COVER SHEET FOR TECHNICAL MEMORANDUM

TITLE QED Text Editor

MM 70 - 1371 - 2
1373 - 3

CASE CHARGED — 70107-002

DATE — June 22, 1970

FILING CASES — 70107-002

AUTHOR — D. M. Ritchie
K. L. Thompson

Ext. 3770 (MH)
2394 (MH)

FILING SUBJECTS — Text editing
Text manipulation
String manipulation

ABSTRACT

QED is an interactive text editing program available under GE-TSS on the GE-635 computer. It is much more powerful than most previous editing systems.

QED deals with both GE-TSS ASCII and card image files. It offers the ability to rearrange arbitrary blocks of text, to execute user-specified and pre-existing macro command sequences, and a very general text location and replacement capability.

This memorandum provides a complete reference manual for QED.

Text - 17 pages
References
Appendix - 6 pages

SEE REVERSE SIDE FOR DISTRIBUTION LIST

Bell Laboratories

subject: QED Text Editor - Case 70107-002

date: June 22, 1970

from: D. M. Ritchie
K. L. Thompson
MM-70-1373-3
1371-2

MEMORANDUM FOR FILE

Introduction

QED is an interactive text editor available under GE-TSS on the GE 635. QED has been characterized as hard to learn but easy to use. In any event, it is much more powerful than most editors; its text manipulation facilities approach those of SNOBOL.

This memorandum is a reference manual for QED. It is a descendant of [1], and contains a complete description of the editor as well as an Appendix with a short reference guide and several examples.

Since this document attempts to be as complete as possible, it is not easy to read. QED novices are advised to use B. W. Kernighan's excellent "A Tutorial Introduction to the QED Text Editor" [2]. The subset of QED described therein is roughly equivalent in power to the GE-TSS EDIT subsystem, and will be enough to satisfy many users' editing needs. Most remaining features of QED are explained and exemplified in the same author's "A Guide to Advanced Use of the QED Text Editor" [3].

History of QED

The original QED was implemented at the University of California, Berkeley [4]. Substantially redesigned versions were written by the second author (KLT) for the CTSS system at MIT [1] and in BCPL for MULTICS. The latter version has also been available under GE-TSS using I/O routines supplied by A. W. Winikoff.

This version of QED

The present incarnation of QED was implemented in GMAP by the first author (DMR). It offers noticeable improvements in speed, program size, and text packing density over the BCPL version, of which it is a direct descendant. New facilities include a redesigned Global command and a numerical capability.

Description of QED

Unlike many editors, QED stores all the text it is working on in core. This gives rapid access to all the text, even if it is addressed randomly, but it also means that only a bounded amount of text can be edited at one time. This amount varies with several factors, but with the usual TSS core limit (32K) QED can process 45- or 50-block files. (This is equivalent to about 30 pages of text like those in this manual.) However, both cost and response time increase with core usage, so very large files should be avoided.

QED is reasonably economical of computer time. For ordinary editing, connect charges always exceed those for CPU and I/O; complicated text processing, especially on large files, can become expensive.

All text in QED is stored in buffers. At any time there is a current buffer to which most commands implicitly refer. In each buffer there is a current line which is changed by most editor commands. The current line is taken as a default address for several commands. The current buffer initially is buffer "0". Text enters a buffer only by explicit means: for example, it may be typed in or read from a file. Text is never filed away automatically; instead it must be written explicitly on a named file. QED does not use the "*SRC" file, although it may be read and written.

The syntax of QED is concise and very regular. There are a number of commands each of which is one character long. Most commands have names intended to be reminiscent of their function. Each command is preceded by zero or more addresses specifying lines in the current buffer. Some commands are followed by parameters as well. In general any number of commands may be typed on a line; however, no command may follow the G, L, O, Q, R, V, or W commands on the same line.

QED uses the ASCII character set internally. It can read and write both GE-TSS ASCII and GE-Hollerith card image files.

Syntax conventions in this manual

Double and single quotes are used with their usual meaning except in obvious instances, for example in the discussion of the `"' (quote) command. Angle brackets are used to enclose the names of QED syntactic categories: "<regexp>" stands for a regular expression; in the next paragraph and elsewhere, "<nl>" stands for the ASCII newline character. Angle brackets are totally without special meaning to QED itself.

The character <nl>

The ASCII <nl> character ("newline") separates lines. It may be typed by means of the "line space" key on the 37 Teletype. However, the character <cr> (carriage return) is translated to <nl> when read from a file or the typewriter keyboard. Since the action character for GE-TSS is <cr>, it will be used most often. Users of <nl> should note, however, that at most 80 characters can be input between <cr>s and that the "line delete" (control X) sequence destroys everything up to the last <cr>, including intervening <nl>s.

Regular expressions

Regular expressions are a unique feature of QED. A regular expression is a pattern which specifies a set of strings of characters; it is said to match certain strings. Regular expressions are used in QED to specify text which is to be replaced by other text and in searching for lines satisfying some condition. Regular expressions are defined rigorously as follows.

a) An ordinary character is a regular expression which matches that character.

b) "^" is a regular expression which matches the null character at the beginning of a line.

c) "$" is a regular expression which matches the null character before the character <nl> (usually at the end of a line).

d) "." is a regular expression which matches any character except <nl>.

e) "[<string>]" is a regular expression which matches any of the characters in the <string> and no others.

f) "[^<string>]" is a regular expression which matches any character but <nl> and the characters of the <string>.

g) A regular expression followed by "*" is a regular expression which matches any number (including zero) of adjacent occurrences of the text matched by the regular expression.

h) Two adjacent regular expressions form a regular expression which matches adjacent occurrences of the text matched by the regular expressions.

i) Two regular expressions separated by "|" form a regular expression which matches the text matched by either of the regular expressions.

j) A regular expression in parentheses is a regular expression which matches the same text as the original regular expression. Parentheses are used to alter the order of evaluation implied by g), h), and i): "a(b|c)d" will match "abd" or "acd", while "ab|cd" matches "ab" or "cd".

k) If "<regexp>" is a regular expression, "{<regexp>}x" is a regular expression, where x is any character. This regular expression matches the same things as <regexp>; it has certain side effects as explained under the Substitute command.

l) If <rexname> is the name of a regular expression named by the E command (below), then "\E<rexname>" is a regular expression which matches the same things as the regular expression specified in the E command. More discussion is presented under the E command.

m) The null regular expression standing alone is equivalent to the last regular expression encountered. Initially the null regular expression is undefined; it also becomes undefined after an erroneous regular expression and after use of the E command.

n) Nothing else is a regular expression.

o) No regular expression will match text spread across more than one line.

In subsequent discussion, "<regexp>" will denote a regular expression.

Here are some examples of regular expressions. In each case the expression is bounded by "/".

/abcd/ matches "abcd" anywhere in a line.

/ab|cd/ matches "ab" or "cd" anywhere in a line.

/ab*c/ matches "ac", "abc", "abbc", "abbbc", ...

/^begin/ matches "begin" at the beginning of a line.

/end$/ matches "end" at the end of a line.

/^begin.*end$/ matches any line beginning with "begin" and ending with "end".

/^$/ matches an empty line.

/abc[1234567890]/ matches "abc" followed by a digit.

/abc[^1234567890]/ matches "abc" followed by a non-digit.

Buffer names

Text may be stored in any of many buffers. Buffers are named by sequences of 14 or fewer characters not including the character <nl>. A buffer name is represented by "(" followed by the characters comprising the name followed by ")". If the name is exactly one character long, it need not be placed in parentheses. Some examples of buffer names are: "()", "(xyz)", "(x)" (which is the same as "x"), and "(a b c)". Inside buffer names the character "\B" has no special meaning (see "Editor Input" below).

Subsequently, <bufname> will indicate a buffer name.

Register names

There are also many number registers whose names are constructed according to the same rules as buffer names. Number registers may be changed and set by the N command described below. There is no relation between a given number register and the buffer of the same name. In subsequent discussion, <regname> will indicate a register name.

The condition register

Finally, there is a condition register which contains either true or false. Several commands set this register, as discussed below; it can be tested and used to control the flow of QED programs. It should be noted that the condition register is seldom used for ordinary editing tasks.

Text addressing

Lines in the current buffer may be addressed in the following ways.

1) By current line number.
   A decimal number addresses the line at the corresponding position of the current buffer. The first line is numbered 1. The number of a line may change during editing.

2) By absolute line number.
   The character "'" followed by a decimal number is interpreted as an absolute line number. Absolute line numbers are assigned to a buffer after a successful read command, and never change otherwise. New lines created during editing have undefined absolute line numbers. The character "'" not followed by a digit causes a search for the first undefined absolute line number after the current line and cycling to the current line. If there is no line with the specified absolute line number, an error

message is given.

3) By ".".
The value of "." is the current line. This value is changed by most QED commands.

4) By "$".
The value of "$" is the last line in the current buffer. Its line number may change during editing.

5) By context.
The structure "/<regexp>/" causes a search for a line containing text that matches the regular expression. The search begins at the line after the current line and cycles forward to the current line. If the search is successful, the value of "/<regexp>/" is the line found.

The structure "?<regexp>?" is just like "/<regexp>/" except that it searches backwards beginning with the line before the current line.

If commands are being taken from the console, context searches that fail will be noted as errors. A search that fails while commands are being taken from a buffer causes the buffer recursion level to be dropped by one. See also the text directive "\B" below.

6) By additive combinations of 1-5.
Two addresses separated by "+" or "-" also form an address. The value is obvious. Evaluation is done left to right. At no time during evaluation of an address may the address exceed the bounds of the buffer. In unambiguous cases, the "+" need not appear: ".+1" and ".1" are identical in value.

Editor Input

The input to QED is a stream of characters. The following sequences are interpreted as directives to the stream and do not perform an editing function directly.

The sequences are removed from the stream as they perform their function. It should be noted that each of these sequences is regarded by QED as a single character; internally, in fact, they are stored as otherwise unused ASCII control characters. Thus one cannot create a "\C" character from a "C" by substituting in a "\". These characters are effective if and only if they appear as part of the command stream (including the <text> which is part of some commands).

Letters following "\" may be in either case.

\B<bufname>
This sequence causes the input stream to be diverted to the buffer <bufname>. At the end of this buffer input reverts back to its original source. The stream is also switched back if a context search fails. Upon occurrence of any error the input is switched back to the console after printing the remaining contents of the buffer currently being executed. (This is a useful debugging aid for QED programs.) "\B" may be used recursively to any level.

\R
This character causes input to be switched to the console for one line of input. On this line, no stream directive characters are effective. The <nl> character ending the line is discarded.

\C
This character causes the next character to be taken literally, ignoring any special meaning the character may have in the current context.

\N<regname>
This sequence is replaced by the digits corresponding to the contents of number register <regname>. See the N command below.

\E
This character is special only in regular expressions. See the E command and the discussion of regular expressions.

Escape sequences

Certain characters are impossible to generate on certain devices, and the commercial-at sign "@" cannot be input to TSS at all. The escape sequences on the left are accepted in lieu of the characters listed on the right.

\A	@
\(	{
\)	}
\!	|
\\	\
\<	[
\>	]
\'	^
\ddd	7-bit character represented by ddd

The "\\" escape is used because the character "\" followed by any character is taken as a single character. In "\ddd", the d's are octal digits. The characters \000 and \777 have internal meaning and are entered at one's risk.

Text input

There are a number of QED commands that expect literal text as argument. This text must be preceded by a space or a <nl> character, which is discarded. The text itself consists of an arbitrary string of characters terminating in "\F". The "\F" is not itself part of the text but only serves to delimit it. Subsequently, "<text>" will indicate literal text input.

Editor commands

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error; those which require one or two addresses use the last one or two addresses and ignore the extras.

In the list of commands below, the zero-address commands are shown with no addresses.

The commands which require one address all assume a default, which is shown in parentheses. The parentheses are not part of the address, but are used to show that the address given is the default.

Likewise, the two-address commands all assume default addresses given in parentheses. If a two-address command is typed with only one address, that address is used for both those required.

Sequences of two or more addresses are separated either by "," or by ";". In the latter case "." is set to the preceding address before the succeeding address is evaluated. The semicolon is used mostly to control the starting line for context searches.

The following is a list of QED's command repertoire. Commands which are letters may appear in either case.

A) ($)A<text> Append.

The text input is placed after the addressed line. "." is set to the last line input. If no text was input, "." is set to the addressed line.

B) B<bufname> Buffer.

Buffer <bufname> becomes the current buffer. If the buffer already existed, its previous value of "." becomes the current value of ".". Initially the current buffer is "0".

C) (.,.)C<text> Change.

The addressed lines are deleted and replaced by the input text. "." is set to the last line input. If no lines were input, "." is set to the first line not deleted. The addressed lines cannot cross line 0 (circularly).

D) (.,.)D Delete.

The addressed lines are deleted. "." is set to the first line not deleted. The addressed lines cannot cross line 0 (circularly).

E) E<rexname>/<regexp>/ Enter

The E command gives the regular expression <regexp> the name <rexname>. The syntax of <rexname>s is the same as that of <bufname>s and <regname>s. The regular expression <rexname> may be referred to in a regular expression by the construct

\E<rexname>

"\E" has no special significance outside of regular expressions.

The empty regular expression, which normally means the last regular expression encountered, cannot be used for the <regexp> in the E command; also, the empty regular expression becomes undefined after an E command.

Regular expressions named by the E command may contain the "\E" construction. Recursive regular expressions are allowed, but left recursive expressions do not work. (They do not, however, loop.) Examples of the use of E and \E are given in the Appendix.

F) F Facts

The F command causes QED to type out 1) the number of words on QED's "free list", 2) the highest memory location used for text storage, and 3) the current core allocation. When the third number gets near 32K, take care not to exceed the core maximum.

G) (1,$)G/<regexp>/<commands><nl> Global.

The remainder of the line after "/<regexp>/" is placed into a hidden buffer. Each line addressed which matches the <regexp> is marked. Then for each marked line the

commands in the hidden buffer are executed, with "." set at the marked line. The commands G, L, Q, R, V, and W may not be executed within a G command. At the conclusion of the command, "." is set at the point where it was left by the last command executed. Any character but space or <nl> may be used instead of "/" to bound the <regexp>.

If it is desired that the <commands> contain a <nl>, \C should precede the <nl> in order to prevent it from terminating the Global command.

I) (.)I<text> Insert.

Text input is inserted before the addressed line. "." is set to the addressed line. Note that "I" is differs from "A" in its default address as well as in the placement of the <text>.

J) JT<commands> Jump on true
JF<commands> Jump on false

This instruction tests the condition register and "jumps" according to its value. If the condition register is true, the JT command causes the rest of the line to be ignored; conversely, a false condition register causes JF to skip. Otherwise the rest of the line is interpreted normally.

The condition register is set by R, W, S, Y, and T commands.

K) (.,.)K<nl> Sort
(.,.)KB<nl>

In the first form of the K command the addressed lines are sorted in increasing ASCII collating sequence. In the second form descending sequence is used.

L) L <filestring><nl> List.
L6 <filestring><nl>
L<nl>

The file is printed on the console. The value of "." is unchanged. In the first form, the file is ASCII; in the second, it consists of card images. If the "L" is followed directly by <nl>, the file of the name and type last mentioned in a L, R, or W command is listed. See also "File I/O" (below).

M) (.,.)M<bufname> Move.

The addressed lines are removed from the current buffer. "." in the current buffer is set to the line following the last line moved. The entire contents of buffer <bufname> are deleted, and then replaced by the addressed lines. "." in the named buffer is set to its last line. The named buffer can be the current buffer.

N) N<regname><character><number> Number.

This command modifies a number register named <regname>. The value of the number register can be retrieved by the sequence "\N<regname>" (See "Editor Input" above). The following <character>s may be used:

: The <number> is assigned to <regname>.

+ The <number> is added to <regname>.

- The <number> is subtracted from <regname>.

* <regname> is multiplied by <number>.

/ <regname> is divided by <number>.

% <regname> becomes <regname> modulo <number>.

p The contents of <regname> are printed on the console. <number> is ignored.

d Whenever <regname> is retrieved by a "\N", it will have at least <number> digits; leading zeros will be supplied to pad. (Initially this number of digits is one.)

O) O<optionlist><nl> Options

The O command sets various editing options. The following options may be used in the <optionlist>:

V Verbose
Whenever QED is in text input mode (after A, C, or I commands) the prompt character "*" is supplied at the beginning of each line. This feature may be useful to those who forget their "\F"'s.

S Silent
This option annuls the effect of option V.

T<number>,<number>,... Tabs
QED turns tab characters into spaces when writing

Hollerith card image files, and tabs are inserted and deleted by the Z command (below). The tab settings 8, 16, 24, 32, ... are the default. The Tabs option changes the tab settings. As many tabs as desired may be set but no tab setting may go into a column beyond 73. Tab characters occurring in columns beyond the rightmost tab stop are treated as spaces.

O special characters Out
After an option O, the following regular expression metacharacters are treated as ordinary characters:

. * ( ) { } [\E ^ $ |

The special effect of these characters in regular expressions may be restored locally by preceding them by "\C".

I special characters In
This character annuls the effect of option O.

P) (.,.)P Print
<addr><nl>
<nl>

The addressed lines are printed on the console. "." is set to the last line printed. The addressed lines cannot cross line 0 (circularly). A completely empty line is equivalent to ".+1p".

Q) Q<nl> Quit.

QED returns to the "system?" level.

R) ($)R <filestring><nl> Read.
($)R6 <filestring><nl>
($)R<nl>

The file is read and appended after the addressed line. "." is set to the last line read. The size of the file in blocks, the number of lines read, and the number of characters read are printed on the console provided the command is not executed from a buffer. In the first form, the file is ASCII; in the second, it is a card image file. If the "R" is followed directly by <nl>, the file of that name and type last mentioned in L, R, or W commands is read. If the command is executed from a buffer, the condition code is set to true to reflect a successful read, otherwise to false. In this case, no error is given, since it is assumed that a program will test the condition register if desired.

S) (.,.)S/<regexp>/<string>/ Substitute.

Occurrences of <regexp> in the addressed lines are replaced by <string>. "." is set to the last line in which a substitution took place. The character "&" in the <string> has a value equal to the text matched by the <regexp>. If a construct of the form "{<regexp2>}x" was used in the <regexp>, the character "x" has value equal to the text matched by <regexp2>.

Any character but space or <nl> may be used instead of "/" to bound the <regexp>.

If any substitutions took place, the condition register is set to true, otherwise to false. It is not an error for a substitution to fail.

T) (.,.)T/<regexp>/ Test

The T command attempts to find an instance of the specified <regexp> among the specified lines; if successful, the condition register is set to true, otherwise to false. "." is left unchnged. The usually illegal address "0" is legal for the T command, so

```
0,$T/$/
```

tests whether there are any lines in the current buffer and never gives an error. ("1,$T/$/" would be erroneous if the current buffer were empty.)

V) (1,$)V/<regexp>/<commands><nl> Exclude.

This command is exactly like the "G" command, except that the commands are performed in lines not matching the <regexp>.

W) (1,$)W <filestring><nl> Write.
(1,$)W6 <filestring><nl>
(1,$)W<nl>

The addressed lines are written into the file. The number of blocks, the number of lines, and the number of characters written will be printed. The value of "." is not changed. In the first form the file will be ASCII; in the second, 14-word Hollerith card images. If "W" is followed directly by <nl>, the file is written with the name and type of the last file mentioned in L, R, or W commands. If the file cannot be opened for writing, a temporary file of the given name is created and a diagnostic printed. When the W command is executed from a

buffer, the condition register is set to indicate the success of the W command, and no error comment is given.

X) X Status.

The name, value of ".", and value of "$" of the current buffer and all non-empty auxiliary buffers are printed on the console. The value of "." is unchanged.

Y) (.,.)Y/<string1>/<string2>/ Transform.

Occurrences of the characters in <string1> within the addressed lines are replaced by the corresponding character of <string2>. <string1> and <string2> must be the same length, and no character may appear twice in <string1>. The value of "." is set to the last line in which any character was transformed. The condition register is set to true if any characters were transformed, otherwise to false. Any character but space or <nl> may be used instead of "/" to bound the strings.

Z) (1,$)ZI Tabs In
(1,$)ZO Tabs Out

In the first form of the command, every sequence of one or more spaces that ends on a tab stop is converted to a tab character. In the second, each tab character is expanded to the right number of spaces according to the current tab stops. Tabs are set by the OT command (see above). Neither ZI nor ZO treats backspace and other non-graphics as special characters, so even if logical tab settings correspond with physical settings visual appearance is not necessarily preserved. Furthermore, tab itself is not a special character during ZI.

"." is set to the last line altered.

:) ($): Absolute line.

The absolute line number of the addressed line is printed. If the absolute line number is undefined, "?" is printed. "." is set to the addressed line.

=) ($)= Current line.

The addressed line's number is printed. "." is set to the addressed line.

!) ! <TSS command> Escape to TSS

When "!" is encountered the remainder of the line is sent to GE-TSS to be executed. The command

```
!filsys ...
```

is particularly useful.

") (.)"<anything but <nl>><nl> Comment.

The remainder of the line is ignored. "." is set to the addressed line.

Usage

At "system?" level, type

```
./qed
```

or the equivalent using lodx.

The break key

Whenever the break key on the teletypewriter is depressed or the ASCII character NULL is sent, QED will stop what it is doing, return to its command level and respond with "?11" (i. e., error number 11). This feature is particularly useful for interrupting long printouts. Caution should be exercised to avoid sending a second break before the first has taken effect; this is likely to return the user to the "system?" level of GE-TSS, with consequent loss of the work in progress.

Core overflow

During the day, GE-TSS limits the user to 32K of core of which about 3.3K is used by the QED program. The remainder can be used for text storage, and it amounts to about 50 blocks ("1links") worth. Unfortunately, TSS aborts a program which asks for more core than TSS is willing to give. Therefore, as a safety feature, QED will simulate an interrupt as its core size passes 31K. If an unexpected error 11 ("?11") occurs, use the F command to discover whether memory is about to overflow.

Line 0

The following description of the way buffers are stored is sometimes useful to know. The lines in each buffer are circularly linked, with a special, always empty, "line 0" between lines 1 and $. That is, near the end of the buffer, the order of lines is ... $-2, $-1, $, 0, 1, 2, ... Line 0 cannot be printed, deleted, or in any way changed; its presence becomes noticible occasionally because "." may be left at line 0 by, for example, deleting line $. Certain searches can find this line, for example "/^/". (But not "/^$/", since the "$" requires a newline to be present.) Finally 0 can sometimes be used as an address: "0a" and "0r" place text at the beginning of the buffer. (Note that "0i" puts the text at the end!)

File I/O

QED accepts filenames in catalogs other than the user's master catalog provided no more than two "/"s are present. Passwords and alternate names are not accepted, and both file and catalog names must be 8 or fewer characters in length.

In order to guard against an unfortunate bug in TSS, the following strategy is used when opening files. If the file was not open (in the AFT), and QED is able to access it with the proper permissions, all is well. If the file is being read, all is assumed to be well. Otherwise, if a file is in the AFT, QED will deaccess it and then attempt to reopen it. This is necessary because if a file is in the AFT with the wrong permissions, an I/O command to the file will cause QED to be aborted. There is no way to discover the permissions attached to a file in the AFT.

As mentioned under the W command, if a file cannot be opened for writing (usually because it dces not exist) QED will create a temporary file which the user must deal with outside of QED.

QED will write TSS-format ASCII and 14-word (84 column) GE Hollerith card image files. It will read TSS ASCII and card image files the records of which may vary in length.

Bootstrapping

QED will automatically read in and execute files containing QED commands. This feature is useful in preparing QED programs for general use. If QED is invoked by the following command line

```
./qed filename arg1 arg2 ...
```

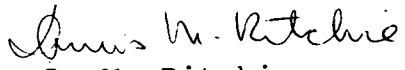
then "r filename" is placed in buffer "f", arg1, arg2, ... are placed in separate lines in buffer "0"; buffer "f" is executed to read filename into buffer ".", and finally buffer "." is executed. This feature is implemented by executing the following commands from a hidden buffer:

```
b0                      "start in b0
a \R
\F                      "read command line
1,$s/  */\C
/                       "split into separate lines
1d                      "delete "./qed"
0;/./;.mf               "move filename (if there) to bf
bf                      "edit filename
1s/^/r /                "make R command
b.                      "move to buffer .
\Bf                     "execute read
b0                      "back to b0
\B.                     "execute file read in
```

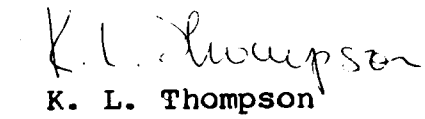
The use of the bootstrapping feature is illustrated in the last example in the Appendix.

Comments

Comments on the (syntax|semantics) of the (paper|program) are welcome.



D. M. Ritchie



K. L. Thompson

MH-1373-DMR

References
Appendix - 6 pages

REFERENCES

[1] Thompson, K. L., "QED Text Editor," Multics Repository Document B0080, February, 1967.

[2] Kernighan, B. W., "A Tutorial Introduction to the QED Text Editor," MM-70-1373-6 (June, 1970).

[3] ----------, "A Guide to Advanced Use of the QED Text Editor," MM-70-1373-7 (July, 1970).

[4] Deutsch, L. P. and B. W. Lampson, "An Online Editor," CACM 10, 12 (December, 1967).

APPENDIX

This Appendix provides a quick-reference summary of QED and several examples of its use.

Commands

The bracketed numbers give the page number on which to find the description of the command.

A)	($)A<text>	Append [8]
B)	B<bufname>	Buffer [8]
C)	(.,.)C<text>	Change [9]
D)	(.,.)D	Delete [9]
E)	E<rexname>/<regexp>/	Enter r. e. [9]
F)	F	Facts on core size [9]
G)	(1,$)G/<regexp>/<commands><nl>	Global [9]
I)	(.)I<text>	Insert [10]
J)	JT<commands>	Jump if true [10]
	JF<commands>	Jump if false
K)	(.,.)K<nl>	Sort (increasing) [10]
	(.,.)KB<nl>	(decreasing)
L)	L <filestring><nl>	List [10]
	L6 <filestring><nl>	
	L<nl>	
M)	(.,.)M<bufname>	Move [11]
N)	N<regname>:<number>	Number (assign) [11]
	N<regname>+<number>	(add)
	N<regname>-<number>	(subtract)
	N<regname>*<number>	(multiply)
	N<regname>/<number>	(divide)
	N<regname>%<number>	(remainder)
	N<regname>p<number>	(print)
	N<regname>d<number>	(set digits)
O)	OV	Option (Verbose) [11]
	OS	(Silent)
	OO	(r. e. chars Out)
	OI	(r. e. chars In)
	OT<number>,<number>,...	(Tab setting)
P)	(.,.)P	Print [12]
	<addr><nl>	
	<nl>	
Q)	Q<nl>	Quit [12]
R)	($)R <filestring><nl>	Read [12]
	($)R6 <filestring><nl>	
	($)R<nl>	
S)	(.,.)S/<regexp>/<string>/	Substitute [13]
T)	(.,.)T/<regexp>/	Test for <regexp> [13]
V)	(1,$)V/<regexp>/<commands><nl>	Exclude [13]
W)	(1,$)W <filestring><nl>	Write [13]
	(1,$)W6 <filestring><nl>	
	(1,$)W<nl>	
X)	X	Status [14]
Y)	(.,.)Y/<string>/<string>/	Transform [14]
Z)	(1,$)ZO	Tabs Out [14]
	(1,$)ZI	Tabs In
:)	($):	Absolute line # [14]
=)	($)=	Current line # [14]
!)	!<TSS command>	Escape to TSS [15]
")	(.)"<anything><nl>	Comment [15]

Diagnostics

?0	Internal buffer overflow
?1	RE search failed
?2	Unrecognized command or address
?3	RE syntax error
?4	Address syntax error
?5	Address wrap around (e.g. 5,2p)
?6	Address out of buffer
?7	Absolute line search failed
?8	Unaccessible file
?9	Command syntax error
?10	Bad status during I/O (usually EOF during write)
?11	Interrupt or memory warning
?12	Cannot open output file; temporary created instead

Regular expression metacharacters

^	Null character at beginning of line
.	Any character but <nl>
$	Null character before <nl>
[	Start of character class
*	Kleene closure
|	Alternation
()	Grouping
\E<rexname>	Named regular expression
{}<char>	Tagged regular expression (makes <char> a substitute metacharacter)

Substitute metacharacters

&	Matched text

Stream directive and escape characters

\B	Expand buffer
\R	Read console
\F	End input mode
\N	Insert number
\C	Take next character literally
\E	Introduce name of regular expression
\A	@
\\	\
\(	{
\)	}
\!	|
\<	[
\>	]
\^	^
\ddd	ASCII character ddd

Examples

1) Creating text.

The following commands create a file of little import and write it out.

```
a
Here are some commands
which create a file of
little import
and write it out.
\f
w litimp
```

2) Moving text.

The commands

```
/abcd/,$m1
0a \b1\f
```

first move several lines from the end of the current buffer to buffer 1. Then the lines are inserted at the beginning of the current buffer.

3) Correcting mistakes.

The command

```
1,$s/speling/spelling/
```

changes every instance of "speling" to "spelling". Better, the command

```
1,$s/{S|s}Xpeling/Xpelling/
```

works even if "speling" is sometimes capitalized.

4) Deleting line numbers.

The command

```
1,$s/^[0123456789]*//
```

will remove line numbers as defined in GE-TSS FORTRAN from a buffer.

5) Adding line numbers.

The command sequence

```
n(counter):1
n(counter)d5
g/^/s''\c\n(counter)#' n(counter)+10
```

will add line numbers (as defined by CARDIN) to a buffer beginning with 00001 and incrementing by 00010.

6) Verification of substitution.

The command

```
g/ABCD/p\c\rs//PQRS/
```

will print each line in which "ABCD" is found and pause. If '<nl>' is typed, "PQRS" will be substituted. If '"<nl>' is typed, the substitution will not take place. Note the use of "\c" to postpone interpretation of "\r" until the G command is executed.

7) Macro expansion.

The command

```
1,$s/ADDMAC    {[^,]*}1,{[^,]*}2,{[^,]*}3/LDA      1\c
        ADA      2\c
        STA      3/
```

turns lines like

```
ADDMAC    P1,SYMB,RES
```

into

```
LDA       P1
ADA       SYMB
STA       RES
```

Note the use of "\c" to insert <nl> into the right-hand side of the substitution.

8) Global move.

The command

```
g/ab\c$cd/ m(temp) b1a \c\b(temp)\f
```

moves all lines containing "ab$cd" to buffer 1. The lines are appended to the buffer and do not replace it. "\c" is used to remove the special meaning of "$" in the left side of the substitution and to prevent the expansion of buffer "temp" during reading of the global command.

9) Summing a table.

Suppose buffer 0 is the current buffer and each line consists of a number; all of these numbers are to be added together. The following commands suffice:

```
n(sum):0
1,$s/^/n(sum)+/
\B0
```

That is, each line in the current buffer becomes like "n(sum)+<number>"; then the buffer is executed to form the sum.

10) The use of "\E"

The command

```
e(alph)/[abcdefghijklmnopqrstuvwxzy]/
```

makes "\E(alph)" refer to the set of lower case letters. The regular expression

```
/\E(alph)/
```

will search for a lower case letter. The regular expression "\E(bal)" defined by

```
e(bal)/[^`']*(`\E(bal)')*[^`']*/
```

matches any string balanced with respect to single quotes "`" and "'".

It should be noted that the ability to define regular expressions recursively makes the term "regular expression" a misnomer: it is not hard to see that expressions can be constructed to match exactly the members of any given context-free language.

11) A useful program.

Here is an edited version of a QED program written by S. C. Johnson. (Read also "./filelist" for more information.) When called by

```
./qed ./list file1 file2 ...
```

the program writes a control card file on *SRC and uses "./rj" to submit it to the batch world. The batch program will list the named files on the high-speed printer. It uses a file which the user must supply, called IDENT. The first line of IDENT should contain the user's UMC name; the second, the $IDENT card to head the control card file. Subsequent lines may contain $USERID and $LIMITS cards if desired. (Note: the actual "./list" uses a slightly different format for the IDENT file and will create this file automatically.)

```
b1                      "read IDENT file into buffer 1
r ident
1m2                     "move UMC name to buffer 2
b0                      "edit filenames
1,$v'/' s'^'%/'         "add "%/" to lines without "/"
b2                      "edit UMC name
1s/^/1,$s'%'/           "UMC becomes "1,$s'%'UMC"
s/$/'/                  "b2 becomes "1,$s'%'UMC'"
b0                      "edit list of files
\B2                     "execute b2 = 1,$s'%'UMC'
1,$s/^/$  prmfl         in,,,/    "make $PRMFL cards
g/prmfl/i\C
$         select        cc/list\C
\F                      "insert $SELECT before each $PRMFL card
a
$         endjob
\F                      "add $ENDJOB card
b1                      "edit remaining part of IDENT file
v/^\C$/d                "remove all but $control cards
$a
\B0\F                   "append $SELECT and $PRMFL cards
                        "write file on *SRC ready for "./rj"
w *src
!./rj
q
```