MSPM SECTION BZ.10.06

Identification

APL Editor

Purpose

The APL editor is the component which mechanizes function definition mode.

Introduction

The editor is a line-by-line editor. To the user it can be a context editor (having the same requests as the Multics EDM editor, in fact), or a line-number editor (having the same operation as the APL/360 editor). Internally, the source text to be edited is maintained as a doubly-threaded list of source lines (each terminated with the NL character). Various requests require old lines to be deleted or create new lines to be inserted or to replace old ones. These operations are managed by re-threading the line list.

This document does not treat the mechanization of the editing requests, inasmuch as those are almost identical to EDM (however, the LOCATE and FIND requests must be done line-by-line, since the source lines are not contiguous as in EDM). However, some important points regarding the interface of this editor to the rest of APL are covered.

Initiation of Edit Mode

Initiation of edit mode in APL must be for a specific function name. The name is inspected to see if it refers to a global variable, a group, or a function. If it refers to a global variable or a group, the request to initiate edit mode fails. If it refers to a function, the state indicator stack is searched to verify that the function is not pendent (otherwise, edit mode will not be entered), and then the existing source lines of the function are located to be edited. If the name is none of the above, no initial lines will be present, and a new function bead will be created for the new function (see description of function segment data formats in MSPM BZ.10.05).

Insertion and Deletion of Lines

Insertion and deletion of lines is done directly on the function stored in the current workspace, no on a copy as with EDM (the current workspace itself is, in general, a copy of some stored workspace to begin with). The source lines are kept as a doubly-threaded list (again see MSPM BZ.10.05).

An existing function has stored with it in the workspace both the source lines and also the lexical analysis of the source lines. This is necessary to avoid the overhead of lexing a line each time it is interpreted. When a function is edited, some lines will be altered, which will invalidate their corresponding lex . At the close of the function definition, the editor must scan the function and present to the lexical analyzer those lines which must be lexed again. Occasionally, a line must be lexed again even if it was not changed during the edit (example: a terminating quote is removed from a preceding line, causing the current line to become part of an alphabetic constant).

To determine which lines must be lexed again, the editor uses the line class flag, slb.class, which takes on the values EOF, CON, END, and NEW, for each source line. The class EOF is reserved for the list header bead and never occurs on a source line itself. The class NEW is used by the editor as a mark to indicate lines which need to be lexed. Stored functions contain only class CON and END, where CON is used for all source lines contributing to one lex line except the last, which is given class END.

When a line is deleted from a function, the line classes are treated as follows: if the line is class NEW, it is simply deleted. If the line is class END, all lines above it which are class CON are changed to class NEW, then it is deleted. If it is class CON, all lines above it which are class CON are changed to class NEW, as well as all lines below it to and including the first one of class END, then the line is deleted.

When a line is inserted into a function following another, the line classes are treated as follows: unless the line in the function is of class CON, the new line is simply inserted and given class NEW. If, however, the existing line is of class CON, then it and all lines above it of class CON are changed to class NEW. Then all lines below it to and including the first one of class END are changed to class NEW. Finally, the new line is inserted and given class NEW.

In each case above when a group of lines of class CON followed by one of class END were changed to class NEW, the editor frees the storage occupied by the lex of the altered line.

When edit mode is left, the editor will present each line marked NEW to the lexical analyzer to be lexed fresh. The lines will be marked CON and END, as appropriate. In addition, if the lexical analyzer continues to request input to complete a lex line when a group of NEW lines is exhausted, another group down to the next END will be marked NEW and presented to the lex. If the end of the function is reached by this procedure, the quotes do not match in the source, a diagnostic is issued, and edit mode remains in effect. (Until the user re-matches the quotes, he will be unable to leave definition mode.)

Termination of Edit Mode

When the user requests termination of function definition mode, the first operation performed is the lex of NEW lines, as discussed above.

Next, the syntax of the header line is checked (this is the only syntactic checking performed at edit time). The name of the function is inspected to see if it has changed. If so, it must not conflict with any global variable, group, or other function. If the header line does not exist, or if no line except the header line exists, the function is considered to be erased.

Next, the result, the arguments, the local variables, and the labels are all cross-checked against one another (by hashing them into a small table) to verify that there are no duplications in naming.

If all these checks succeed, the interpreter allows the return to execution mode. The usage bead corresponding to the old function is deallocated, and the new one is installed.

Reading and Writing Multics Files

The insert-file and write-file requests take as arguments Multics path names. The characters in the lines in question are transferred verbatim between the APL workspace and the Multics files. In particular, the character codes of such files must correspond to the APL internal codes (see MSPM BZ.10.04), or else such files will not constitute proper APL function definitions. In particular, the ANSCII escapes sequences listed in MSPM BZ.10.04 are not performed by editor requests; they are presumed to have been done already.