SW,CCLFLG →|; CLEAR FLAG
$$
*
```

5.2 ERROR MESSAGES

When TECO encounters an illegal command or a command that cannot be executed, an error message is printed on the user's terminal. An error message consists of three parts, some of which are printed automatically and some of which can be printed at the user's option. The first part of the message is a question mark followed by a 3-letter mnemonic code for the error message. The second is a brief, one-line, statement of the error condition. The last part is a more complete explanation of the error.

In the standard version of TECO the first two parts of the error message are automatically printed; the third part is printed only if the user requests it. In Section 5.2.2 there is an explanation of how to obtain the optional parts of the error message, and in Section 5.2.3 there is an explanation of how to change TECO so that more or less of the error message is printed automatically.

When an error message is generated, the command to which it refers is not executed, the remainder of the command string is ignored, and TECO returns to command mode. Also any commands that the user might have typed ahead are erased.

Example:

```
*SWORD $ -4DUINEW $$
?NAU No Argument Before U
*
```

The error message points out the presence of a U command not preceded by a numeric argument. The commands SWORD $ -4D have been executed, but the commands UINEW $ have not.

After an error message has been printed, the user has the option to use either or both of two special commands, ? and /, that are designed to help the user after a command error has been encountered. These commands are described below. Note, however, that these two commands have the special properties described below only immediately after an error has occurred. If any other command is typed after an error has occurred, TECO assumes that a new command string is being typed and the ability to use the ? and / commands for this error is lost.

Also note that the *i command described in Section 3.8.8 is frequently useful after an error is encountered.

5.2.1 Question Mark Command

In some cases, the user may not be able to determine immediately which command in the string caused the error. This could occur, for example, if there were several commands of the same type in the command string. In such a case, the user can use the question mark command to obtain more information. The question mark command, when used immediately after an error message typeout, causes the offending command and several of the preceding characters in the command string to be typed out. A maximum of 10 characters of the command string are typed; usually this number is sufficient to identify the command that caused the error. Note that when the question mark command is used in this manner, it is not necessary to type altmode or any other character after the question mark.

A second question mark is always typed after the last character of the group. The character at which the error was detected is the last character before the second question mark typed.

Another use of the question mark command is explained in Section 3.17.

Example:

```
*H X 2PG2ZJ-1U2PG2ZJ $ $
?NTQ No Text in Q-register 2
*? 2ZJ-1U2PG2?
*
```

According to the error message, one of the G2 commands specifies a Q-register that does not contain text. The question mark command is used and the second G2 command is identified as the offending command.

5.2.2 Slash Command

When a command error occurs, one or more of the three parts of the corresponding error message is automatically printed. If all three parts of the error message have not yet been printed and the user needs a more detailed explanation of the error, he may type the slash command to obtain more information.

The slash command, when used immediately after an error message, causes the next unprinted part of the error message to be printed. It may be used enough times to cause all three parts of the error message to be printed, but no more. Note that when the slash command is used in this manner, it is not necessary to type altmode or any other character after the slash.

NOTE

The verbal parts 2 and 3 of the error messages printed by TECO are obtained from a system file (TECO.ERR) external to TECO itself. If for any reason this file cannot be read, only the code portion of the error message is printed, and this is followed by the special message "?EEE Unable to Read Error Message File". In this case the / command cannot be used.

Another use of / is described in Section 2.7.2.

Example:

```
*EBTEST.CBL $ EX $ $
?BAK Cannot Delete Old Backup File
*/ Failure in rename process at close of editing job
initiated by an EB command or a TECO command. There
exists an old backup file TEST.BAK with a protection
<777> such that it cannot be deleted. Hence the in-
put file TEST.CBL cannot be renamed to "TEST.BAK".
The output file is closed with the filename "009TEC.TMP".
The RENAME UUO error code is 2.
```

5.2.3 EH Command

As was stated above, TECO error messages consist of three parts. The first, or code, part is always automatically typed. With the standard version of TECO, the second, brief message, part is also automatically typed. The third, more lengthy part is obtained by the / command at the option of the user.

By use of the EH command, the user may change TECO so that more or less of the error message is automatically typed. This is done as follows:

1EH sets TECO so that only the code part of the error message is automatically printed.

2EH sets TECO so that both the code and the 1-line message parts of the message are automatically printed.

3EH sets TECO so that all three parts of the error message are always automatically typed.

0EH resets TECO to the system standard mode of error message typeout. (Normally equivalent to 2EH.)

EH (with no argument) returns the value of the current EH setting.

Appendix A
TECO Error Messages

The following table lists the error messages from TECO. The three-letter message preceded by a question mark is always typed; the second part of the error message, which is a short explanation of the error, is always typed in standard versions of TECO. The detailed message is typed if the user types a slash command (/) immediately following the short error message.

Table A-1
TECO Error Messages

?ARG		Improper Arguments The following argument combinations are illegal:
1)	,	(no argument before comma)
2)	m,n,	(where m and n are numeric terms)
3)	H,	(because H=B,Z is already two arguments)
4)	,H	(H following other arguments)
?BAK		Cannot Delete Old Backup File Failure in rename process at close of editing job initiated by an EB command or a TECO command. There exists an old backup file filnam.BAK with a protection<nnn> such that it cannot be deleted. Hence the input file filnam.ext cannot be renamed to "filnam.BAK". The output file is closed with the filename "nnnTEC.TEMP", where nnn is the user's job number. The RENAME UUO error code is nn.
?COR		Storage Capacity Exceeded The current operation requires more memory storage than TECO now has and TECO is unable to obtain more core from the monitor. This message can occur as a result of any one of the following things: 1) command buffer overflow while a long command string is being typed, 2) Q-register buffer overflow caused by an X or [command, 3) editing buffer overflow caused by an insert command or a read command.
?COS		Contradictory Output Switches The GENLSN and SUPLSN switches may not both be used with the same output file.
?EBD		EB with Device dev Is Illegal The EB command and the TECO command may be specified only with file structured devices, i.e., disk and DECtape.

Table A-1 (Cont)
TECO Error Messages

?EBF	EB with Illegal File filnam.ext The EB command and the TECO command may not be used with a file having the filename extension ".BAK" or with a file having the name "nnnTEC.TMP". Where nnn is the user's job number, the user must either use an ER-EW sequence, or rename the file.
?EBO	EB, EW, or EZ Before Current EB Job Closed After an output file has been opened by a TECO command or an EB command, no further EB, EW, or EZ commands may be given until the current output file is closed.
?EBP	EB Illegal Because of File filnam.ext Protection The file filnam.ext cannot be edited with an EB command or a TECO command because it has a protection <nnn> such that it cannot be renamed at close time.
?EEE	Unable to Read Error Message File An error, whose code was typed previous to this error message, has occurred, and while TECO was trying to find the proper error message in the error message file, one of the following errors occurred: 1) the error message file, TECO.ERR, could not be found on device SYS:, 2) an input error occurred while TECO was reading the file TECO.ERR, 3) the error message corresponding to that error code is missing from TECO.ERR, 4) the user's TECO job does not currently have enough room for a buffer to read the error message file into, and no more core can be obtained from the monitor, 5) for some strange reason device SYS: could not be initialized for input.
?EMA	EM with Illegal Argument nn The argument n in an nEM command must be greater than zero.
?EMD	EM with No Input Device Open EM commands apply only to the input device, and so should be preceded by an ER (or equivalent) command. To position a tape for output, that unit should be temporarily opened for input while doing the EM commands.
?ENT-00	Illegal Output Filename "filnam.ext" ENTER UUO failure 0. The filename "filnam.ext" specified for the output file cannot be used. The format is invalid.
-01	Output UFD dev:[pj,pg] Not Found ENTER UUO failure 1. The file filnam.ext[pj,pg] specified for output by an EW, EZ, or MAKE command cannot be created because there is no user file directory with project-programmer number [pj,pg] on device dev.
-02	Output Protection Failure ENTER UUO failure 2. The file filnam.ext[pj,pg] specified for output by an EW, EZ, EB, MAKE, or TECO command cannot be created either because it already exists and is write-protected <nnn> against the user, or because the UFD it is to be entered into is write-protected against the user.

Table A-1 (Cont)
TECO Error Messages

-03	Output File Being Modified ENTER UUO failure 3. The file filnam.ext specified for output by an EW, EZ, EB, MAKE, or TECO command cannot be created because it is current being created or modified by another job.
-06	Output UFD or RIB Error ENTER UUO failure 6. The output file filnam.ext cannot be created because a bad directory block was encountered by the monitor while the ENTER was in progress. The user may try repeating the EW, EB, or TECO command, but if the error persists, it is impossible to proceed. Notify your system manager.
-14	No Room or Quota Exceeded on dev: ENTER UUO failure 14. The output file filnam.ext cannot be created because there is no more free space on device dev:, or because the user's quota is already exceeded there.
-15	Write Lock on dev: ENTER UUO failure 15. The output file filnam.ext cannot be created because the output file structure is write-locked.
-16	Monitor Table Space Exhausted ENTER UUO failure 16. The output file filnam.ext cannot be created because there is not enough table space left in the monitor to allow the ENTER. The user may try repeating the EW, EB, or TECO command, but if the error persists he will have to wait until conditions improve.
-23	Output SFD Not Found ENTER UUO failure 23. The output file filnam.ext cannot be created because the sub-file-directory on which it should be ENTERed cannot be found.
-24	Search List Empty ENTER UUO failure 24. The output file filnam.ext cannot be created because the user's file structure search list is empty.
-25	Output SFD Nested too Deeply ENTER UUO failure 25. The output file filnam.ext cannot be created because the specified SFD path for the ENTER is nested too deeply.
-26	No Create for Specified SFD Path ENTER UUO failure 26. The output file filnam.ext cannot be created because the specified SFD path for the ENTER is set for no creation.
-nn	ENTER Failure nn on Output File filnam.ext The attempted ENTER of the output file filnam.ext has failed and the monitor has returned an error code of nn. This error is not expected to occur on an ENTER. Please send the TTY printout showing what you are doing to DEC with an SPR form.

Table A-1 (Cont)
TECO Error Messages

?EOA	nEO Argument Too Large The argument n given with an EO command is larger than the standard (maximum) setting in EO=n for this version of TECO. This must be an older version of TECO than the user thinks he is using; the features corresponding to E0=n do not exist.
?FNF-00	Input File filnam.ext Not Found LOOKUP UUO failure 0. The file filnam.ext specified for input by an ER, EB, or TECO command was not found on the input device dev.
-01	Input UFD dev:[pj,pg] Not Found LOOKUP UUO failure 1. The file filnam.ext[pj,pg] specified for input by an ER, EB, or TECO command cannot be found because there is no User File Directory with project-programmer number [pj,pg] on device dev.
-02	Input Protection Failure LOOKUP UUO failure 2. The file filnam.ext[pj,pg] specified for input by an ER, EB, or TECO command cannot be read because it is read-protected <nnn> against the user.
-06	Input UFD or RIB Error LOOKUP UUO failure 6. The input file filnam.ext cannot be read because a bad directory block was encountered by the monitor while the LOOKUP was in progress. The user may try repeating the ER, EB, or TECO command, but if the error persists all is lost. Notify your system manager.
-16	Monitor Table Space Exhausted LOOKUP UUO failure 16. The input file filnam.ext cannot be read because there is not enough table space left in the monitor to allow the LOOKUP. The user may try repeating the ER, EB, or TECO command, but if the error persists he will have to wait until system conditions improve.
-23	Input SFD not Found LOOKUP UUO failure 23. The input file filnam.ext cannot be found because the sub-file-directory on which it should be looked up cannot be found.
-24	Search List Empty LOOKUP UUO failure 24. The input file filnam.ext cannot be found because the user's file structure search list is empty.
-25	Input SFD Nested too Deeply LOOKUP UUO failure 25. The input file filnam.ext cannot be found because the specified SFD path for the LOOKUP is nested too deeply.
-nn	LOOKUP Failure nn on Input File filnam.ext The attempted LOOKUP on the Input file filnam.ext has failed and the monitor has returned an error code of nn. This error is not expected to occur on a LOOKUP. Please send the TTY printout showing what you were doing to DEC with an SPR form.

Table A-1 (Cont)
TECO Error Messages

?FUL	Device dev: Directory Full ENTER UUO failure n. The file filnam.ext specified for output by an EW or MAKE command cannot be created on DECtape dev because the tape directory is full.
?IAB	Incomplete <...> or (...) in Macro A macro contained in a Q-register and being executed by an M command contains an iteration that is not closed within the Q-register by a >, or a parenthetical expression that is not closed within the Q-register by a).
?ICE	Illegal Control-E Command in Search Argument A search argument contains a ↑E command that is either not defined or incomplete.
?ICT	Illegal Control Command ↑<char> in text Argument In order to be entered as text in an Insert command or search command, all control characters (↑@ - ↑H and ↑N - ↑←) must be preceded by ↑R or ↑T. Otherwise they are interpreted as commands. The control character "↑<char>" is an undefined text argument control command.
?IDV	Input Device dev Not Available Initialization failure. Unable to initialize the device dev for input. Either the device is being used by someone else right now, or else it does not exist in the system.
?IEC	Illegal Character "<char>" After E The only commands starting with the letter E are EB, EF, EG, EH, EM, EO, ER, ET, EU, EW, and EZ. When used as a command (i.e., not in a text argument) E may not be followed by any character except one of these.
?IEM	Re-Init Failure on Device dev After EM Unable to re-initialize the device dev after executing an EM command on it. If this error persists after retrying to initialize the device with an ER command (or EW command if output to the device is desired), consult your system manager.
?IFC	Illegal Character "<char>" after F The only commands starting with the letter F are FS and FN. When used as a command (other than EF or in a text argument) F may not be followed by any character other than one of these.
?IFN	Illegal Character "<char>" in Filename File specifications must be of the form dev:filnam.ext[m,n] $ where dev, filnam, and ext are alphanumeric, and m and n are numeric. No characters other than the ones specified may appear between the EB, ER, EW, or EZ command and the altmode terminator($).
?ILL	Illegal Command <char> The character "<char>" is not defined as a valid TECO command.
?ILR	Cannot Lookup Input File filnam.ext. to Rename It Failure in rename process at close of editing job initiated by an EB command or a TECO command. Unable to do a LOOKUP on the original input file dev:filnam.ext in order to rename it "filnam.BAK". The output file is closed with the name "nnnTEC.TMP", where nnn is the user's job number. The LOOKUP error code is nn.

Table A-1 (Cont)
TECO Error Messages

?INP-nn0000	Input Error nn0000 on File filnam.ext. A read error has occurred during input. The input file filnam.ext has been released. The user may try again to read the file, but if the error persists, the user will have to return to his backup file. The input device status word error flags are nn0000. (Note: This number represents the I/O status word (rh) with bits 22-35 masked out.) (040000 -- block too large). (100000 -- parity or checksum error). (140000 -- block too large and parity error). (200000 -- device error, data missed). (240000 -- block too large and device error). (300000 -- parity error and device error). (340000 -- block too large, parity error, and device error). (400000 -- improper mode). (440000 -- block too large and improper mode). (500000 -- parity error and improper mode). (540000 -- block too large, parity error, and improper mode). (600000 -- device error and improper mode). (640000 -- block too large, device error, and improper mode). (700000 -- parity error, device error, and improper mode). (740000 -- block too large, parity error, device error, and improper mode).
?IOS	Illegal Character "<char>" in I/O Switch The only valid characters in switches used with file selection commands are the alphabetic characters.
?IQC	Illegal command "<char> The only valid " commands are "G, "L, "N, "E, "C, "A, "D, "V, "W, "T, "F, "S, and "U.
?IQN	Illegal Q-register Name "<char>" The Q-register name specified by a Q, U, X, G, %, M, [,], or * command must be a letter (A thru Z) or a digit (0 thru 9).
?IRB	Cannot Rename Input File filnam.ext to filnam.BAK Failure in rename process at close of editing job initiated by an EB command or a TECO command. The attempt to rename the original input file filnam.ext to the backup filename "filnam.BAK" has failed. The output file is closed with the name "nnnTEC.TMP", where nnn is the user's job number. The RENAME UUO error code is nn.
?IRN	Cannot RE-Init Device dev for Rename Process Failure in rename process at close of editing job initiated by an EB command or a TECO command. Cannot reinitialize the original input device dev in order to rename the input file filnam.ext to "filnam.BAK". The output file is closed with the name "nnnTEC.TMP", where nnn is the user's job number.
?ISA	n Argument with Search Command The argument preceding a search command indicates the number of times a match must be found before the search is considered successful. This argument must be greater than 0.

Table A-1 (Cont)
TECO Error Messages

?MAP	Missing ' In attempting to execute a conditional skip command (a " command whose argument does not satisfy the required condition) no ' command closing the conditional execution string can be found. Note: n"...' strings must be complete within a single macro level.
?MEE	Macro Ending with E A command macro being executed from a Q-register ends with the character "E". This is an incomplete command. E is the initial character of an entire set of commands. The other character of the command begun by E must be in the same macro with the E.
?MEF	Macro Ending with F A command macro being executed from a Q-register ends with the character "F" (not an EF). This is an incomplete command. F is the initial character of an entire set of commands. The other character of the command begun by F must be in the same macro with the F.
?MEO	Macro Ending with Unterminated O Command The last command in a command macro being executed from a Q-register is an O command with no altmode to mark the end of the tag-name argument. The argument for the O command must be complete within the Q-register.
?MEQ	Macro Ending with " A command macro being executed from a Q-register ends with the " character. This is an incomplete command. The " command must be followed by one of the characters G, L, N, E, C, A, D, V, W, T, F, S, or U to indicate the condition under which the following commands are to be executed. This character must be in the Q-register with the " .
?MEU	Macro Ending with ↑ A command macro being executed from a Q-register ends with the ↑ character. This is an incomplete command. The ↑ command takes a single character text argument that must be in the Q-register with the ↑ .
?MIQ	Macro Ending with <char> A command macro being executed from a Q-register ends with the character "<char>". This is an incomplete command. The <char> command takes a single character text argument to name the Q-register to which it applies. This argument must be in the same macro as the <char> command itself.
?MLA	Missing < There is a right angle bracket not matched by a left angle bracket somewhere to its left. (Note: an iteration in a macro stored in a Q-register must be complete within the Q-register.)
?MLP	Missing (Command string contains a right parenthesis that is not matched by a corresponding left parenthesis.

Table A-1 (Cont)
TECO Error Messages

?MRA	Missing > In attempting to exit from an iteration field with a ; command (or to skip over an iteration field with a 0 argument) no > command closing the iteration can be found. Note: iteration fields must be complete within a single macro level.
?MRP	Missing) The command string contains, within an iteration field, a parenthetical expression that is not closed by a right parenthesis.
?MUU	Macro Ending with ↑↑ A command macro being executed from a Q-register ends with control-↑ or ↑↑. This is an incomplete command. The ↑↑ command takes a single character text argument that must be in the Q-register with the ↑↑.
?NAE	No Argument Before = The command n= or n== causes the value n to be typed. The = command must be preceded by either a specific numeric argument or a command that returns a numeric value.
?NAI	No Altmode after nI Unless the EO value has been set to 1, the numeric insert command nI must be immediately followed by altmode.
?NAQ	No Argument Before " The " command must be preceded by a single numeric argument on which the decision to execute the following commands or skip to the matching ' is based.
?NAU	No Argument Before U The command nUi stores the value n in Q-register i. The U command must be preceded by either a specific numeric argument or a command that returns a numeric value.
?NCS	No Command String Seen Prior to *i The *i command saves the precedirg command string in Q-register i. In this case no command string has previously been given.
?NFI	No File for Input Before issuing an input command (Y or A) it is necessary to open an input file by use of an ER, EB, or TECO command.
?NFO	No File for Output Before giving an output command (PW, P, N, EX, or EG) it is necessary to open an output file by use of an EB, EW, EZ, MAKE, or TECO command.
?NTQ	No Text in Q-register x Q-register x, specified by a G or M command, does not contain text.
?OCT	"8" or "9" in Octal Digit String In a digit string preceded by ↑0, only the octal digits 0-7 may be used.

Table A-1 (Cont)
TECO Error Messages

?ODV	Output Device dev Not Available Initialization failure. Unable to initialize the device dev for output. Either the device is being used by someone else right now, or it is write locked, or else it does not exist in the system.
?OLR	Cannot Lookup Output File dev:filnam.ext to Rename It Failure in rename process at close of editing job initiated by an EB command or a TECO command. The special LOOKUP on the output file filnam.ext required for DECtape in order to rename the file to "filnam.ext" has failed. The original input file filnam.ext has been renamed "filnam.BAK", but the output file is closed with the name "nnnTEC.TMP", where nnn is the user's job number. The LOOKUP UUO error code is nn.
?OUT-nn0000	Output Error nn0000 - Output File nnnTEC. TMP Closed An error on the output device is fatal. The output file is closed at the end of the last data that was successfully output. It has the filename "nnnTEC.TMP", where nnn is the user's job number. See Section 4.3 for a recovery technique. The output device status word error flags are nn0000. (Note: This number represents the I/O status word (rh) with bits 22-35 masked out.) (000000 -- end of tape). (040000 -- block number too large: device full or quota exceeded). (100000 -- parity or checksum error). (140000 -- block number too large and parity error). (200000 -- device error, data missed). (240000 -- block number too large and device errror). (300000 -- parity error and device error). (340000 -- block number too large, parity error, and device error). (400000 -- improper mode or device write locked). (440000 -- block number too large and improper mode). (500000 -- parity error and improper mode). (540000 -- block number too large, parity error, and improper mode). (600000 -- device error and improper mode). (640000 -- block number too large, device error, and improper mode). (700000 -- parity error, device error, and improper mode). (740000 -- block number too large, parity error, device error, and improper mode).
?PAR	Confused Use of Parentheses A string of the form (...<...) has been encountered. Parentheses should be used only to enclose combinations of numefic arguments. An iteration may not be opened and not closed between a left and right parenthesis.
?POP	Attempt to Move Pointer Off Page with J, C, R, or D The argument specified with a J, C, R, or D command must point to a position within the current size of the buffer, i.e., between 0 and Z, inclusive.

Table A-1 (Cont)
TECO Error Messages

?PPN	Illegal Character "<char>" in Project-programmer Number Project-programmer numbers in file specifications must be given in the form [m,n] where m and n are octal digit strings separated by a comma. No characters other than the ones specified may appear between the enclosing brackets.
?RNO	Cannot Rename Output File nnnTEC.TMP Failure in rename process at close of editing job initiated by an EB command or a TECO command. The attempt to rename the output file nnnTEC.TMP to the name "filnam.ext" originally specified in the EB or TECO command has failed. The original input file filnam.ext has been renamed "filnam.BAK", but the output file is closed with the name "nnnTEC.TMP", where nnn is the user's job number. The RENAME UUO error code is nn.
?SAL	Second Argument Less Than First In a two-argument command, the first argument must be less than or equal to the second.
?SNA	Initial Search with No Argument A search command with null argument has been given, but there was no preceding search command from which the argument could be taken.
?SNI	; Not in an Iteration The semicolon command may be used only with a string of commands enclosed by angle brackets, i.e., in an iteration field.
?SRH	Cannot Find "<text>" A search command not preceded by a colon modifier and not within an iteration has failed to find the specified character string "<text>". After an S search fails the pointer is left positioned at the beginning of the buffer. After an N or ← search fails the last page of the input file has been input and, in the case of N, output, and the buffer cleared. Note that when this message occurs, the text string printed includes all control-character commands included in the search argument.
?STC	Search String Too Long The maximum length of a search string is 80 characters including all string control commands and their arguments.
?STL	Search String too Long The maximum length of a search string is 36 character positions, not counting extra characters required to specify a single position.
?TAG	Missing Tag !xxx! The tag !xxx! specified by an O command cannot be found. This tag must be in the same macro level as the O command referencing it.
?TAL	Two Arguments with L The L command takes at most one numeric argument, namely, the number of lines over which the buffer pointer is to be moved.

Table A-1 (Cont)
TECO Error Messages

?TTY	Illegal TTY I-O Device A terminal may be specified as an input-output device in an ER, EW, EZ, or MAKE command only if it is not being used to control an attached job, the user's own terminal included.
?UCA	Unterminated ↑A Command A ↑A message type-out command has been given, but there is no corresponding ↑A to mark the end of the message. ↑A commands must be complete within a single command level.
?UFS	Macro Ending with Unterminated File Selection Command The last command in a command macro being executed from a Q-register is a file selection command (ER, EW, EB, or EZ) with no altmode to mark the end of the file specifications. The file selection command must be complete within the Q-register.
?UIN	Unterminated Insert Command An insert command (possibly an @ insert command) has been given without terminating the text argument at the same macro level.
?UIS	Undefined I/O Switch "/xxx" The switch "/xxx" is not defined with either input or output file selection commands. The only switches currently defined for input or output file selection commands are /GENLSN and /SUPLSN.
?USR	Unterminated Search Command A search command (possibly an @ search command) has been given without terminating the text argument at the same macro level.
?UTG	Unterminated Tag A command string tag has been indicated by a ! command, but there is no corresponding ! to mark the end of the tag. Tags must be complete within a single command level.
?UUO	Illegal UUO Internal error. The illegal instruction <lh,rh> has been encountered at address nnnnnn. This is caused by either a TECO bug or a monitor bug. Please give printout to your system manager, or submit it to DEC with an SPR.

Appendix B
ASCII Characters

Table B-1
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Comment and Section Reference
Null or Control-Shift-P	↑@	000	0	Ignored on input. Ignored on type-in. nl $ insert only.
Control-A	↑A	001	1	TECO command (Section 3.17).
Control-B	↑B	002	2	Monitor command (Section 3.18). A special character (Section 2.2).
Control-C	↑C	003	3	Monitor command (Section 3.10). A special character (Section 2.2). nl $ insert only. Echoes as ↑C-carriage return-line feed.
Control-D	↑D	004	4	TECO command (Section 3.17).
Control-E	↑E	005	5	TECO command (Sections 3.11 and 3.16).
Control-F	↑F	006	6	TECO command (Section 3.16). Monitor command (Section 3.18). A special character (Section 2.2).
Bell	↑G	007	7	Echoes and prints as a single bell ring and ↑G. Double ↑G and ↑G␣ are TECO commands (Section 5.1) and special characters (Section 2.2).
Backspace	↑H	010	8	TECO command (Section 3.16). Prints as ↑H.
Tab	⇥	011	9	TECO command (Section 3.8).
Line Feed	↓	012	10	Ignored in command strings except as a text argument (Section 3.18). The symbol ↓ is used only to represent an explicitly-typed line feed. It is not used for the line feed that the monitor

Table B-1 (Cont)
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Comment and Section Reference
Line Feed (Cont)				generates when a carriage return is typed. In data, line feed defines the end of a line (Section 2.3).
Vertical Tab	VT	013	11	In data, vertical tab defines the end of a line (Section 2.3).
Form Feed	FORM	014	12	TECO command (Section 3.6). In data, form feed defines the end of a page (Section 2.3).
Carriage Return	↵	015	13	Ignored in command strings except as a text argument. (Section 3.18). When this character is typed the monitor automatically generates a line feed following it.
Control-N	↑N	016	14	TECO command (Section 3.11).
Control-O	↑O	017	15	Monitor command (Section 3.6). A special character (Section 2.2). nl $ insert only. Echoes as ↑O-carriage return-line feed.
Control-P	↑P	020	16	Monitor command (Section 3.18). A special character (Section 2.2).
Control-Q	↑Q	021	17	TECO command (Section 3.11).
Control-R	↑R	022	18	TECO command (Sections 3.8 and 3.11).
Control-S	↑S	023	19	TECO command (Section 3.11).
Control-T	↑T	024	20	Two different uses as TECO commands (Sections 3.8, 3.11, 3.16).
Control-U	↑U	025	21	TECO command (Section 5.1). A special character (Section 2.2). nl $ insert only. Echoes as ↑U carriage return-line feed.
Control-V	↑V	026	22	TECO command (Sections 3.8 and 3.11).
Control-W	↑W	027	23	TECO command (Sections 3.8 and 3.11).
Control-X	↑X	030	24	Two different uses as TECO commands (Section 3.11).
Control-Y	↑Y	031	25	
Control-Z	↑Z	032	26	TECO command (Section 3.10). Echoes as ↑Z-carriage return-line feed. Used as end-of-file signal when doing data input from a TTY.

Table B-1 (Cont)
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Comment and Section Reference
Altmode or (Control-Shift-K)	$	033	27	Alphanumeric argument terminator (Section 2.7). A special character (Section 2.2). Echoes and prints as $. Two successive altmodes are used to terminate a command string (Section 2.6).
Control-Shift-L	↑\	034	28	TECO command (Section 3.11).
Control-Shift-M	↑]	035	29	
Control-Shift-N	↑↑	036	30	Two different uses as TECO commands (Sections 3.8, 3.11, 3.16).
Control-Shift-O	←	037	31	
Space	␣	040	32	TECO command (Section 2.7). Ignored in command strings except as a text argument or when used instead of + with numeric arguments (Section 3.18).
!		041	33	TECO command (Section 3.13).
"		042	34	Used as a prefix for a whole class of TECO commands (Section 3.13).
#		043	35	TECO command (Section 2.7).
$		044	36	
%		045	37	TECO command (Section 3.14).
&		046	38	TECO command (Section 2.7).
'		047	39	TECO command (Section 3.13).
(		050	40	TECO command (Section 2.7).
)		051	41	TECO command (Section 2.7).
*		052	42	Two different uses as TECO commands (Sections 2.7 and 2.14).
+		053	43	TECO command (Section 2.7).
,		054	44	TECO command (Section 2.7).
-		055	45	TECO command (Section 2.7).
.		056	46	TECO command (Sections 3.2 and 3.4).
/		057	47	Two different uses as TECO commands (Sections 2.7 and 5.2).

Table B-1 (Cont)
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Comment and Section Reference
0		060	48	
1		061	49	
2		062	50	
3		063	51	
4		064	52	
5		065	53	
6		066	54	
7		067	55	
8		070	56	
9		071	57	
:		072	58	TECO command (Section 3.11). Device name delimiter (Section 3.2).
;		073	59	TECO command (Section 3.12).
<		074	60	TECO command (Section 3.12).
=		075	61	TECO command (Section 3.15).
>		076	62	TECO command (Section 3.12).
?		077	63	Two different uses as TECO commands (Sections 3.17 and 5.2).
@		100	64	TECO command (Sections 3.8 and 3.11).
A		101	65	Two different uses as TECO commands (Sections 3.3 and 3.15).
B		102	66	TECO command (Section 3.4). Also used in the EB command (Section 3.2).
C		103	67	TECO command (Section 3.5). Also used in the "C command (Section 3.13).
D		104	68	TECO command (Section 3.7).
E		105	69	Used as a prefix for many TECO commands: EB, EF, EG, EH, EM, EO, ER, ES, ET, EU, EW, EX, EZ (Sections 3.2, 3.6, 3.9, 3.10). Also used in the "E command (Section 3.13).

Table B-1 (Cont)
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Comment and Section Reference
F		106	70	Used in the EF commands (Section 3.9). Also in FS and FN commands (Section 3.11).
G		107	71	TECO command (Section 3.14). Also used in the EG command (Section 3.10) and "G command (Section 3.13).
H		110	72	TECO command (Section 3.4).
I		111	73	TECO command (Section 3.8).
J		112	74	TECO command (Section 3.5).
K		113	75	TECO command (Section 3.7).
L		114	76	TECO command (Section 3.5). Also used in the "L command (Section 3.13).
M		115	77	TECO command (Section 3.14). Also used in the EM command (Section 3.2).
N		116	78	TECO command (Section 3.11). Also used in the "N command (Section 3.13). Also used in FN command (Section 3.11).
O		117	79	TECO command (Section 3.13).
P		120	80	TECO command (Section 3.9).
Q		121	81	TECO command (Section 3.14).
R		122	82	TECO command (Section 3.5). Also used in the ER command (Section 3.2).
S		123	83	TECO command (Section 3.11). Also used in ES and FS commands (Section 3.11).
T		124	84	TECO command (Section 3.6). Also used in the ET command (Sections 3.6 and 3.16).
U		125	85	TECO command (Section 3.14).
V		126	86	
W		127	87	Used in the EW command (Section 3.2) and the PW command (Section 3.4). Otherwise ignored in command strings.
X		130	88	TECO command (Section 3.14). Also used in the EX command (Section 3.10).

Table B-1 (Cont)
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Command and Section Reference
Y		131	89	TECO command (Section 3.3).
Z		132	90	TECO command (Section 3.4). Also used in the EZ command (Section 3.2).
[		133	91	TECO command (Sections 3.2 and 3.14).
\		134	92	Two different uses as TECO commands (Sections 3.8 and 3.14).
]		135	93	TECO command (Sections 3.2 and 3.14).
↑ or ∧	↑	136	94	When used as a command, indicates that the next character is to be interpreted as a control character.
← or _	←	137	95	TECO command (Section 3.11).
`		140	96	
a		141	97	Equivalent to A in command strings.
b		142	98.	Equivalent to B in command strings.
c		143	99	Equivalent to C in command strings.
d		144	100	Equivalent to D in command strings.
e		145	101	Equivalent to E in command strings.
f		146	102	Equivalent to F in command strings.
g		147	103	Equivalent to G in command strings.
h		150	104	Equivalent to H in command strings.
i		151	105	Equivalent to I in command strings.
j		152	106	Equivalent to J in command strings.
k		153	107	Equivalent to K in command strings.
l		154	108	Equivalent to L in command strings.
m		155	109	Equivalent to M in command strings.
n		156	110	Equivalent to N in command strings.
o		157	111	Equivalent to O in command strings.
p		160	112	Equivalent to P in command strings.
q		161	113	Equivalent to Q in command strings.

Table B-1 (Cont)
ASCII Characters

Character	Manual Symbol	Octal	Decimal	Comment and Section Reference
r		162	114	Equivalent to R in command strings.
s		163	115	Equivalent to S in command strings.
t		164	116	Equivalent to T in command strings.
u		165	117	Equivalent to U in command strings.
v		166	118	Equivalent to V in command strings.
w		167	119	Equivalent to W in command strings.
x		170	120	Equivalent to X in command strings.
y		171	121	Equivalent to Y in command strings.
z		172	122	Equivalent to Z in command strings.
{		173	123	
|		174	124	
}		175	125	Converted to altmode (033) when read from TTY unless user has specified TTY LC mode. Equivalent to altmode (033) when executing commands or being typed as text if the EO value has been set to 1.
		176	126	Converted to altmode (033) when read from TTY unless user has specified TTY LC mode. Equivalent to altmode (033) when executing commands or being typed as text if the EO value has been set to 1.
Rubout or Delete	RO	177	127	TECO command (Section 5.1). A special character (Section 2.2). nl$ insert only. Does not print. Echoes as the character being erased.

Appendix C
Summary of Commands

C.1 INITIALIZATION AND FILE SELECTION

Table C-1
Command Description

Command	Function	Reference
	INITIALIZATION AND FILE SELECTION	
dev:filnam.ext[proj,prog]	File specifications	(Section 3.2)
ERfilespecification $	Select file for input.	(Section 3.2)
nEM	Position magnetic tape	(Section 3.2)
EWfilespecification $	Select file for output.	(Section 3.2)
EZfilespecification $	Zero directory and select file for output.	(Section 3.2)
EBfilespecification $	Select file for input and output, with back-up file protection.	(Section 3.2)
MAKEfilespec↵	Equivalent to EWfilnam.ext $.	(Section 3.1)
TECOfilespec↵	Equivalent to EBfilnam.ext $ Y.	(Section 3.1)
/GENLSN	Used with EW or EB to cause line sequence numbers to be generated.	(Section 3.2)
/SUPLSN	Used with ER, EB, or EW to suppress line sequence numbers.	(Section 3.2)
	INPUT	
Y	Clear Buffer and input one page.	(Section 3.3)
A	Input one page and append to current buffer contents.	(Section 3.3)
	BUFFER POSITIONS	
B	Before first character; 0.	(Section 3.4)
.	Current pointer position; number of characters to the left of the pointer.	(Section 3.4)

Table C-1 (Cont)
Command Description

Command	Function	Reference
Z	End of the buffer; number of characters in the buffer.	(Section 3.4)
m,n	m+1st through nth characters in the buffer.	(Section 2.7)
H	Entire buffer; B,Z.	(Section 3.4)
	ARGUMENT OPERATORS	
m+n	Add.	(Section 2.7)
m␣n	Add.	(Section 2.7)
m-n	Subtract.	(Section 2.7)
m*m	Multiply.	(Section 2.7)
m/n	Divide and truncate.	(Section 2.7)
m&n	Logical AND.	(Section 2.7)
m#n	Logical OR.	(Section 2.7)
( )	Perform enclosed operations first.	(Section 2.7)
↑O	Accept number in octal radix.	(Section 2.7)
	POINTER POSITIONING	
nJ	Move pointer to position between nth and n+1st characters.	(Section 3.5)
nC	Advance pointer n positions.	(Section 3.5)
nR	Move pointer back n positions. Equivalent to -nC.	(Section 3.5)
nL	Move pointer to beginning of nth line from current pointer position.	(Section 3.5)
	TYPE-OUT	
nT	Type all text in the buffer from the current pointer position to the beginning of the nth line from the pointer position.	(Section 3.6)
m,nT	Type the m+1st through the nth characters.	(Section 3.6)
n=	Type the decimal integer n.	(Section 3.15)
n==	Type the octal integer n.	(Section 3.15)
1ET	Change typeout mode so that no substitutions are made for nonprinting characters.	(Section 3.6)
0ET	Restore typeout mode to normal.	(Section 3.6)

Table C-1 (Cont)
Command Description

Command	Function	Reference
0EU	Flag lower case characters on typeout (standard).	(Section 3.6)
1EU	Flag upper case characters on typeout.	(Section 3.6)
-1EU	No case flagging on typeout.	(Section 3.6)
-1ES	Set automatic typeout after searches.	(Section 3.11)
nES(n>0)	Set automatic typeout after searches and include a character to indicate the position of the pointer.	(Section 3.11)
0ES	Set to no automatic typeout after searches.	(Section 3.11)
↑A message ↑A	Type the message enclosed.	(Section 3.17)
↑L or form feed	Type a form feed.	(Section 3.6)
↑O	Inhibit typeout.	(Section 3.6)
	DELETION	
nD	Delete the n characters following the pointer position.	(Section 3.7)
-nD	Delete the n characters preceding the pointer position.	(Section 3.7)
nK	Delete all characters in the buffer from current pointer position to the beginning of the nth line from the pointer position.	(Section 3.7)
m,nK	Delete the m+1st through the nth characters.	(Section 3.7)
	INSERTION	
Itext $	Insert the text delimited by I and altmode.	(Section 3.8)
nI$	Insert the character with ASCII value n (decimal).	(Section 3.8)
@I/TEXT/	Insert the text delimited by the arbitrary character following I.	(Section 3.8)
n\	Insert the ASCII representation of the decimal integer n.	(Section 3.8)
↑V	Translate to lower case.	(Section 3.8)
↑W	Translate to upper case.	(Section 3.8)
↑↑	When used inside text arguments, this means translate special characters @, [, \,], ↑, ← to "lower case" range.	(Section 3.8)

Table C-1 (Cont)
Command Description

Command	Function	Reference
	INSERTION (Cont)	
↑R	Accept next character as text.	(Section 3.8)
↑T	Used inside text arguments to cause all control characters except ↑R , ↑T , and altmode to be taken as text. Nullified by a second ↑T .	(Section 3.8)
	OUTPUT AND EXIT	
PW	Output the current page and append a form feed character to it.	(Section 3.9)
nP	Output the current page, clear the buffer, and read in the next page. Continue this process until the nth page from the current page has been input.	(Section 3.9)
m,nP	Output the m+1st through the nth characters. Do not append a form feed character, and do not change the buffer.	(Section 3.9)
EF	Close the output file.	(Section 3.9)
↑Z or ↑Z	Close the output file and exit to the monitor.	(Section 3.10)
↑C	Exit to the monitor.	(Section 3.10)
EX	Output the remainder of the file, close the output file, and then exit to the monitor.	(Section 3.10)
EG	Output the remainder of the file, close and then re-execute the last compile-class command that was typed.	(Section 3.10)
	SEARCH	
nStext $	Search for the nth occurrence (following the pointer) of the text delimited by S and altmode, but do not go beyond the end of the current page.	(Section 3.11)
nFStext $ text $	Search for the nth occurrence (following the pointer) of the first text string and replace it with the second text string. Do not go beyond the end of the current page.	(Section 3.11)
nNtext $	Equivalent to nStext $ except that if the text is not found on the current page, pages are input and output until it is found.	(Section 3.11)

Table C-1 (Cont)
Command Description

Command	Function	Reference
	SEARCH (Cont)	
nFNtext $ text $	Equivalent to nFStext $ text $ except that if the text is not found on the current page, pages are input and output until it is found.	(Section 3.11)
n← text $	Equivalent to nNtext $ except that it does input only, no output.	(Section 3.11)
:nStext $	Equivalent to nStext $ except that it returns a value of -1 if the search succeeds or 0 if it fails instead of an error message. The : command can also be used with FS,N, FN, and ←.	(Section 3.11)
@nS/text/	Equivalent to nStext $ except that the text is delimited by the arbitrary character following the S. The @ command may also be used with FS,N, FN, and ←.	(Section 3.11)
0 ↑X or 0↑X	Reset search mode to accept either case.	(Section 3.11)
n ↑X or n↑X (n≠0)	Set search mode to "exact" mode.	(Section 3.11)
↑V	Translate to lower case.	(Section 3.11)
↑W	Translate to upper case.	(Section 3.11)
↑↑	When used inside text arguments, this means translate special characters @, [, \ ,], ↑, ← to "lower case" range.	(Section 3.11)
↑R	Accept next character as text.	(Section 3.11)
↑T	Used inside text arguments to cause all control characters except ↑R , ↑T , and altmode to be taken as text. Nullified by a second ↑T.	(Section 3.11)
↑\	Used inside search argments to indicate accept either case for following characters. Nullified by a second ↑\ .	(Section 3.11)
↑X	When used inside a text argument, accept any character at this position in the search string.	(Section 3.11)
↑S	Accept any separator character at this position.	(Section 3.11)
↑N a	Accept any character except the arbitrary character a following ↑N .	(Section 3.11)

Table C-1 (Cont)
Command Description

Command	Function	Reference
	SEARCH (Cont)	
↑Q	Take the next character in the search string literally, even if it is a control character.	(Section 3.11)
↑E A	Accept any alphabetic character as a match.	(Section 3.11)
↑E V	Accept any lower case alphabetic character as a match.	(Section 3.11)
↑E W	Accept any upper case alphabetic character as a match.	
↑E D	Accept any digit as a match.	(Section 3.11)
↑E L	Accept any end-of-line character as a match.	(Section 3.11)
↑E S	Accept any string of spaces and/or tabs as a match.	(Section 3.11)
↑E <nnn>	Accept the ASCII character whose octal value is nnn as a match.	(Section 3.11)
↑E [a,b,c...]	Accept any one of the characters in the brackets as a match.	(Section 3.11)
	ITERATION AND FLOW CONTROL	
n< >	Perform the enclosed command string n times.	(Section 3.12)
n;	If n=0, jump out of the current iteration field.	(Section 3.12)
;	Jump out of the current iteration field, if the last search executed failed.	(Section 3.12)
!tag!	Define a position in the command string with the name "tag".	(Section 3.13)
Otag $	Jump to the position defined by Itag!.	(Section 3.13)
n"Ecommands'	If n=0, execute the commands specified between "E and '; otherwise, skip to the '.	(Section 3.13)
n"Ncommands'	If n≠0, execute the enclosed commands.	(Section 3.13)
n"Lcommands'	If n<0, execute the enclosed commands.	(Section 3.13)
n"Gcommands'	If n>0, execute the enclosed commands.	(Section 3.13)
n-1"Lcommands'	If n≤0, execute the enclosed commands.	(Section 3.13)
n+1"Gcommands'	If n≥0, execute the enclosed commands.	(Section 3.13)
n"Ccommands'	If n is the ASCII value (decimal) of a symbol constituent character, execute the enclosed commands.	(Section 3.13)

Table C-1 (Cont)
Command Description

Command	Function	Reference
	ITERATION AND FLOW CONTROL (Cont)	
n"Dcommands'	If n is a digit execute the enclosed commands.	(Section 3.13)
n"Acommands'	If n is alphabetic, execute the enclosed commands.	(Section 3.13)
n"Vcommands'	If n is lower case alphabetic, execute the enclosed commands.	(Section 3.13)
n"Wcommands'	If n is upper case alphabetic, execute the enclosed commands.	(Section 3.13)
n"Tcommands'	If n is true, execute the enclosed commands.	(Section 3.13)
n"Fcommands'	If n is false, execute the enclosed commands.	(Section 3.13)
n"Scommands'	If n is "successful", execute the enclosed commands.	(Section 3.13)
n"Ucommands'	If n is "unsuccessful", execute the enclosed commands.	(Section 3.13)
	Q-REGISTER	
nUi	Store the integer n in Q-register i.	(Section 3.14)
Qi	Equal to the value stored in Q-register i.	(Section 3.14)
%i	Increment the value in Q-register i by 1 and return this value.	(Section 3.14)
nXi	Store, in Q-register i, all characters from the current pointer position to the beginning of the nth line from the pointer.	(Section 3.14)
m,nXi	Store the m+1st through nth characters in Q-register i.	(Section 3.14)
Gi	Place the text in Q-register i at the current pointer position.	(Section 3.14)
Mi	Execute the text in Q-register i as a command string.	(Section 3.14)
[i	Push the current contents of Q-register i onto the Q-register pushdown list.	(Section 3.14)
]i	Pop the last stored entry from the Q-register pushdown list into Q-register i.	(Section 3.14)
*i	(As first command in a string.) Save the preceding command string in Q-register i.	(Section 3.14)

Table C-1 (Cont)
Command Description

Command	Function	Reference
	SPECIAL NUMERIC VALUES	
1A	The ASCII value (decimal) of the character following the pointer.	(Section 3.16)
$\textcircled{\uparrow E}$ or ↑E	The form feed flag. Equals 0 if no form feed character was read on the last input, -1 otherwise.	(Section 3.16)
$\textcircled{\uparrow N}$ or ↑N	The end-of-file flag; equals -1 if end of input file seen on last input. Otherwise equals 0.	(Section 3.16)
↑F or $\textcircled{\uparrow F}$	Decimal value of the console data switches.	(Section 3.16)
$\textcircled{\uparrow H}$ or ↑H	The time of day in 60th's of a second.	(Section 3.16)
ET	The value of the type-out mode switch. Equals 0 for normal type-out, -1 otherwise.	(Section 3.16)
$\textcircled{\uparrow X}$ or ↑X	Value of the search mode flag. (0=either case mode, -1= exact mode.)	(Section 3.11)
EU	The value of the EU flag. +1 = flag upper case characters. 0 = flag lower case characters, -1 = no case flagging on typeout.	(Section 3.6)
EO	The value of the EO flag. 1= version 21A, 2= versions 22 and 23.	(Section 3.17)
EH	The value of the EH flag. 1= code only, 2= code plus one line, 3= all of error message.	(Section 5.2)
$\textcircled{\uparrow\uparrow}$ x or ↑↑x	Equivalent to the ASCII value (in decimal) of the arbitrary character x following ↑↑.	(Section 3.16)
\	Equivalent to the decimal value of the digit string following the pointer.	(Section 3.16)
$\textcircled{\uparrow T}$ or ↑T	Stop command execution and then take on the ASCII value (in decimal) of the character typed in by the user.	(Section 3.16)
	AIDS	
/	When used after an error message, this causes a more detailed explanation of the error to be typed.	(Section 5.2)
*i	When used at the beginning of a command string, this causes the entire command string (with one of the two concluding altmodes) to be moved into Q-register i.	(Section 5.2)

Table C-1 (Cont)
Command Description

	AIDS (Cont)	
?	When used after an error message, this causes the offending command to be typed out (with a few of the commands preceding it).	(Section 5.2)
?	Enter trace mode. A second ? command takes TECO out of trace mode.	(Section 3.17)
RO	Erase last character typed in the command string.	(Section 5.1)
↑G ↑G	Erase the entire command string.	(Section 5.1)
↑U	Erase everything typed in back to the last break character.	(Section 5.1)
↑G ␣	Retype current line of command string.	(Section 5.1)
0EO	Restore the EO value to standard.	(Section 3.17)
nEO (n≠0)	Set the EO value to n.	(Section 3.17)
1EH	Type only code part of error messages.	(Section 5.2)
2EH	Type error code plus one line.	(Section 5.2)
3EH	Type all three parts of error.	(Section 5.2)
0EH	Equivalent to 2EH.	(Section 5.2)