bcc	title: Preliminary Description of Compiler System Editor	prefix/class-number.revision: CSED/W-12
checked: [signature]	authors: L. Peter Deutsch	approval date: 6/20/69 / revision date:
checked: [signature]		classification: Working Paper
approved: [signature]		distribution: Company Private / pages: 11

ABSTRACT and CONTENTS

A preliminary description of the command language for the editor in the Model 1 SPL/Fortran Compiler System. Syntax is given for everything except line collection.

I. General Comments

This document constitutes a preliminary description of the editing language for the Model 1 Compiler System. While this editor might prove a reasonable starting point for a text editor, it does contain features specifically intended for editing SPL and Fortran programs and has limited capability for working on arbitrary strings of characters.

The editor accepts input of two distinct kinds. Commands arrive through the M1CS command processor, which prints a herald at the beginning of each line: * for normal editor command mode, : for brief editor command mode (see below for the distinction). As for the other command modes, the user composes commands using the standard line collector, which includes a QED-like line editing facility if the terminal is a teletype and a more modest line editing facility if it is a typewriter; the command is not acted on until the line is completed with carriage return (or the equivalents of control-D or control-F). Null or deleted lines result in no action beyond a new herald. A QUIT (rubout) is equivalent to a line-delete and is echoed in the same way, i.e., a special printing character followed by a new line and herald, as in all command modes. Also, if the first character of a command line is a command mode herald, that becomes the command mode immediately, starting with the remainder of the command line. The distinction between the two editor command modes is that in normal mode, any prefix of a command name is allowed, but it must be terminated by a blank or by the end

of the command line; in brief mode, only the first character of the command name must be typed, and no termination is expected. Every editor command begins with a different letter, so no confusion will arise. Finally, the "old line" for the line collector in command mode is always the previous command line, so that for example the user may obtain an exact repetition of the previous command by typing just the equivalent of control-F or control-D. For further discussion of the command processor, see the document on the MlCS Command Language.

The other kind of input to the editor from the terminal is text input. In this mode, there is no herald; the text being typed in constitutes an argument for the preceding command. Text input also works through the line collector: the "old line" may be a line of stored text (for the EDIT command) or an empty line (for the other commands requiring text input). A QUIT in this mode normally acts as a line delete but has no other effect; a second QUIT with no intervening typing will abort the command with a message. A QUIT at any other time, i.e., between the completion of a command line and the next time the editor expects input from the terminal, is ignored, except as noted under individual commands where an interruptive action has meaning.

The MlCS editor works on an encoded representation of the user's source program, called preprocessed text or PPT, rather than on the actual character strings. In the PPT form, each operator, symbol, or constant is represented by a

single item called a token. Thus any relation between the symbol AND and the letters (symbols) A, N, D is obscured. A further discussion of this question, and the exact algorithm used for dividing a statement into tokens, may be found in the SPL Reference Manual. As a result of this representation, operations implying the existence of the string form, such as string search or substitute, are slow since they require reconstruction of the string from the PPT. However, when one works in a compiler language, the existence of the string form is normally unimportant, and the standard search and substitute operations which work on tokens rather than characters are expected to prove satisfactory.

The two remaining sections of this document describe respectively the exact formats of parameters for the editor commands and the command repertoire.

II. Parameters

The fundamental concept of the editing language is the interval, which in turn is made up of 1 or 2 line addresses. A line address designates a particular line of text in the program being edited; an interval designates a group of lines (possibly only one line). At any moment, a specific line is current; this convention allows for more compact designation of operations on lines nearby. Each command makes a well-defined line current, as described under the individual commands. At any moment, a single block (function or COMMON block) is also considered current, namely the block containing the current line, and operations such as searches or addressing in general are normally limited to this block. By suitable addressing, however, the user can make another block current or perform edits on the program as a whole, by arranging for a line in another function to become current at the end of the edit.

A line address is composed of a head and a tail. The head specifies an initial address; the tail specifies a sequence of searches and increments to be performed starting at the address specified by the head. The head may be:

1) The current line, specified by a "." ;

2) The last line of the current function, specified by a "$";

3) A specified line of the current function, the first line being line 1, specified by "#" and the line number;

4) A line in the current function with a specified label, specified by the label (an identifier for SPL or a decimal integer for Fortran);

5) Any of 2, 3, or 4 for some other function, specified by prefixing the specification with a function name enclosed in "<" and ">";

6) Any of 2, 3, or 4 considering the entire program at once, specified by prefixing the specification with "◇".

The tail, optionally absent, is composed of an arbitrary sequence of elements of the following sort:

1) Increments, specified by a decimal integer possibly preceded by a + or a - sign.

2) Searches. These operate over the current function, or over the entire program in case 6 above; prefixing with a "-" indicates a backward search rather than forward. The search begins at the line specified by that part of the address which precedes the search itself. Label searches appearing in the head start at the current line in case 4, the first line of the function in case 5, and the first line of the program in case 6. A search consists of a label enclosed in ":"s, specifying a label search; a string of tokens enclosed in "/"s, specifying a search for that token string appearing anywhere in a line; or a string of arbitrary characters (not, of course, including the line editing characters or the enclosing character) enclosed in '"'s or "'"s, specifying a search for that string of characters appearing anywhere in a line. The last type of

search is slow and should be avoided. If the string of tokens or characters in a search is empty, the same string as was used in the last token or character search respectively is used again.

The exact syntax for line addresses follows:

```
interval = address ["," address]
address = heads tails / headn tailn
heads = "." / [function] "$"
function = "<" [name]">"
name = letter $(letter / digit)
k = 1$ digit
headn = [function] ("#" k / label)
label = k / name
tails = [k] tailn
tailn = $(("+" / "-") k / ["-"] search [k])
search = ":" [label] ":" / "/" [tokens] "/" /
     "'" [string] "'" / '"' [string] '"'
tokens = <a string of tokens excluding :>
string = < a string of characters as described
     above>
```

The argument which specifies the strings for the SUBSTITUTE command resembles a search, namely:

```
subspec = ["-"] "/" [tokens] "/"[tokens] "/" /
    "'" [string] "'" [string] "'" /'"' [string]
        '#'   [string] '"'
```

The second string, the one being substituted for, enjoys the same status as a search; if empty, the last token or

character string searched or substituted for is used. An empty string in the first place, the one being substituted in, has no special significance

When a command expects text as input, the user may supply the text in one of three ways:

1) A single line of typed input, supplied on the line following the command line, if the command is terminated with a semicolon just before the end-of-line;

2) An interval from anywhere in the program, supplied within [] following the other arguments of the command on the command line; if the [is preceded by a =, the interval will be deleted as well;

3) An arbitrary number of lines of typed input, supplied following the command line and terminated by the equivalent of a control-D at the beginning of a line, if neither of the above formats is used; the number of lines expected is fixed for EDIT as the number of lines being edited, but is arbitrary for the other commands.

The exact syntax for text input follows:

text = textfrom / textin

textfrom = ["="] "[" interval "]" "¢"

textin = ";" "¢"textline / "¢" $ textline control

textline = ⟨a line of text⟩

control = ⟨a control-D or the equivalent⟩

III. List of Commands

"APPEND" [interval] text

Appends the text after the interval. If no interval is specified, appends after the current line. The last line appended becomes current; if no lines are supplied, the last line of the interval becomes current.

"CHANGE" [interval] text

Replaces the interval by the text. If no interval is specified, replaces the current line. The last line of the replacement becomes current; if no replacement lines are supplied, the line just before the first line of the interval becomes current.

"DELETE" [interval] "¢"

Deletes the interval. If no interval is specified, deletes the current line. The line before the interval becomes current.

"EDIT" [interval] textin

Line-edits the interval. Works like CHANGE except that the text must be supplied from the terminal, and the "old line" for line input is the corresponding existing line of text rather than an empty line. If no interval is specified, edits the current line. The last line of the interval becomes current. A QUIT during editing makes the last line edited current but does not undo the effect of the editing.

"INSERT" [interval] text

Inserts the text before the interval. If no interval is specified, inserts before the current line. The last line inserted becomes current; if no lines are supplied, the line before the first line of the interval becomes current.

"LIST" [interval] "¢"

Prints the interval. If no interval supplied, prints the current line. The last line actually printed becomes current; this is true also if the command is interrupted with a QUIT.

"MODES" $ modespec "¢"

Sets the default modes according to the modespecs. Does not affect which line is current.

"NEXT" [n] "¢"

Prints, as in LIST, the n lines following the current line. If no n is given, prints 1 line. The last line printed becomes current as in LIST. If n is negative, equivalent to PREVIOUS -n.

"PREVIOUS" [n] "¢"

Prints, as in LIST, the n lines preceding the current line. If no n is given, prints 1 line. The last line printed becomes current as in LIST. If n is negative, equivalent to NEXT -n.

"SUBSTITUTE" subspec [interval] "¢"

Substitutes in the interval according to subspec. If no interval is specified, substitutes only in the current line. The last line printed becomes current; if no lines were printed, the current line is not affected. The command may be interrupted with QUIT only when it is printing; the QUIT takes effect (if at all) when the line being printed is completed.

"TABS" (k $("," k) / text)

Sets simulated tabs at positions k, counting the left margin as position 1. If no positions were specified, tabs are set in all positions given by non-blank characters in the last line of the text; this can be used to advantage by putting a comment line into a program with characters at desired tab positions and using the [] method of specifying text. If the text consists of no lines, or the last line is empty, tabs are assumed to be at standard positions, namely 8+5k for k=0,1,...

"UNDO" "¢"

Undoes the previous command which affect the text. The exact set of commands which can be undone is not entirely specified as yet; it will probably include all DELETEs and all other commands which involve changing less than some particular number of lines (like 10).

"XCHANGE" interval ["[" interval "]"]

Exchanges the two intervals. If the second

interval is not specified, exchanges with the current line. The line which replaces the last line of the first interval becomes current.

There will be additional commands in the editor for communicating with the file system. The 940 implementation will include the two given below.

"READ" file [interval]"¢"

Reads the file and appends it at the end of the interval. The last line read in becomes current. If no interval is specified, appends at the end of the program. In the 940 implementation, the file name must be either surrounded by single-quotes or terminated by a blank or end of line.

"WRITE" file [interval] "¢"

Writes the interval on the file. If no interval is specified, writes the entire program. The current line is not affected. Specification of the file is the same as for READ.