bcc	title M1CS Phase One Language Editor	prefix/class-number.revision CSED/M- 12

checked Larry L Barnes	authors L. Peter Deutsch L. Peter Deutsch R. K. Dove R. K. Dove	approval date 10/15/69	revision	date
checked		classification Manual		
approved Mel		distribution Company Private		pages 20

ABSTRACT and CONTENTS

This document serves as a user manual for the phase 1 language editor. The appendix documents the line input editing facility. Syntax and semantics appearing in this manual will be maintained for in house use and are not expected to change until the phase 2 version is released. Phase 2 will be a complete finished system for external use.

INTRODUCTION

The following pages are concerned with the M1CS language editor for SPL and FORTRAN programs. This facility manifests itself to the user as a collection of commands and concepts in two flavors: basic and extended. The extended language editor is upward compatible to the basic version and provides the experienced user with quick and convenient ways to do complex editing.

Additionally, the distinction of phase 1 and phase 2 is necessary. The phase 1 language editor is an interim facility to be used in house only and will eventually be superseded by phase 2. The primary objective of the following pages is to document the phase 1 language editor as it is implemented. Phase 1 does not make a distinction between basic and extended versions. The phase 2 basic version will be a subset of what appears on the following pages, while the phase 2 extended version will be a superset.

GENERAL CONCEPTS

The basic difference between a language editor and a text editor is in the way the material to be edited is viewed. Usually, a text editor views its material as a collection of characters. On the other hand, a language editor has a higher level of understanding which allows it to view its material as a collection of tokens, where each token is a collection of characters. In other words, a text editor might view "132+TEMPERATURE" as 15 characters, whereas a language editor would view it as 3 tokens. A more sophisticated language editor would further recognize the 3 tokens as a number, an operator, and a symbol.

The M1CS language editor views both SPL and FORTRAN programs as collections of tokens. Furthermore, it recognizes certain tokens and structural concepts. Structurally it is aware of lines and blocks, where the definition of block depends upon the programming language. The tokens recognized include block names, all FORTRAN labels, and SPL labels which appear as the first token on a line.

The purpose of this language editor is to provide a means for creating and altering programs. Consequently, there are ways to request editing actions to be performed as well as address physically where they are to occur in the program. The basic addressable quantity is a line. The language editor is always aware of a "current line" and allows addressing of the subsequent line to be expressed as relative to the current line, relative to the current block, or absolute to the entire program. A complete discussion appears in the semantics section.

SYNTAX

The syntax appearing in this section is strictly for phase 1. It is anticipated that phase 2 will be an upward compatible extension. The character "¢" signifies the end of a line and represents a carriage-return-line-feed. Although the 14 commands appear in their verbose form, the language editor will recognize any contiguous subset of characters which starts with the initial character. Thus, SUBSTITUTE means the same as SUBST, SUB, and S. The first character following the command must be other than a letter or digit.

```
language:editor:command =
                 "APPEND" [address] text
               | "CHANGE" [interval] text
               | "DELETE" [interval] ¢
               | "EDIT" [modes] [address] ¢ line ¢
               | "INSERT" [address] text
               | "LIST" [modes] [interval] ¢
               | "MODE" modespecs ¢
               | "NEXT" [modes] [integer] ¢
               | "PREVIOUS" [modes] [integer] ¢
               | "READ" file [address] ¢
               | "SUBSTITUTE" [modes] subspec [interval] ¢
               | "UNDO" ¢
               | "VALUE" [address] ¢
               | "WRITE" file [interval] ¢ ;
```

```
modes           = modespecs ":" ;
modespecs       = 1$(modespec) ;
modespec        = "A" | "B" | "C" | "I" | "N" | integer ;
text            = ⊄ $(line ⊄) B^c
                | ["="] "[" interval "]" ⊄ ;
B^c             = <character code 142_8 = control B> ;
sign            = "+" | "-" ;
integer         = 1$(digit) ;
file            = <a 940 file name> ;
line            = $(character-⊄) ;
interval        = address
                | address "," address ;
address         = head $(tail)
                | [block] search $(tail) ;
head            = "."
                | [block] label
                | [block] "#" integer
                | [block] "$" ;
tail            = search
                | sign integer ;'
block           = "<" [name] ">" ;
search          = ["-"] token:search ;
token:search    = "/" [tokens-"/"] "/"
                | "*" [tokens-"*"] "*" ;
label           = name ;
name            = letter $(letter | digit | "'") ;
```

subspec	= "/" [tokens-"/"] "/" [tokens-"/"] "/"
	| "*" [tokens-"*"] "*" [tokens-"*"] "*"
tokens	= token $(token)
token	= <defined by the language's syntax>

SEMANTICS

APPEND:	Appends the text after the address. If no address is specified, appends after the current line. The last line appended becomes the current line. If no lines are supplied, the addressed line becomes current.
CHANGE:	Replaces the interval by the text. If no interval is specified, replaces the current line. The last line of the text becomes the current line. If no lines are supplied, the first line of the interval becomes current. The number of lines changed will be printed if the interval consisted of two addresses rather than one. CHANGEs may not extend across block boundaries. The only way to CHANGE the first line of a block is to CHANGE the entire block.
DELETE:	Deletes the interval. If no interval is specified, deletes the current line. The line before the interval becomes current. The number of lines deleted will be printed if the interval consisted of two addresses rather than one. The only way to delete the first line of a block is to delete the entire block. Deletions may not extend across block boundaries.
EDIT:	Uses the addressed line as the "old line" for the line editor. Editing conventions for the line

editor are in the appendix. If no address is specified, the current line is used. The EDITed line becomes current. Meaningful modes are "A", print the new line after EDITing; and "B", print the old line before EDITing. Modes appearing with the EDIT command are temporary and do not disturb the permanent modes set by the MODE command. For a complete discussion of modes, see the semantics of the MODE command.

INSERT: Inserts the text before the specified line or before the current line if no address. The last line inserted becomes current. If no lines are supplied the addressed line becomes current.

LIST: Prints the interval. If no interval, prints the current line. The last line actually printed becomes current. "I", interpret, is the only meaningful mode (see MODE Semantics). Modes appearing with the LIST command are temporary and do not disturb the permanent modes set by the MODE command.

MODE: The editor executes certain commands in different ways depending on a set of internal state variables called modes. A permanent set of modes are always in effect and can be set and reset by the MODE command. The permanent modes can be over-ruled for the duration of one command with temporary modes as in EDIT, SUBSTITUTE, and LIST. Basically, there are five modes:

1) A (after): if on, causes lines being affected by EDIT and SUBSTITUTE to be printed after the operation is complete.
2) B (before): if on, causes lines being affected by EDIT and SUBSTITUTE to be printed before the operation commences.
3) C (confirm): if on, causes lines being affected by SUBSTITUTE to be printed and requests confirmation prior to actual substitution. Permission to SUBSTITUTE is granted by typing "Y" (yes) and denied by typing "N" (no).
4) I (interpret): if on, control-L will print as code 154_8 (in phase 2 it will cause a page eject); otherwise, "control-L" prints as "&L".
5) integer: the value of the integer determines the maximum number of SUBSTITUTEs which can occur. If more than this value are attempted, a message will be printed which indicates the SUBSTITUTE command was limited to this number.

One additional letter may appear with modes: "N". When "N" is encountered, all alphabetic modes following it are reset. Initially, A, B, C, and I are reset (off) and the substitution limit is set to 50. Thus, "MODE 1ØØINABC" sets the substitution limit to 1ØØ, sets Interpret, and resets After, Before, and Confirm.

NEXT: Prints the next N lines to the current line just as LIST would, where N is the number following the NEXT command. If N is omitted, the next line is printed. The last line printed becomes the current line.

PREVIOUS: Prints the previous N lines to the current line just as LIST would, where N is the number following the PREVIOUS command. If N is omitted, the previous line is printed. The last line printed becomes the current line.

READ: Reads the file and INSERTs it before the addressed line. The last line read becomes the current line. If no address is specified, the file is APPENDed to the end of the entire program. The file name must be either surrounded by single quotes or terminated by a blank or end of line.

SUBSTITUTE: Searches the interval for occurrences of the second set of tokens and SUBSTITUTEs the first set of tokens for each occurrence. If no interval is specified, the current line is used. The last line having a substitution made in it becomes the current line. Note that the second set of tokens will match regardless of spacing. That is, /␣C-A/␣A␣+␣C␣/ will find a match in "X←A+␣␣C;". The first set of tokens is inserted exactly as stated. The replacement in the example will result in "X<␣C-A;". Simply stated, the first set of tokens is inserted in the line as if it were a string of characters. Comments may not appear in the first set of tokens. If the first set of tokens is null, the first set from the previous SUBSTITUTE will be used.

UNDO: This command will undo the deletion caused by the last CHANGE, DELETE, and EDIT; provided no commands affecting text have been executed since. Thus, if a grievous mistake has been made, the drudgery of restoring the old lines is alleviated. Note, however, that the new line insertions made by CHANGE and EDIT are not undone and therefore must be normally attended to. The old lines will be physically located, as a group, following the new lines. It is recommended that one not grow too accustomed to this command, as its usefulness will be compromised. Executing two UNDOs in a row <u>will not</u> restore the state which existed two changes previous. The current line is left unchanged.

VALUE: This command will print the editor address of the specified line in two forms. For example, "#4 = <>#57" would mean line #4 on the block containing the line and line #57 of the whole program. If no line is specified, the current line is used. The addressed line becomes the current line.

WRITE:	Writes the specified interval on the file. If no interval is specified, the entire program is written. The last line written will be the new current line.
text:	Basically, there are two ways of specifying text. The first is just a series of lines entered from the teletype under the control of the line editor. The old line is always the previous entered line except for the first line, which has a null old line. See the Appendix for a discussion of the line editor. The second method for specifying text allows one to use lines which are already in the program. The lines to use are specified by the interval. Additionally, if the "=" is present, the specified lines are deleted.
interval:	An interval specifies a group of one or more contiguous lines. The second address must have an absolute line number which is not less than the first. A block specified in the first address will be used for the second address if not overridden.
address:	A line address is composed of a starting point

(head or search) possibly followed by a series of line increments (tail).

head: This specifies a specific line. The current line is referenced as ".". The appearance of a block permits a line other than one in the current block to be specified. "$" will address the last line of the appropriate block. "#" followed by an integer will select the line numbered as the integer. The first line of a block is #1. A line may also be addressed by its label.

tail: This takes the line specified by head and increments forward or backward accordingly. The signed integer increments that number of lines. The search is explained below.

block: This defines the scope in which lines are addressed. The presence of a name confines the line selection to the block of the same name. If "<>" appears, the selection is over the entire program.

search: Searches are normally forward unless the "-" is present, in which case, they are backward by line. Associated with a search is a starting point and a scope. The starting point in the address syntax is the line adjacent to the current line if either no block or the unnamed block is specified; and the line adjacent to #1 if a specific block is named. The search is circular within the scope, looking at the first line after the last line if the search is forward, and vice versa if backward. The scope is defined by the semantics of block, if block is present. Otherwise, the current block defines the scope.

token:search: This will find the first occurance of the specified set of tokens and address that line. Spaces are ignored and SPL comments are illegal.

label: The only SPL labels recognized by the language editor are those which appear as the first token on a line followed by ":" on the same line.

<u>QUIT</u>

Typing QUIT while the language editor is in control will function as a break facility in phase 1. This can be used to safely terminate the current action. Actions which are QUITable are:

1) LIST, PREVIOUS, NEXT, and WRITE: will terminate after the line being processed when QUIT occurs.

2) APPEND, CHANGE, INSERT: will terminate after the text line, being processed when QUIT occurs, is completed.

3) SUBSTITUTE: will terminate after the substitution, being processed when QUIT occurs, is completed.

4) search: will terminate after the line, being searched when QUIT occurs, is found not to contain the searched for item.

RESTRICTIONS

Unfortunately, there are currently some problems associated with block boundaries. These are itemized as follows:

1) You cannot DELETE lines from more than one block at a time.
2) You cannot DELETE the first line of a block unless you DELETE the entire block.
3) You cannot EDIT the first line of a block.
4) You cannot CHANGE lines from more than one block at a time.
5) You cannot CHANGE the first line of a block unless you CHANGE the entire block.
6) You cannot introduce text which has SPL COMMON, PROGRAM, or END in it unless it will go in between already existing blocks--at the very end, or at the very beginning. This applies to APPEND, CHANGE, INSERT, and READ.
7) You cannot SUBSTITUTE a string which has SPL COMMON, PROGRAM, or END in them for anything.

APPENDIX

LINE EDITOR CONCEPTS

The line editor is the input interface between the teletype and M1CS. When one of the M1CS subsystems needs a line of teletype input, the line editor receives and retains control until the user is done composing a new line, at which time control and the entire new line are returned to the controlling subsystem.

Instead of typing in all the characters, the user may compose the new line by editing the "old line." The content of the old line is determined by the controlling subsystem and is usually the previous new line received by that subsystem.

Both the new line and old line have character pointers associated with them; initially these are set to the first character position. As characters are typed in from the keyboard, both character pointers are advanced. Thus, if the old string initially has "ABCDE" in it, the new string nothing, "XYZ" is typed and the editing facility is used to copy the next character from the old to the new string; the resultant new string will contain "XYZD".

The user communicates with the editing facility by typing control characters. In some instances, the editing facility also listens to one character following a control character (indicated by <u>C</u> below). The list below gives the different control characters and their resultant actions. A control character is typed by depressing the CTRL key while typing a normal character. Using control A as an example, control characters are signified as A^c. Note that normal character typing advances the pointers of both the old and new strings except during insert mode (between E^c brackets).

LINE EDITOR COMMANDS:

A^c	Backspace one character in new string and print "↑ ".
B^c	Print CR-LF and finish.
C^c	Copy one character from old string to new string and print copied character.
D^c	Copy rest of old line into new line and finish, printing copied characters and CR-LF.
E^c	Initiate and terminate insert mode, print "<" or ">". Characters typed after "<" and before ">" will not advance the old line character pointer.
F^c	No type version of D^c.
G^c	(Nothing)
H^c	Copy rest of old line into new line, printing copied characters. Just like D^c except CR-LF is not printed and line is not finished.
I^c	Insert spaces in new line up to next tab stop, printing them; advance old line that number of spaces. NOTE: if current character is in column #4 and tabs are 5 & 10, I^c will insert 5 spaces. The first tab is set at 8 and there-

after at five space increments (8, 13, 18, 23...63, 68).

J^c	Puts CR-LF into new string and continues to accept input after printing CR-LF.
K^c	(Nothing)
L^c	(Nothing)
M^c	Print CR-LF and finish.
N^c	Backspace one character in both old & new string and print "↑".
$O^c\underline{C}$	Copy characters <u>up to</u> $\underline{C}$ in old line into new line, printing; if the very next character is $\underline{C}$, the following one is used to terminate the copy.
$P^c\underline{C}$	Skip over characters in old line <u>up to</u> $\underline{C}$ printing "%" for each character; if very next character is $\underline{C}$, the following one is used to terminate the skip.
Q^c	Restart line anew, printing "←". Reset old and new string character pointers.
R^c	Retype unaligned by printing LF, rest of old line, CR, LF, all of new line; continue to accept input.
S^c	Skip one character in old line, print "%".
T^c	Retype aligned (like R^c) - Note, control characters printed by the "&C" convention will count as 1 character; thus, the number of characters unaligned indicates the number of control characters in the line.
U^c	Copy, to next tab, characters from the old line into the new line -- just like I^c only print copied characters instead of spaces.
$V^c\underline{C}$	Take $\underline{C}$ literally unless it is less than 100_8 in which

case add 100_8, print C, advance old line character pointer by 1.

W^c	Backspace new line to first blank preceding a non-blank. Thus, "ABC DEF W^c" will end up as "ABC " as will "ABC DW^c".
X^cC	Skip through C - like P^c.
Y^c	Concatenate new and old strings into old string and re-edit, print CR-LF.
Z^c	Copy through C, like O^c.