TO: MSPM Distribution

FROM: Charles Garman

DATE: August 3, 1966

SUBJ: Context Editor, BX.9.01

The attached MSPM section BX.9.01 is a major revision and obsoletes the version of 8 May 1966. The major changes are:

1. Addition of implementation specifications
2. Substitution of segment addressing for stream-name text I/O.

Published: 8/3/66
Major revision
(Supercedes: BX.9.01, 4/8/66)

Identification

Context Editor
edit
Charles Garman

Purpose

Edit is a context editor used to create or modify canonical-form (ASCII) text files (see sections BC.2.00-2.04, Character Input/Output for Multics) through user interaction.

Introduction

The edit command is a lineal descendant of the TYPSET command in CTSS by Jerry Saltzer (AH.9.01), and in general follows TYPSET's conventions; some modifications were made to reflect experience gained using TYPSET and similar context editors. On the other hand, certain of TYPSET's innovations have gained system-wide acceptance in Multics (e.g., canonical-form character operations).

Usage

edit input_file_name -output_file_name-

Input_file_name is the name of the file which provides the source text for edit.

Output_file_name is the name of the file in which the edited text will be placed; if not provided, it is assumed to be the same as input_file_name.

When edit is ready for typing to begin, the word "Input" or "Edit" is typed, and the user may begin. If input_file_name could not be read (i.e., the file was empty), the user begins in Input mode; otherwise, he begins in Edit mode.

Input mode

In Input mode, the user types character strings separated by new-line (NL) characters. He does not wait for response from edit between lines, but types as rapidly as he desires. Lines containing only the new-line character are entered as text.

The user leaves Input mode and enters Edit mode by typing a line containing only the mode-change-character (MCC)

followed by the NL character. The default value of the MCC is the period (.); however, this may be changed by a user option. When switching modes, edit acknowledges the change by typing the name of the new mode, "Input" or "Edit"..

Edit mode

In Edit mode, edit recognizes "requests" of the form

request_name -pertinent_arguments-

All requests take effect immediately on a copy of the text being edited. Except where a request is expected to cause a response (such as print), further requests may be entered immediately (queued) without waiting for a response from edit; however, if any message is printed, because of error conditions or due to the user's "complete" option settings at the end of a particular request, all currently queued requests will be deleted. Requests must be separated by NL characters; request lines containing only the NL character will be ignored.

Requests

Editing is done line by line. One may envision an index or pointer which at the beginning of editing points to the first line of the text. This pointer is moved to different lines by some requests, while other requests specify some action to be done to the line designated by the pointer (the "current line").

The normal movement of the pointer is from the beginning of the text to the end; however, the direction of pointer movement may be modified for certain operations, either for the duration of a single request, or by changing an option. Those requests to which pointer direction reversal may be applied are marked in the Summary by "-".

All requests may be abbreviated by giving only the first letter. Illegal or misspelled requests (including non-numeric characters in argument positions denoted by "-n-" in request definitions) will receive an error indication; the request will be ignored.

A single asterisk (*) in place of an n has the special meaning of "infinity", or more precisely, the number of lines between the pointer position and the end of the text. The absence of characters where an n is expected (the null string) denotes the default value of 1.

Responses printed during or upon completion of individual requests, as specified in the descriptions below, are subject to the current state of the user's brief/complete options for the particular request; for this implementation,

however, error messages may not be suppressed.

locate -character_string-

This request moves the pointer to the first line which contains the given character string. Only enough of the line need be specified to identify it uniquely. The line which has been found is printed in its entirety.

If character_string was not specified, the locate request will search with the character string used in the most recent locate request.

The occurrence of any simple carriage motion (see BC.2.02, The Canonical Form) in the character string will match any simple carriage motion in the text string. Thus one or more blanks or tabs are equivalent between print-positions of the character string.

If the locate request fails to find a line containing the given character string before reaching the end of the text, an error message is printed; the pointer is left where it was before the locate request was given.

NOTE: Because trailing blanks are stripped from input lines by the canonical-form converter, the user may not locate trailing blanks after a character sequence directly; in such cases, he may provide further context by embedding the blanks between the desired character sequence and some portion of the text immediately to the right of the blanks, or else re-issue the locate request without arguments as many times as necessary.

print -n-

Starting with the current line, n lines are "displayed" on the user's console; the pointer is not moved.

delete -n-

This request deletes n lines, starting with the current line. The pointer is moved to the line after the last deleted line; the new line is printed.

next -n-

This request moves the pointer n lines; the n'th line is printed.

insert new_line

The line new_line will be inserted below the current line (or above the line if pointer movement is in the bottom-to-top direction). The first blank following the request word is part of the request word, and not part of the new line. The pointer is set to the new line.

To insert more than one line, the user gives several insert requests, or types the MCC followed by NL to switch to Input mode. When the user returns to Edit mode by typing the MCC followed by NL, the pointer sits by the last inserted line.

If the user's first request to edit is an insert, the inserted lines are placed after the first line of the text. If an insert is given after the pointer has run off the end of the text, the inserted lines are placed after the last line of the text.

replace new_contents

The line new_contents replaces the current line. The first blank following the request word is part of the request word and therefore is not part of the new line. The pointer is not moved.

change /string_1/string_2/ -n- -"g"-

If the letter "g" is present, the next n occurrences of the character-string string_1 will be replaced by the character-string string_2, beginning with the current line and continuing as far as necessary.

If "g" is absent, only the first occurrence of string_1 within a given line will be changed.

If string_1 is void, "g" has no effect, and string_2 will be inserted at the beginning of n successive lines, beginning with the current line.

Any "print position" not occurring within either character string may be used in place of the slash (/) for "quoting" the character strings. "Print position" is used here in the sense defined in MSPM Section BC.2.02 (Canonical Form Character I/O).

All lines in which a change was made are printed; the pointer is not moved.

If the end of the text is encountered before n occurrences are found, an error message is printed.

Example:

```
line:          It is a nice day in Boston.
request:       change "is"was"
new line:      It was a nice day in Boston.
request:       change xwasxisx
new line:      It is a nice day in Boston.
request:       change ' '.' g
new line:      It.is.a.nice.day.in.Boston.
request:       change '.''
new line:      Itis.a.nice.day.in.Boston.
request:       change "tis"t is"
request:       change '.' ' g
request:       change 'on'on.'
new line:      It is a nice day in Boston.
```

top

This request sets the pointer to the first line of the text, which is then printed.

bottom

This request sets the pointer to the last line of the text, which is then printed.

option parameter -argument_list-

The initial setting of options will be taken from user_profile, as set by the Shell and the option commands; the option request modifies these settings for the duration of the edit command only, or until reset by another option request.

Parameter is the name of the option being referred to; argument_list provides additional arguments as specified below.

The following are the permissible forms of the option request.

$$\text{option} \left\{ \begin{matrix} \text{brief} \\ \\ \text{complete} \end{matrix} \right\} \text{-request\_1- ... -request\_n-}$$

In this form, the request_i refer to the names of the edit requests which print responses during their operation, or upon completion. Responses from particular requests may be suppressed by use of the brief parameter, or re-enabled by use of the complete parameter. In the Summary of edit requests, the request_i which may occur in argument_list are flagged by "*". If no request_i were specified in argument_list, all of the pertinent responses are suppressed (enabled). The request names may be

abbreviated in the same manner as the requests themselves. (For the purposes of the option request, the Input/Edit switching is named "mode_change", abbreviation "m".) The initial or default setting is:

option complete

option forward
option reverse

This form sets the direction of pointer movement to top-to-bottom (bottom-to-top). The default setting is:

option forward

(To specify the direction for the current edit request only, the characters "+" (forward) or "-" (reverse) may precede the request name.)

option mode_change print_position

The contents of "print_position" will replace the current MCC. The default setting is:

option mode_change .

option status

The current settings of edit's options will be displayed in a symbolic form on the user's console, including the current MCC.

send -file_name-

The new or modified text will be placed in the specified file; edit then returns to its caller (usually the Shell). Any argument to the send request over-rides the arguments of the edit command itself.

exit

This request causes edit to return to its caller without send-ing; all modifications made to the text will be forgotten.

Implementation

The operation of the edit command may be conceptually separated into three phases:

1) transformation of the user's input text (input_file_name) into an equivalent representation in a threaded list;
2) processing of user requests, which may involve modification, insertion, or deletion of items on the thread; and
3) transformation of the threaded-list representation back to sequential canonical-form ASCII by transmission of the items on the thread into output_file_name.

Each of the three phases could be a separate procedure, all controlled by a minimal "main" program which (trivially) would call each in succession, with appropriate arguments as returned by the previous phases; in fact, Phases I and II are intermingled to reduce overhead.

Internal Representation ofText

The text which the user edits is maintained internally in a bi-directional threaded list (to allow implementation of backward and forward operations), each item (bead) on the thread representing one line of ASCII text (including the new-line character (NL)).

In Phase I, a unique segment-name is manufactured by concatenating input_file_name with a unique identifier. This segment name is used to create the segment which holds the threaded-list representation of the text, and to obtain the pointer-variable which identifies that segment.

The backward- and forward-pointers of each bead are 18-bit quantities which represent the index within the segment of the predecessor and successor beads. This scheme was chosen to allow later implementation of a resumption phase, which would begin work immediately on the threaded segment left over from a previous invocation of the editor, thus saving the time spent in Phase I threading the list; the segment may be reactivated later with a different segment-number since it does not have segment-dependent information (i.e., its pointers) embedded in its contents.

In addition, information is kept within each bead of the number of print-positions occupied by each line, as well as the number of characters in the text of the line, to simplify and speed up searching procedures.

Reference Mechanism

In order to implement the threaded list with embedded text and still preserve its segment-independence, a modified threaded-list structure is used. This structure and some of the operations upon it are described below. The PL/I declarations, as shown, are not final, but are given to provide some flavor of the method of implementation; since the proposed data-structure is somewhat implementation-dependent, references will be localized within one procedure.

```
dcl (p, q, r) ptr;

dcl 1 line ctl(q),
      2 index,
        3 (backward, forward) bit(18),
      2 print_positions,
        3 (non_blank, total) fixed,
      2 character_count fixed,
      2 text char (q -> line.character_count);

dcl ptr ext entry (ptr, bit(18)) ptr;

dcl rel ext entry(ptr) bit(18);
```

Here the function ptr provides an its pointer to the origin of a particular instance of a controlled structure in a segment, when given the base of the segment and the (18-bit) offset of the structure within the segment; the function rel provides the inverse transformation, i.e., it returns the (18-bit) offset within the text segment of the pointer which identifies a particular structure in a threaded list.

With these declarations in mind, let p contain the pointer to the base of the threaded segment, and let q point to one instance of line on the thread, the text for which is contained in line.text.

The statement

```
r = ptr(p, q -> line.index.forward);
```

places into r a pointer to the successor item in the thread; thus, a reference to

```
r -> line.print_positions.total
```

would give the total number of print-positions (columns) contained in the following line.

Phase 1: Initialization and Text Transformation

Phase I involves all operations neccessary to prepare the user's text for editing. This includes:

1) setting up editor options from user_profile.
2) generation of the unique segment name and creation and initialization of the segment.
3) activation of input_file_name.
4) removal of lines from the input segment, scanning for the NL character.
5) addition of lines read in (4) to the threaded list.
6) repetition of (4) and (5) until the end of the text input is reached.
7) deactivation of input_file_name.

Items (4), (5) and (6) of the algorithm are in fact performed on an as-needed basis on the first trip through the input text; that is, the structure shows an end-of-thread index ("0"b) on the last item of the structure, but the end-of-text is not recognized until the last line has been removed from the input text segment, at which time item (7) above is performed.

Phase II: Editor Requests and New Text Input

Upon completion of items 1 -> 3 under Phase I, the edit command is ready to accept either the user's editing requests or his input. The mode is set for Input if the file in the branch was of zero-length; otherwise the mode is set for Edit.

All input from the user's console-stream is processed through a procedure which reads a line using NL as a break character, counts the number of print-positions in the line, and produces a map of the print positions. If this procedure discovers that the input line contains only one print-position, and that the contents of the print-position match the current mode-change-character (MCC) it uses an alternate return to its caller, which may then perform any further manipulations necessary before switching to the opposite mode.

In Input mode, after reading each line, it is then inserted into the threaded list, and control returns to the point at which more input is requested.

In Edit mode, for each request line read, if it contains only the NL character, it is ignored, no comment is printed, and the editor immediately awaits the next request. Otherwise, the line is scanned to form the request name or its abbreviation, a lookup operation is performed, and control passes to the appropriate request handler. If no match was found an error comment is printed, followed by the

contents of the line in error, and control proceeds to look for a new request.

Request Handling

This section provides an abbreviated description of the operations of each request.

locate: move remainder of input line to comparison buffer and set length of buffer (unless usage of comparison text from previous locate request was specified), then call a procedure which compares the given string against the text line-by-line. On a "match" return, set the current pointer to the line in which the match was found, and proceed to the section which prints responses (in this and all subsequent references, printing is of course conditiond by the setting of the brief/complete option for the particular request). If no match is found print error message, but do not modify current pointer.

print: decode n, print line and follow thread for n lines, do not change current line index.

delete: decode n, release line and follow thread n times; move to succeeding line and print.

next: decode n, follow thread n times, change current index, print.

insert: insert remainder of line as a new item; set current index to point to new line.

mode_change: obtain specified print position contents and replace over previous mode-change-character.

replace: insert line in place (as at insert) and release contents of current line.

change: obtain print position whose contents acts as "quoting character", search for matches in request line to isolate strings involved; call for search for first substring, on match replace with second substring; if "global", request search for further matches in current line, otherwise proceed to next line; stop after finding n matches. (Print lines as changed if printing is requested.)

top, bottom: reset index to index of remembered top (bottom) item.

option: decode parameters and further sub-arguments, set/reset option switches as indicated; print symbolic summary for "status".

send: obtain optional parameter of request and return its value through argument list to caller of Phase II; return to caller of Phase II.

exit: take alternate return to caller of Phase II, indicating no invocation of Phase III.

Phase III: Transformation to Serial File

Read through the threaded list line by line, beginning at the top, transmit the text of each line to output_file_name. Finally, release segment obtained in Phase I, return to original caller of edit.

Note on Search Algorithm

The search method is a three stage procedure; it checks first to see if enough print-positions (blank and non-blank) exist (remain) in the line; succeeding, it then compares the maps of the print positions of the items by character count; and with successful match here, then and only then beings comparing the contents of appropriate print positions.

Print-position maps are representations of canonical-form lines in terms of the number of storage-character positions occupied by each print-position, together with indices representing the location (a la SUBSTR) of the beginning of each print position relative to the beginning of a scalar character string.

The print-position map for the contents of the "memory" buffer required for the default argument of the locate request remains available until changed; all other print-position maps are generated as the character string becomes available. In the particular case of instances of the structure line, a print-position summary is kept in the structure: the total number of print positions, and the number occupied by non-blank characters.

Free Storage and Related Items

The items in the threaded list are allocate-ed and free-ed using the standard EPL-PL/I statements; the segment created during Phase I is set up as a controlled area and storage in it is subsequently assigned via an allocate statement (using the in-clause).

Restrictions

Internal buffers in "automatic" storage impose the following restrictions (the values chosen are essentially arbitrary):

1023 characters per line
511 print-positions per line.

Summary of edit requests

abbreviation	request	response
- l	locate -character_string-	line found, * end-of-text
- p	print -n-	printed lines, end-of-text
- d	delete -n-	following line, * end-of-text
- n	next -n-	line reached, * end-of-text
- i	insert new_line	(none)
(m)	(mode_change)	"Input", * "Edit" *
r	replace new_contents	(none)
- c	change /xx/yyy/ -n- -"g"-	changed lines, * end-of-text
t	top	first line *
b	bottom	last line *
o	option parameter -list-	invalid option-name
s	send -file_name-	(response from Shell)
e	exit	(response from Shell)

* These responses are subject to the brief/complete settings of the option request; the names of the requests to which they refer are the only ones which may be set by option brief/complete.

- These requests are subject to changes of direction.

The slash (/) in the change request may be replaced by any "print position" not appearing within the quoted character strings.

Summary of option-request forms

abbreviation	form	
b	brief	-request_names-
c	complete	
f	forward	
r	reverse	
m	mode_change print_position	
s	status	