



(22) Date de dépôt/Filing Date: 1999/07/29

(41) Mise à la disp. pub./Open to Public Insp.: 2001/01/29

(45) Date de délivrance/Issue Date: 2004/12/28

(51) Cl.Int.⁶/Int.Cl.⁶ G06F 17/21

(72) Inventeurs/Inventors:

FARN, BRIAN, CA;
KNIGHT, CHRISTINE N., CA;
STORISTEANU, ADRIAN, CA;
BAKER, BRUCE R., CA

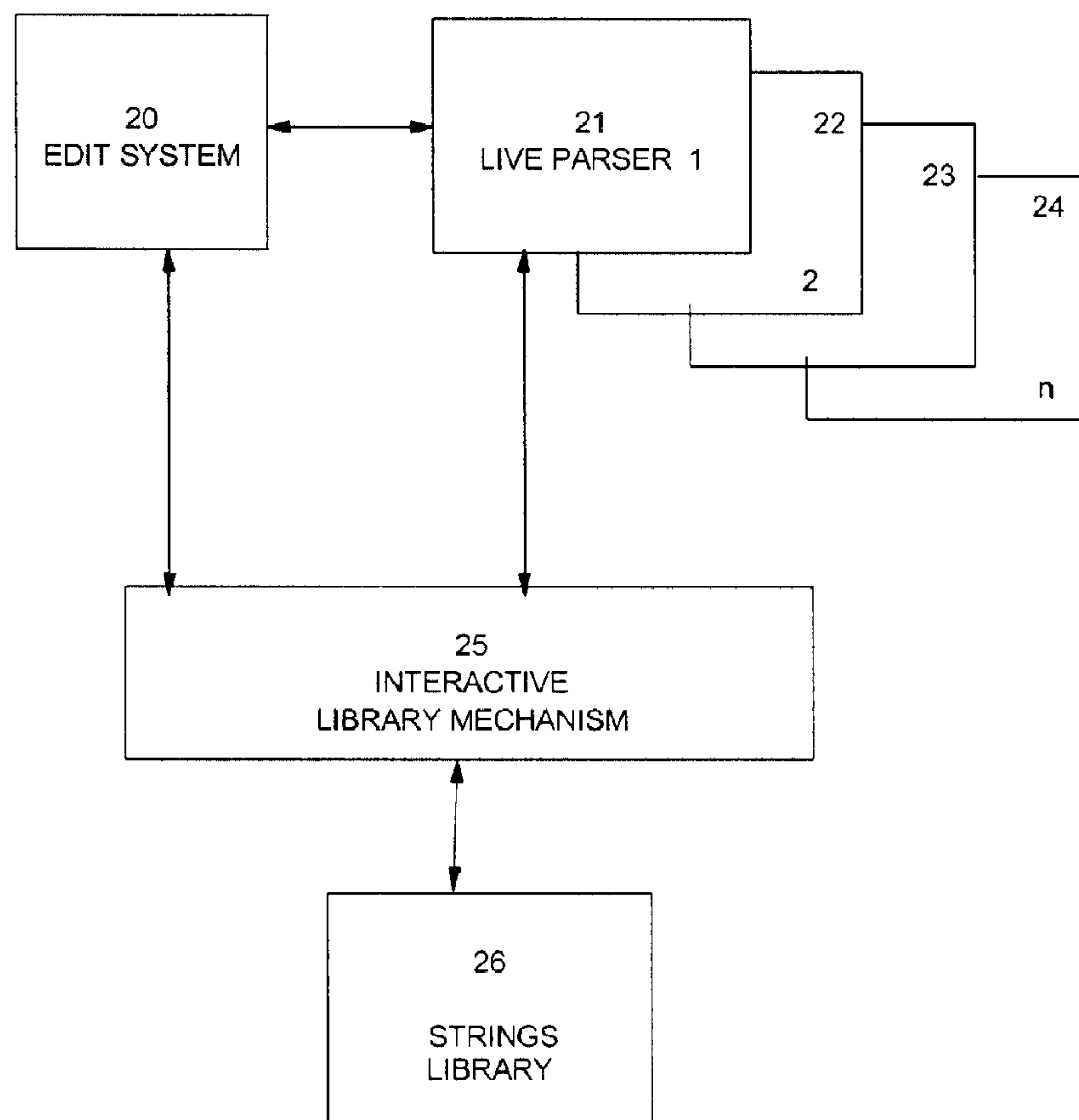
(73) Propriétaire/Owner:

IBM CANADA LIMITED-IBM CANADA LIMITEE, CA

(74) Agent: SAUNDERS, RAYMOND H.

(54) Titre : BIBLIOTHEQUE INTERACTIVE DE CHAINES DE TEXTE DANS UN SYSTEME D'EDITION

(54) Title: INTERACTIVE LIBRARY OF TEXT STRINGS IN AN EDIT SYSTEM



(57) Abrégé/Abstract:

A method and system are disclosed for storing and retrieving text strings for use in creating and editing documents. The disclosure provides an edit system that includes a function storing and retrieving frequently-used text strings, combined with producing

(57) **Abrégé(suite)/Abstract(continued):**

context-dependent alternatives to frequently-used text strings and generating context-sensitive document structures, in which a live parser parses the document to detect tokens relating to the text strings and to modify the text as required by the context in the document. Also disclosed is a computer-readable medium including computer program code that implements a system and method for storing and retrieving frequently-used text strings in conjunction with a live parser. Strings may be nested and may include commands and references to other strings.

CA9-98-046

INTERACTIVE LIBRARY OF TEXT STRINGS IN AN EDIT SYSTEM
ABSTRACT OF THE DISCLOSURE

A method and system are disclosed for storing and retrieving text strings for use in creating and editing documents. The disclosure provides an edit system that includes a function storing and retrieving frequently-used text strings, combined with producing context-dependent alternatives to frequently-used text strings and generating context-sensitive document structures, in which a live parser parses the document to detect tokens relating to the text strings and to modify the text as required by the context in the document. Also disclosed is a computer-readable medium including computer program code that implements a system and method for storing and retrieving frequently-used text strings in conjunction with a live parser. Strings may be nested and may include commands and references to other strings.

INTERACTIVE LIBRARY OF TEXT STRINGS IN AN EDIT SYSTEM

Field of the Invention

The present invention relates to operation of interactive edit systems, and more particularly to storage and reuse of frequently-used text strings in conjunction with a live parser.

5 Background of the Invention

In computerized information processing, edit systems allow a person to enter and modify text files. When editing a particular text file, the position where text will be inserted or replaced is indicated by a cursor, which may take any of several forms, including a one-character-wide block, a hairline or a caret. The position of a cursor can be changed by using keyboard
10 navigation keys, and within a graphical editor, by clicking a mouse button over the desired cursor position within an edit window. Edit systems also provide mechanisms for selecting a region of text either within a line, or spanning several lines, for a subsequent edit operation.

A live parser is an extension to an edit system. A live parser first gets control after a file has been loaded in the edit system, and subsequently it is called following certain edit system events,
15 such as changes made to the document. A live parser analyses and is aware during an edit session of the current structure of the document, for which it maintains information in the edit system such as element classes and text styles, and consequently is able to carry out various context-sensitive operations on the document. Among the tasks done by a typical live parser are detecting tokens in the programming language for which the parser is used, parsing language
20 constructs, and colorizing language constructs for programmer convenience.

In the context of an edit system, the repetitive typing of a text string is both tedious and error-prone. Several attempts have been made in the past to address the control of text strings in documents.

US Patent 5,050,071 to Harris et al describes a system for retrieving text strings from
25 applications external to the subject application, by using keywords in text files created by those applications. An essential feature of that system is the ability to launch those applications and search for desired text files. US Patent 5,644,775 to Thompson et al describes a system for

CA9-98-046

2

translating text associated with application programs, including string libraries. The patent is directed to a string reader and formatter which loads a desired library and retrieves the designated terms in the target language to replace the text entries in the original language. Other patents directed to string libraries storage and retrieval are US Patent 4,934,851, Sasaki, and US Patent 5,067,070, Miyao et al.

Despite these disclosures there remains a need for a system to store and retrieve text strings as needed by a person creating and editing large amounts of standardized text, where the edit system has knowledge of the context of the text strings involved and operates in real time as in an edit system using a live parser.

10 **Summary of the Invention**

Accordingly, the present invention provides a method and system for storing and retrieving text strings for use in creating and editing documents. Further, the present invention provides a method and system for storing and retrieving frequently-used text strings, combined with producing context-dependent alternatives to frequently-used text strings and generating context-sensitive document structures, in an edit system which works in conjunction with a live parser. The invention also provides a computer-readable medium including computer program code that provides a system for storing and retrieving frequently-used text strings in conjunction with a live parser. The invention also provides a computer-readable medium including computer program code that implements a method for storing and retrieving frequently-used text strings in conjunction with a live parser.

Brief Description of the Drawings

A preferred embodiment of the invention will now be described by reference to the accompanying drawings, in which:

Figure 1 is a flow diagram of a method of storing and retrieving text elements according to a preferred embodiment of the invention.

Figure 2 is a block diagram illustrating components of a preferred embodiment of the invention.

CA9-98-046

3

Figure 3 is a block diagram illustrating components of a typical edit system adapted to implement the invention.

Figure 4 illustrates a screen representation of an edit view of a Java source code file following activation of a pop-up menu containing functions according to a preferred embodiment of the invention.

Figure 5 illustrates a screen representation of an edit view of the file of Figure 4 showing a document comment structure generated by a preferred embodiment of the invention after a selection of Document Method is made.

Figure 6 illustrates a screen representation of an edit view of a source code file in the C++ language, similar to Figure 4, according to a second preferred embodiment of the invention, using a C++ parser.

Figure 7 depicts a block diagram of control and data structures maintained by an edit system according to a preferred embodiment of the invention.

Figure 8a illustrates the interactive context of a string library according to a preferred embodiment of the invention, showing the response of the edit system to opening of a document.

Figure 8b illustrates the responses of a preferred embodiment of the invention to selecting of text in a request to store the text to a strings library.

Figure 8c illustrates the responses of a preferred embodiment of the invention to requesting extraction of a string from a library.

Figure 8d illustrates the responses of a preferred embodiment of the invention to requesting an exit of the editor.

Description of the Preferred Embodiments

Referring first to Figure 1, illustrated is a block diagram of a computer system 10 incorporating the invention. The system includes a central processing unit 11, which performs operations on and interacts with a main system memory 13. The system further includes at least one mass storage device 12, which contains permanent files of information, including data created by the user and at least one library as discussed below. All components of the system communicate

CA9-98-046

4

over a system bus 18. The user interacts with the system using a keyboard 14 and a position-sensing device 15, for example a mouse. The output of either device can be used to designate information or select particular areas of a screen display 16 corresponding to functions to be performed by the system.

5 Figure 2 illustrates in block form elements of a computer system according to a preferred embodiment of the invention. Edit system 20 allows an operator to enter and modify text files. At least one live parser 21 is attached to the edit system. The live parser enables the real-time processing of changes to the text made by the programmer or operator. Parsing in general is breaking down all or some portion of the contents of a document into language constructs
10 recognizable by the parser that represent a structural feature of the document, for example tokens in programming language source code. Parsing also determines whether the arrangement of such tokens represents a valid structure of the document in the context of its type. Types of documents include program source code for a given programming language, database files, electronic mail notes. Live parsers do such parsing in real time, that is, as the operator is using
15 the edit system. The typical live parser is aware of the cursor position and of the context of the material that immediately precedes or follows the location of the cursor at any point during operation of the edit system 20. Figure 2 shows live parsers 1 to n (reference numbers 21, 22, 23 and 24). One of the live parsers 21 to 24 is in control after a file has been loaded into edit system 20 and the controlling live parser maintains information on the structure of the file, for example
20 element classes and text styles. The live parser that is in control is called by the edit system following loading the file to be edited into memory 13 (Figure 1), and following events made by the operator using edit system 20, for example a change in text in a document being edited. For the present description it will be assumed that live parser 21 is in control. Optionally a second live parser 22 can take control, for example if the edit system 20 encounters a control tag in the
25 document designating a second programming language. Following a second control tag indicating the end of text for the second programming language, control will revert to the original live parser 21.

Interactive library mechanism 25, which will be described in detail below, receives text strings and control codes from edit system 20 with respect to a document being edited. The overall

CA9-98-046

5

function of interactive library mechanism 25 is to provide saving, listing, and retrieving of text strings to and from documents being edited. Interactive library mechanism 25 is aware, because of messages received from live parser 21, of the current cursor location in a document being edited. Thus a string to be retrieved into a document will be inserted at the cursor position as is known in the art. Interactive library mechanism 25 stores to and retrieves from strings library 26, text strings that are selected by an operator. Selection of text to be stored from a screen display can be performed by any of the selection means known to the skilled person, for example by moving a pointing device, for example a mouse, over text to be selected. Selection of text strings to be retrieved from strings library 26 can be performed by selecting from a nickname list in a menu, for example a pop-up or pull-down menu, as will be described in more detail below, or by entering a keyword in text displayed on-screen, by engaging an action key, for example Control-R, or by any convenient means. Preferably, the system of the invention is used with one strings library 26 for each type of document; however if desired, the user may select a specific library for an individual document by means of a selection dialogue box by techniques known in the art.

Referring to Figure 3, a brief high-level description of the main components of a typical edit system 20 are provided below with particular reference to the implementation of the subject invention. The blocks shown represent the functions performed by the code in edit system 20 to support the subject invention. Edit control 31 is responsible for the instantiation and management of the various components of the edit system. Edit control 31 processes the edit system internal commands. Edit control 31 also handles the binding of internal commands or external commands, utilities and edit macros to menu items, key actions, mouse actions, etc. When an action is received, edit control 31 invokes the command bound to it, if any. Text processing 32 handles the loading of documents into the edit system's Edit Buffer, all the text editing operations and the formatting and reformatting of the text, and the saving of the documents. Changes to the document are recorded into a stack for subsequent processing by any live parser registered in the edit system for the document. Graphical user interface control 33 controls edit views of the document as exemplified in Figures 4 to 6 below, and builds and renders the edit views. It manages display attributes of elements rendered in the display as set up by the edit system 30 and its attached tools, for example text selection, or tokenization and

colorization of the text by the live parser described with respect to Figure 2 above. Commands and macros interface 34 controls interfaces to internal and external commands and edit macros 35, including live incremental parsers. Commands and macros interface 34 loads and runs external commands and macros. Commands and macros are well known to the person skilled in the art, and will not be further described herein. Commands and macros interface 34 also controls interfaces to system commands 36. System commands are also well known to the person skilled in the art, and will not be further described herein.

Examples of Use by an Operator

Figures 4 to 6 represent a series of document images on a screen display illustrating a sample basic application of the invention.

Figure 4 shows an edit view of a Java programming-language source code file named `example.java`. This program source contains a few classes and their methods, three of which are visible on the screen: `charAt`, `getChars`, and `getBytes`. The Java language parser attached to the editor has tokenized this view, updated its menus with Java source specific actions, and initialized the pop-up menu for this view with actions specific to the active string library in use.

In Figure 4, the user has activated a pop-up menu related to the illustrated view of the document. The bottom four items in the menu are library menu items: "Document class" generates a documentation template by methods known in the art for the Java class in whose scope the cursor is located. "Document method" generates a documentation template for the Java method in whose scope the cursor is located. "Expand keyword" looks up the word at the cursor and expands into a previously-stored template. "Library" points to a submenu of selectable library strings.

Below is a description of the operation of this invention when the user selects the "Document method" pop-up menu item in Figure 4. The library string item consists of the nickname "Document method", and calls, upon selection, an edit system command `JavaParser doc`, which invokes the Java-language live parser with the "doc" argument.

A live parser as described in reference to Figure 2 locates the first text element in an edit view of the method in whose scope the cursor is located. For example, in Figure 4 the cursor is located

within the `getChars` method. If any such method is found, the live parser checks whether the method has been documented already. If the method has been documented, the live parser merely locates the cursor inside the existing documentation structure for the user to edit the document; otherwise the live parser generates a doc comment structure for it, located above it in the file, as described below with reference to Figure 5.

Referring now to Figure 5, a start-of-comment `/**` and a first, empty, document block text element `*` are inserted. The live parser, with the aid of information generated during the previous parsing of the file (i.e., element classes and text styles), analyses the details of the method signature (header), and generates appropriate documentation tags in accordance with the various attributes of the method: one `@param` documentation tag for each formal argument of the method, a `@return` tag for a typed method, and `@exception` tags for exceptions thrown by the method. Such documentation tags are known to the person skilled in the art.

The live parser then retrieves the library string nicknamed "author", if any exists, to add an `@author` tag, here containing the text "A. Programmer". The `@author` tag causes the text "A. Programmer" to be nested, that is, embedded, into the string being edited. Finally, the live parser closes the documentation block with the end-of-comment `*/`, and relocates the cursor inside it for the user to complete, for example to add specifics about the method and detail the parameters.

The indentation level of the added comment block is adjusted, and the new text elements inserted are tokenized as comments, by the live parser, which is called after the library operation "Document method" has completed.

Figure 6 illustrates that selection of the same library item in a C++ document will operate in a similar manner, and the live parser, here the C++ language parser, will insert a C++ comment block for the function. While the processing is similar, the reformatting is carried out in a manner specific to the file type, i.e., a C++ source file. The library string item in this instance consists of the nickname "Document function", and calls an edit system command `CppParser doc`, which invokes the C++ live parser with the same argument "doc".

Figure 7 illustrates detail of structural elements of edit system 20 and interactive library mechanism 25 shown in Figure 2. Edit control structure 71 contains global data for controlling

CA9-98-046

8

the edit system. Document control structure 72 contains information to control the operations on a document in the edit system. There are several of these controls created, one for each document. Edit buffer 77 is a linked list of all the elements for a particular document, namely element structure 73. Edit buffer 77 contains both the element structure and meta data for the document. Classes are defined for a document, that broadly classify element types in the document. Classes are used to filter out elements from the view, and are used by the parsers or external commands to interpret the edit buffer contents at various degree of detail. Each element can belong to one or more classes.

Element structure 73 contains text contents, formatting flags and display attributes, and class information. Text elements that make up the edit buffer 77 include the contents of the document loaded in the edit system. The display attributes for each character are usually set by a tokenizer function in a live parser attached to the edit system.

Library control structure 74 maintains the library items; it loads them in, updates them as necessary, and stores them to one or more string libraries (reference 26 in Figure 2).

Library item structure 75 holds all the information for one library item, for example nickname or text. A library item is a stored text string, which can be edited and resaved, or simply extracted for use in a document being edited under the control of edit control structure 71. The preferred embodiment includes multiple library items, thus multiple layers in the library item structure 75 are shown. Library buffer 78 includes library item structure 75, the individual stored text strings that make up the library items and preferably additional information as discussed in greater detail below.

View control structure 76 includes data related to the control of an edit view for one document in an edit system window. Typically there is one view control structure for each edit view of the document, thus multiple views are shown in the view control structure 76. For example, the data include display attributes defined for the view, screen fields for the document view consisting of the particular area of edit buffer 77 that is displayed and other data items that will be readily apparent to the person skilled in the art.

CA9-98-046

9

Referring again to Figure 2, one or more live parsers 21 to 24 may be attached to edit system 20 for a document. A parser first gets control after the file has been loaded in edit system 20 (initial total parse), and subsequently it is called by Text Processing following changes made to the document (incremental parse) either by the user through editing, or by an edit system internal or external command.

A live parser is an external command that parses the edit document. It sets display attributes for the text elements in the edit view according to the syntax of the document (token highlighting), and records classes in the edit system with information regarding various constructs found in the document (e.g., C-language function declarations). The information set by the parser is used by the edit control to display the document in a view, for example colorized, filtered sections of the source, or a bounded segment of the source, by other tools attached to the edit system 20 to obtain detailed information about the structure of the document in order to carry out context-sensitive actions, and by the parser itself as reference points for incremental parsing.

A live parser can carry out several operations. Total parse is an operation called by the edit system 20 after a document has been loaded in. Incremental parse is called by the edit system 20 after changes have been made to the document. Other parser operations include language-sensitive functions, for example auto-indent, which includes repositioning of cursor and of text newly inserted in the document; navigation between classes and tokens, which is smart navigation between structural elements of the document; pre-processing of a selection for the storing of library strings; and smart reformatting for the insertion of library strings.

The interactive library mechanism according to a preferred embodiment of the invention will be further described with reference to Figures 8a through 8d, which illustrate the interactive context of a string library.

As shown in Figure 8a, as a document is opened for editing at step 81, the string library appropriate to that document is read into the memory of the subject edit system at step 82.

With reference to Figure 8b, the function of selecting and saving text in step 83 is accomplished through any of the means available in the particular edit system. Such means include: incremental selection of an arbitrary stream of characters via a keystroke sequence, or the

holding and dragging of a mouse button; selection keystroke sequences, for example select line, select block, select stream; context-sensitive selection, also known as smart selection, for example the selection of a language token accomplished through the services of the live parser 21 as shown in Figure 2, attached to the edit system 20 of the invention.

- 5 A live context-sensitive parser can optionally be used. For example, in a source code editor editing a C++ source file, the parser attached to the editor tokenizes the source, and provides an additional set of selection operations, such as quick selection of an entire identifier, or a class member name.

- 10 After the desired text is selected, a pull-down menu choice, popup menu choice, and/or keystroke sequence is activated to store the selected text into the active library associated with the type of the document or file being processed.

If the type of the document being edited does not have a string library associated with it, a string library is created.

- 15 Optionally, post-selection processing of a string selected at step 83 for which a request to store has been issued at step 84 may be provided by the contextual parser or a dedicated editor extension in certain environments, specific to the document type being edited, to improve the reusability of the stored strings. This processing takes place at step 85 before the string is stored as an item in the library at step 86, and may include one or a combination of beneficial modifications to the stored string and settings that will be used when a text selection is extracted
- 20 from the library in a sequence of steps beginning at step 87 of Figure 8c, for example context-sensitive modifications to the selected string, including capitalization, or addition of a prefix or suffix. It may also include determination of the optimal entry point and editing mode to be activated following the extraction of the string from the library, including the cursor position, an optional change of the insert/replace mode, and an optional selection of a portion of the
- 25 inserted string.

When a function name is stored in a C++ context, or a method name is stored in a Java context, a pair of parentheses is added to the stored string so that, for example, the selected string:

CA9-98-046

11

ProjectInitializeContainer

becomes, after processing:

```
ProjectInitializeContainer ( )
                          "
```

5 After this pre-processing at step 85, the stored string is associated with a cursor positioning indicator that provides for the user, when the text is selected from the library at step 89 of Figure 8c and inserted into a document at step 90, that the cursor is repositioned in insert mode at the location indicated by ", for the user to type in arguments that are desired to include with the text string.

10 Similarly, parameters, sometimes called arguments, may be filled in, for example from header files available in the development environment, or from a set of templates, so that the selected string:

ProjectInitializeContainer

becomes after processing:

```
15 ProjectInitializeContainer (pItemsList)
                          " " " " " " " " " " " "
```

Referring now to Figure 8c, when a request to extract this text item is made of the system at step 89, the active library is presented to the user at step 88, for example by means of the display 16 described with respect to Figure 1. The argument string is then selected from the library at step 89, and inserted into the document. The argument string is selected, and the cursor is set to insert mode. The user can then type over the selected text as required.

20

Similarly, for square brackets added to array identifiers, the selected string:

ProjectCheckVector

becomes after processing:

```
25 ProjectCheckVector [ ]
                          "
```

Following insertion at step 90, the cursor, if not moved by the operator, will be positioned immediately after the inserted string, no text will be selected, and the cursor mode will not be changed.

CA9-98-046

12

If post-processing of the string being stored is provided in the active editing environment, the post-processing option can be turned off. The invention preferably follows the insertion step 90 with reformatting and live parsing at step 91.

Referring to Figure 8d, when the user exits the edit system at step 92 according to a preferred embodiment, the edit system causes an automatic store of each updated library at step 93.

For some development environments that have the capability to handle utilities, which are known to the person skilled in the art, a utility to scan header files and build an initial library of templates can be written. Such a utility can either stand alone, or work in conjunction with the edit system and a language parser, as will be readily understood by the skilled person.

Referring again to Figure 2, in the strings library 26 desirable information can be retained for each string item in the library. Optionally a nickname can be retained, that is, a nickname for the string, which will be used when the library items are visually displayed for extraction; optionally the string itself, which is the actual text string. Optionally, the user can specify a nickname inside the string to include the text of another string from the library, for example a copyright notice. In this manner, generic string libraries can be easily customized to generate expansions for any particular environment as desirable for the user. The cursor position relative to the beginning of the string may be retained, which will place the cursor at a desired position after the string insertion. Optionally, the insert/replace keyboard mode can be retained; this can then be enforced after the string extraction from the library and insertion in a document, in one of the usual modes where no indicator means to leave the mode as is being used by the operator at the time, where the mode indicator "insert" sets the keyboard mode to insert, and the indicator "replace" sets the system to replace mode.

Optionally one or more text selection indicators can be included, usually two positions relative to the beginning of the stored string delimiting the selection. Optionally, an editor command, either an internal editor command or an external command, can be stored.

Optionally, an indicator to suppress presentation may be used on library items that the user desires not to be presented for direct selection. Also, an indicator may be included to specify

CA9-98-046

13

which items in the library appear where on the menus for user selection. Either of these indicators can be included by means known in the art.

Although both nickname and string were described as optional above, one of a nickname or a string must be set. The operator may avoid inputting a string when an editor command generates
5 the string dynamically; if this is done, the nickname will be displayed for user selection.

The ability of a library string to include other text strings from the library and to invoke commands, expands the concept of a simple string into a dynamic, smart string.

Preferably, a new library item is inserted at the top of the list. The library is stored in a file which can be read from, and written to, for use in succeeding edit sessions, by means that are
10 known in the art. For platforms which provide profile, for example, *.ini, facilities, the profiling mechanism may be taken advantage of for this purpose.

The preferred embodiment of the invention includes several optional methods of displaying and inserting text. The active library can be displayed on request by using pull-down menus, a popup menu, or a combination box list. The person skilled in the art can readily adapt the use of any of
15 these display techniques as desired, within the scope of the invention. The items are presented as located in the library, the most recently used preferably being at the top, or optionally sorted. The currently used library item, either an added or an extracted item, is preferably repositioned to the top of the library by known techniques.

When an item is too large for a reasonable presentation, optionally a portion of the beginning of
20 the string and its end may be shown, with ellipsis marks "..." in between. For library items that have nicknames defined, the nickname will be displayed in the list, instead of the full text string, optionally in a recognizable font such as with bold or italic properties, which makes the item easily distinguishable from the other library items.

Optionally, known techniques of showing long strings, either partially shown, or identified by
25 their nicknames, can be used to view the entire string, for example a string can be displayed in a fly-over bubble.

CA9-98-046

14

Accelerator keys can be assigned to the more frequently-used insertion texts by methods known to those skilled in the art.

When the user selects an item from the library, preferably the text represented by that item is inserted at the cursor position, and the text entry point along with the optional text-selection and
5 insert/replace mode are set.

Preferably, the currently active formatting facilities in the edit system, such as indentation and spell-checking, are activated on the inserted text, and the incremental (live) parser is notified of the changes to the document.

The preferred embodiment includes means of maintaining the strings library. Preferably a
10 dialogue is provided to allow various maintenance operations on string libraries, for example: editing of the string items; addition of strings; removal of strings; merging contents of different libraries; defining nicknames for strings for improved usability; and associating libraries with document or file types.

Users may optionally also set a persistent value for the maximum number of strings they wish to
15 have in the library. When an item to be inserted exceeds this maximum, an item in the library is removed, based on an implementation-chosen criterion; for example, a particular implementation of the invention can be set to remove the item used least recently.

Optionally, the edit system can be built with predetermined initial libraries for various types of documents. In the case of programming editors, such libraries can be provided for each of the
20 programming languages supported (e.g., C++, Java, etc.), or development toolkits. A project-specific built library may be distributed to an entire team of developers as an initial string library. The user has the option to set a new active string library for an edit session, or to persistently associate a new string library with a particular file type.

The function `store library item` is a request by the user to store the text selection as a library
25 item. The parser is called to do language-sensitive pre-processing of the selected text that may be required.

CA9-98-046

15

```

    if text selection
        text := text-selection
        if parser registered for document
            call to pre-process selection
5         text := updated text selection
        create library item
        add to active library.

```

Insert library item is a function that is activated when the user selects a library item. After the insert operation completes, the edit system preferably reformats the newly-inserted text, for example by auto-indentation, and then activates incremental parsing. An example of

10 pseudo-code that accomplishes that activity is shown below:

Input: pItem = pointer to a library item.

```

    if pItem = null
        return.

15    // insert the text
    set a new undo checkpoint in edit system
    call insertText(pItem) // see below

    // set-up text entry point & text selection
    if pItem->cursor != null
20        set cursor position in document
    if pItem->sel1 != null and pItem->sel2 != null
        set text selection in document

    // set insert/replace mode
    if pItem->mode = insert
25        set insert mode
    else if pItem->mode = replace
        set replace mode

    // run editor command, if any defined in library item
    if pItem->command != null
30        call pItem->command.

```

As seen in the preceding exemplary code fragment, the function `insertText` is called to insert the text contents of a stored library item into a document. A string in a stored library item can consist of as little as a short character string to as much as numerous text lines separated by new-line control characters, plus commands and references to other stored library items.

CA9-98-046

16

Input: pItem = pointer to a library item.

```
// in no library item or no text content, return
if pItem = null
    return.
```

```
5  if pItem->string = null
    return.
```

```
// avoid infinite recursion - already expanding this item?
if pItem->expanding = true
    return.
```

```
10 pItem->expanding := true
```

```
// insert text in document & imbed other strings if needed
for all text lines in pItem->string
    parse line for "{..}" - other library strings to imbed
    for all strings to imbed found in line
15     insert text line up to that point
        pImbedItem := find string by nickname
        insertText(pImbedItem)
    insert rest of text line
```

```
pItem->expanding := false.
```

20 It will be seen by persons skilled in the art that the invention is valuable for improving the accuracy of text input, and consistency of wording, as well as saving time for users of the invention. In this context, the invention provides the following, through a general, flexible, extensible, and context-sensitive mechanism: Selection of an arbitrary string, and its saving into a persistent library of previously selected strings; display of the string library, or a subset of the

25 library to a user on demand; a method which allows a user to select one of the library's strings, have it inserted at the active entry point and smartly merged into the document; generation of context-dependent document structures; and maintenance of the contents of the library. Maintenance of library contents can include manual addition of strings, manual removal of strings, editing of the contents of strings, together with the ability to have them reference other

30 strings in the library and invoke edit system commands, merging contents of different libraries, defining nicknames for the strings for easier access, and associating libraries with document or file types. The system and method of the present invention are particularly useful in editing programming text, for example high level language source code.

The disclosed invention as embodied in a method, computer system and computer program

35 product, is particularly applicable to the live parsing extensible editor (LPEX) environment.

CA9-98-046

17

Those skilled in the relevant art will appreciate that the invention is suitable and relevant to various computer system environments, which are appropriately programmed in accordance with the features described.

The terms used herein are for description rather than limitation, and there is no intention to
5 exclude any equivalent of the features shown and described. It is to be recognized that modifications are possible within the scope of the invention as claimed.

CA9-98-046

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A method for storing and retrieving text strings during editing of a document in an edit system comprising a live parser, said method comprising:
 - 5 (a) parsing said document in said live parser;
 - (b) selecting a text string from text incorporated into said document;
 - (c) saving said text string in a persistent library, said text string item in said library comprising:
 - 10 (i) one of said text string, or a nickname for said text string; and
 - (ii) a current version of said text string after processing by said live parser;
 - (d) extracting a first library text string from said library; and
 - (e) placing said extracted library text string at a position in said document indicated by a cursor in said document.
2. A method as claimed in Claim 1, further comprising the step of pre-processing said string by
15 said live parser.
3. A method as claimed in Claim 1 or Claim 2, wherein said text string library further comprises at least one of:
 - (a) a cursor position variable;
 - (b) an insert mode variable;
 - 20 (c) a pair of text selection position variables;
 - (d) a reference to a second library text string in said library; and
 - (e) a command usable by the edit system directly or by an external utility.
4. A method as claimed in any one of Claims 1 to 3, further comprising the step of reformatting said document in said live parser according to the location of said cursor.

CA9-98-046

5. A method as claimed in any one of Claims 1 to 4, further comprising the steps of parsing said extracted library text string and placing the cursor for editing at a location determined by said parsing step.
6. A method as claimed in Claim 5, wherein said location is an opening parenthesis of method parameters in a method included in said extracted string.
7. A method for storing and retrieving text strings included in an object-oriented programming language source code document in an edit system comprising a live parser, said method comprising:
 - (a) parsing said document to detect tokens associated with source code of said language;
 - 10 (b) selecting a text string from text incorporated into said document;
 - (c) providing a menu of actions based on said parsed selected text string, wherein said menu comprises actions for:
 - (i) generating a documentation template for a class;
 - (ii) generating a documentation template for a method;
 - 15 (iii) expanding a selected word into an existing template; and
 - (iv) generating a list of existing library strings;
 - (d) storing a title or nickname of a selected text string in a strings library, said stored input text string further including at least one of:
 - (i) if said input text string includes a method, a variable for each parameter detected for
20 said method by said live parser;
 - (ii) a cursor position variable;
 - (iii) an insert mode variable;
 - (iv) a pair of text selection position variables;
 - (v) a reference to a second library text string in said library; and
 - 25 (vi) a command usable by the edit system directly or by an external utility;
 - (e) displaying titles or nicknames of existing stored library text strings for selection;

CA9-98-046

- (f) after selection of a stored library text string, extracting said selected stored library text string and inserting said extracted stored library text string into said document at a position indicated by a cursor; and
 - (g) after inserting said extracted stored library text string, placing said cursor at a location of a first opening parenthesis in said extracted stored library text string.
- 5
8. A method as claimed in any one of Claims 1 to 6, further comprising the steps of:
- (a) prior to extracting said stored library text string, editing said stored library text string, and
 - (b) saving said edited stored library text string in said strings library.
- 10 9. A method as claimed in any one of Claims 1 to 8, including at least a second strings library.
10. An edit system for storing and retrieving text strings into a document, said system comprising:
- (a) selection means to select a text string from text incorporated into a document;
 - (b) a live parser adapted to parse said document ;
 - 15 (c) a storage library adapted to store said selected text string, a nickname for said string, and required parameters for each token detected by said parser;
 - (d) a display for displaying a list of stored library text strings in said library;
 - (e) extraction means for extracting a selected stored library text string from said library; and
 - (f) an editor for placing said extracted library text string into a document being edited.
- 20 11. An edit system as claimed in Claim 10, said system further comprising cursor placement means to place said cursor at a predetermined point in said extracted text string.
12. An edit system as claimed in Claim 11, wherein said extracted text string is a programming method and said predetermined point is at an opening parenthesis defining parameters for said programming method.

CA9-98-046

13. An edit system for storing and retrieving text strings into a source code document for a programming language, said edit system including a live parser, said system comprising:

- (a) selection means to select a text string from text incorporated into a document;
- (b) a live parser to parse said document to detect tokens associated with source code of said language;
- (c) a menu of actions based on said parsed selected text string, wherein said menu comprises actions to implement:
 - (i) a documentation template generator for a class;
 - (ii) a documentation template generator for a method;
 - (iii) means for expanding a selected word into an existing template; and
 - (iv) a list generator for existing strings;
- (d) a storage library for storing a title or nickname of a selected text string in a strings library, said storage library further enabling storage of at least one of:
 - (i) if said selected text string includes a method, a variable for each parameter detected for said method by said live parser;
 - (ii) a cursor position variable;
 - (iii) an insert mode variable;
 - (iv) a pair of text selection position variables;
 - (v) a reference to a second library text string in said library; and
 - (vi) a command usable by the edit system directly or by an external utility;
- (e) a visual display for displaying titles or nicknames of existing stored library text strings for selection;
- (f) extraction means for extracting said selected string and inserting said extracted string into said document at a position indicated by a cursor; and
- (g) cursor placement means for placing said cursor at a location of a first opening parenthesis in said extracted string.

CA9-98-046

14. A computer-readable medium including computer program means tangibly embodying a computer executable program of instructions for directing a data processing system to implement the method of any one of Claims 1 to 9.

5 15. A computer-readable medium including computer program means tangibly embodying a computer executable program of instructions for directing a data processing system to implement the edit system of any of Claims 10 to 13.

FIG. 1

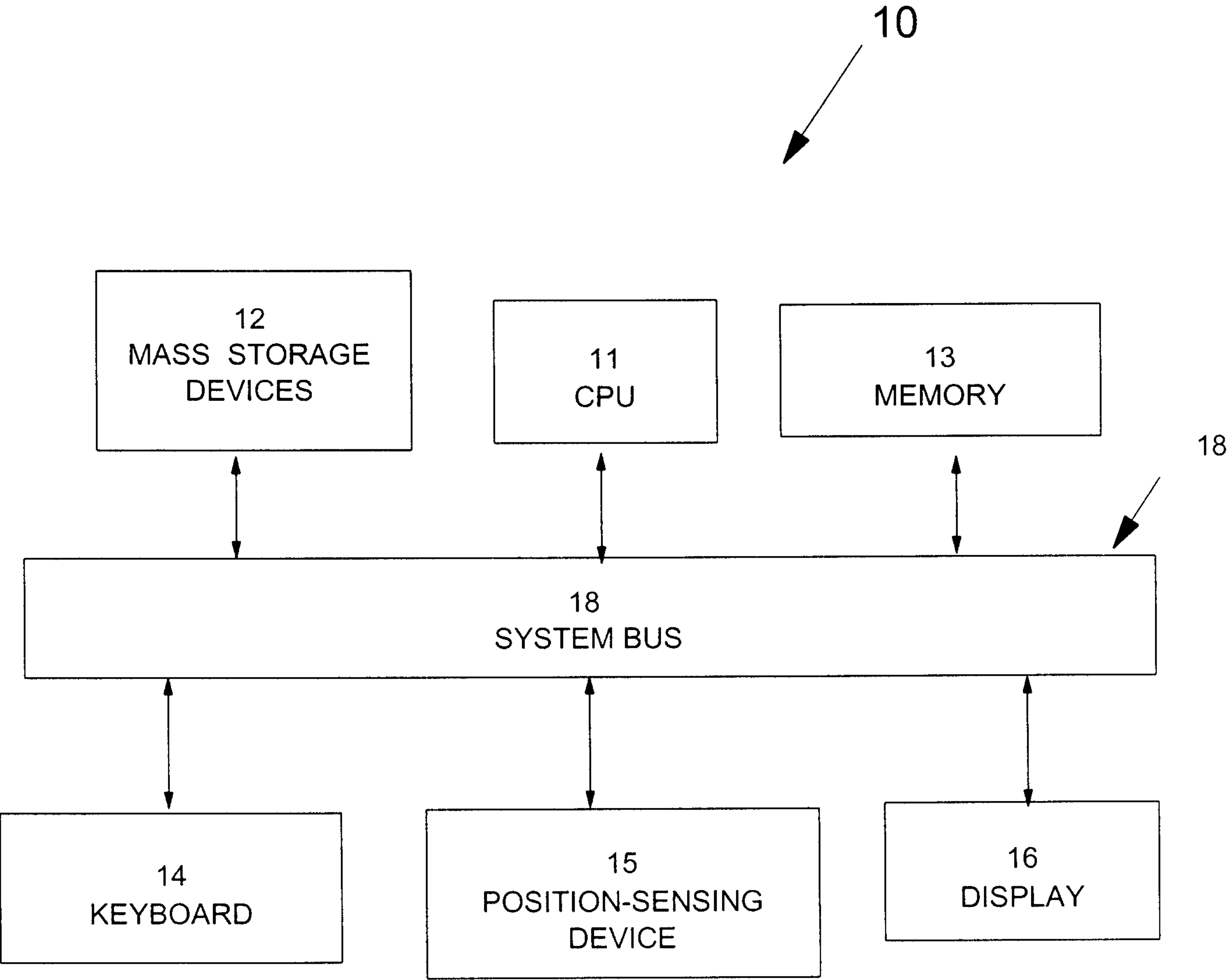


FIG. 2

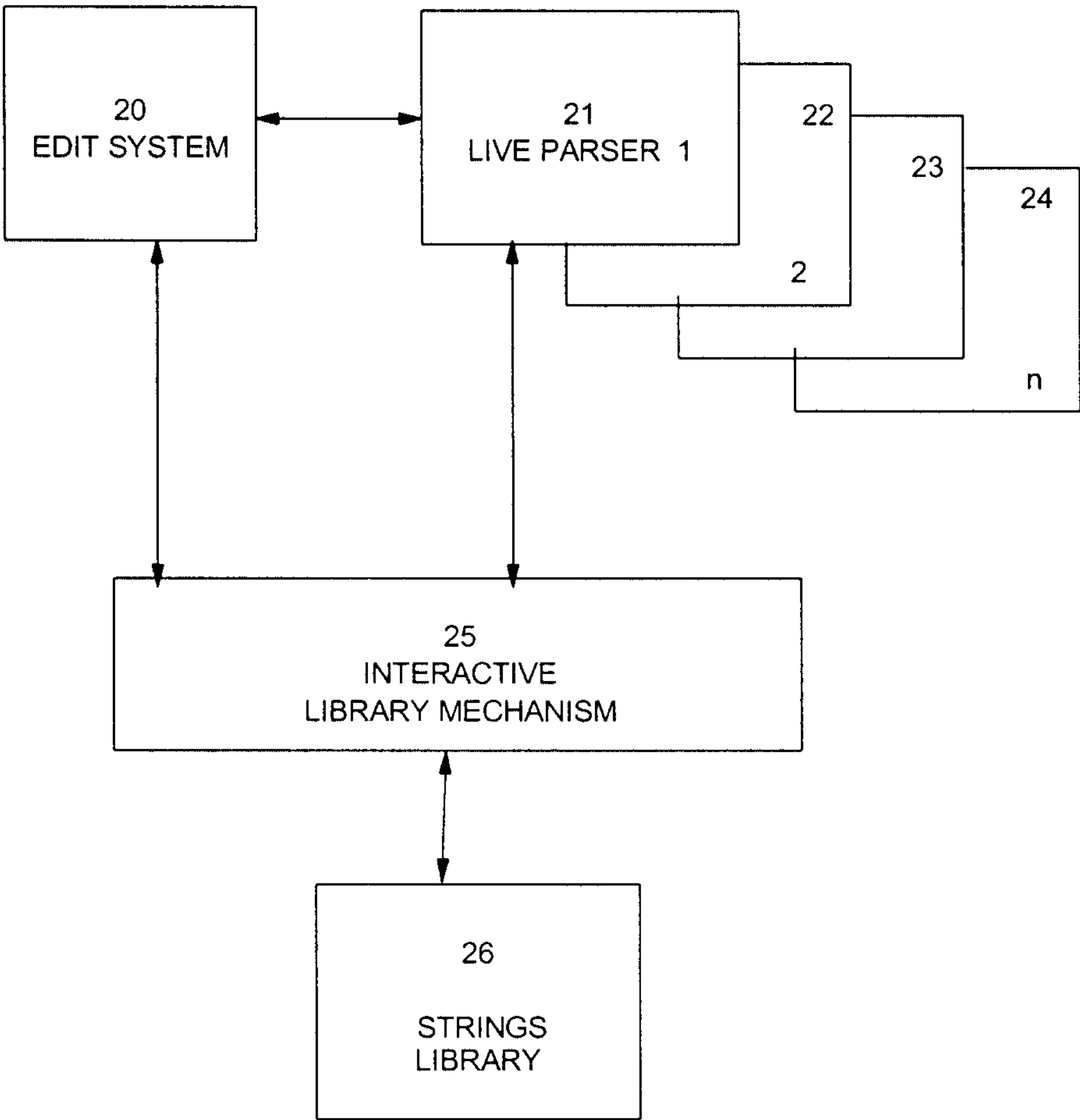


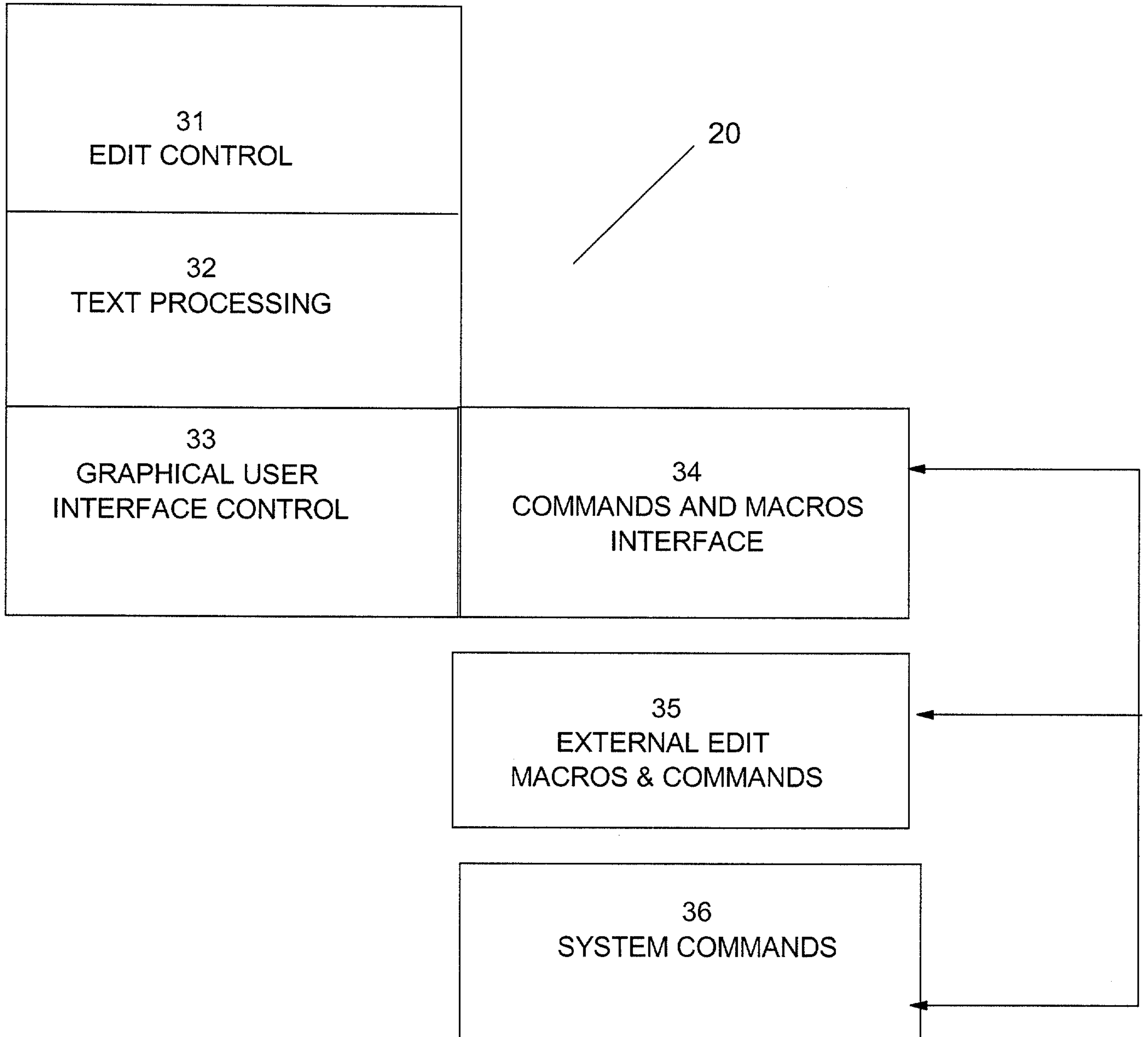
FIG. 3

FIG. 4

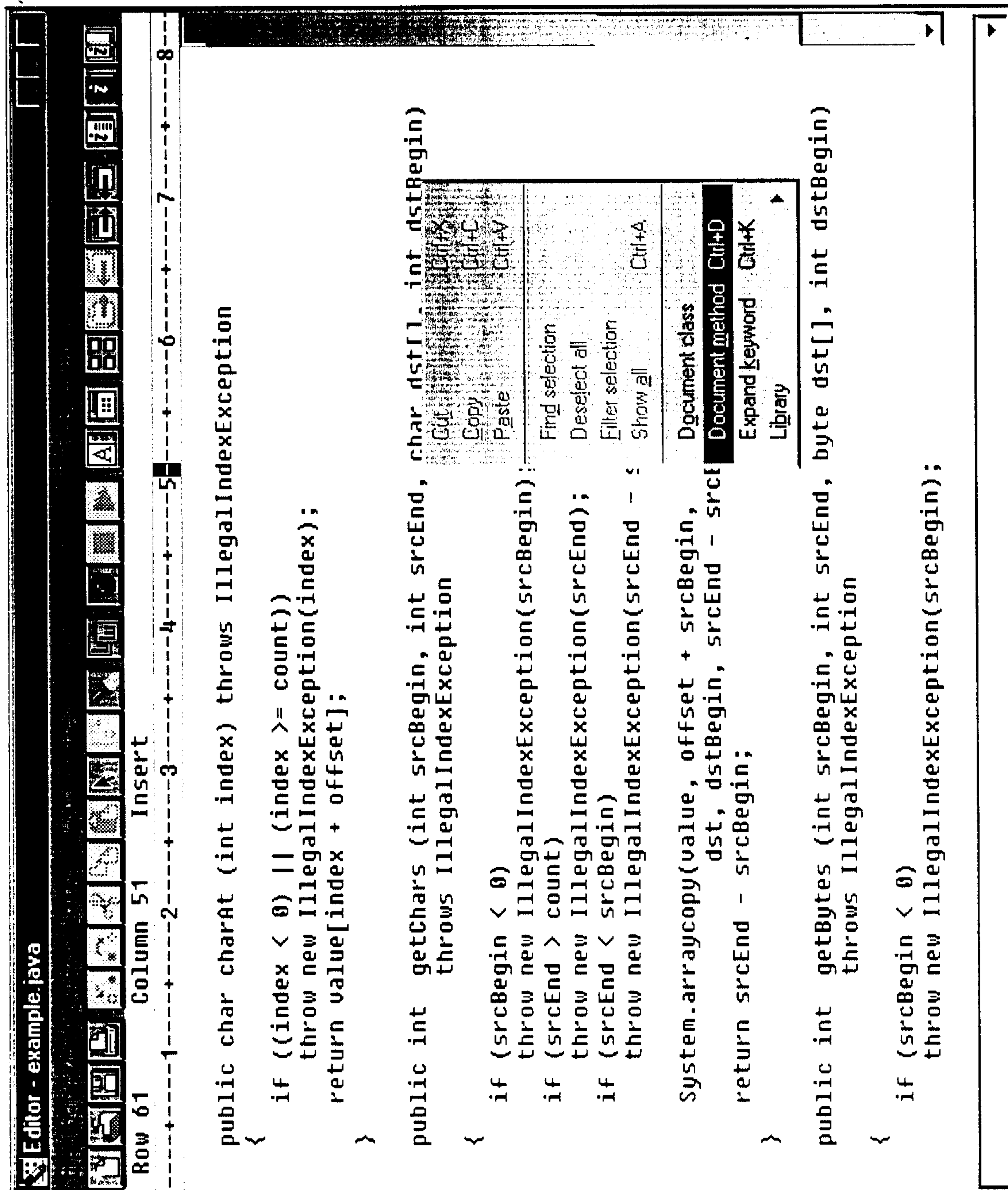


FIG. 5

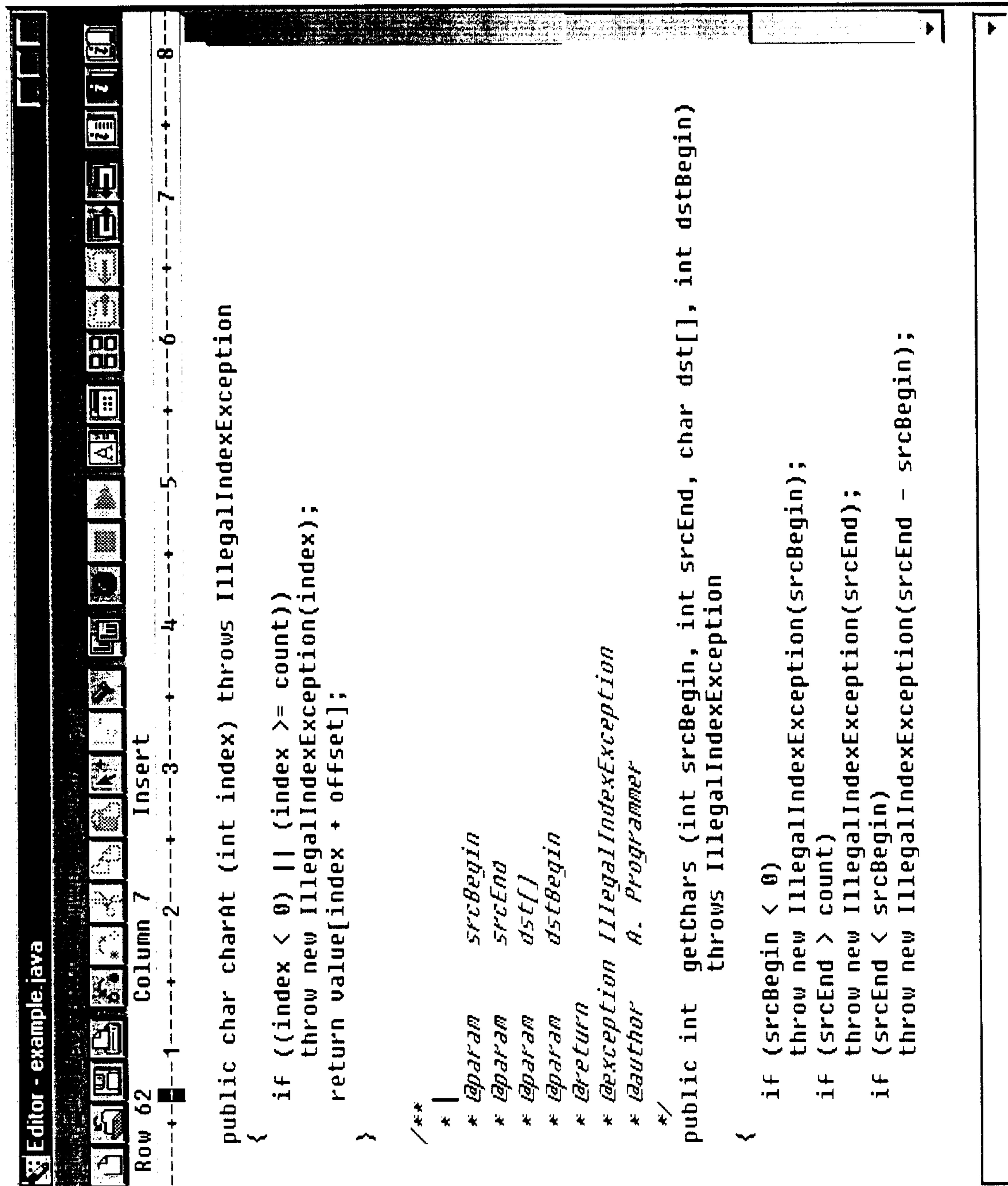


FIG. 6

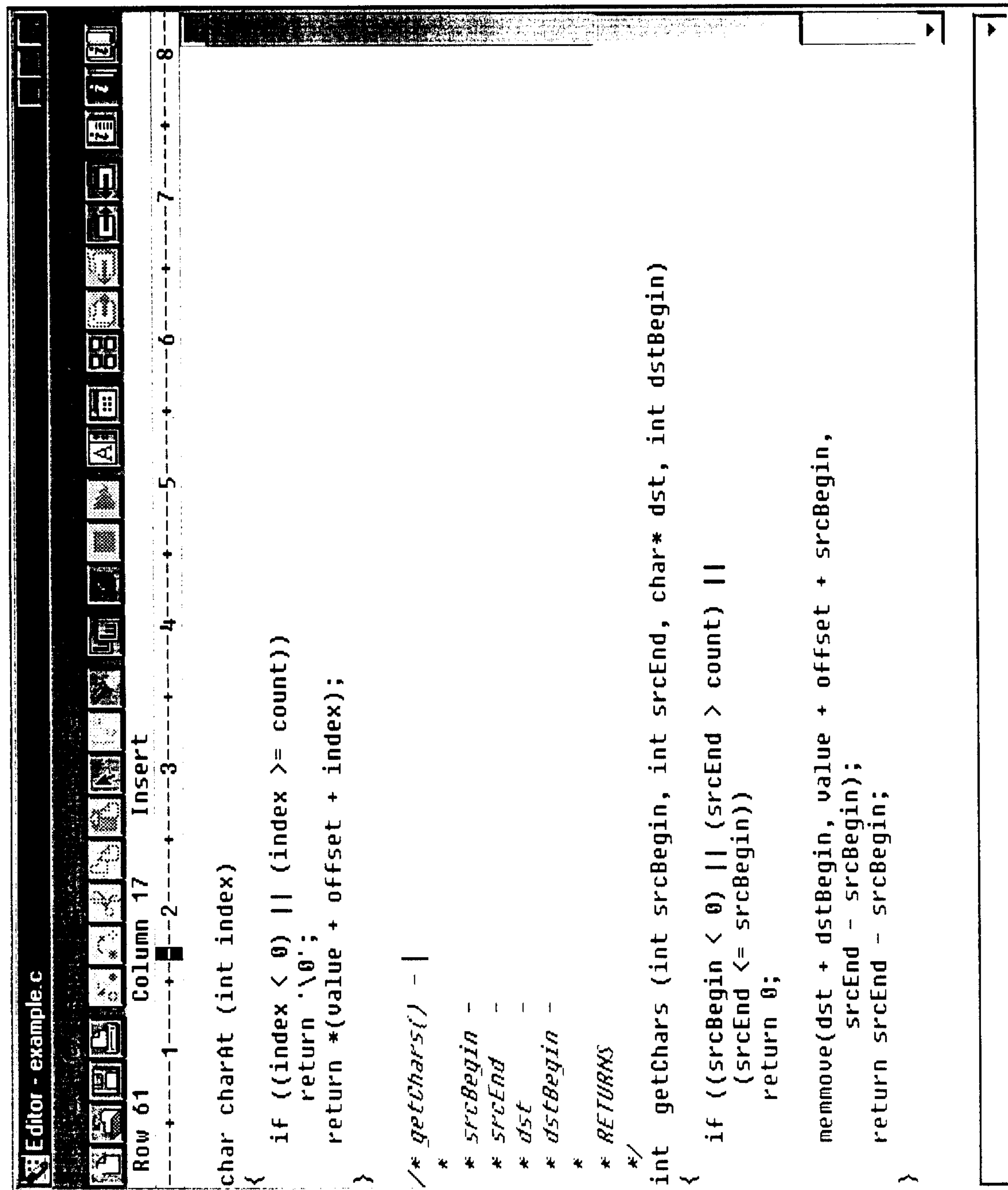


FIG. 7

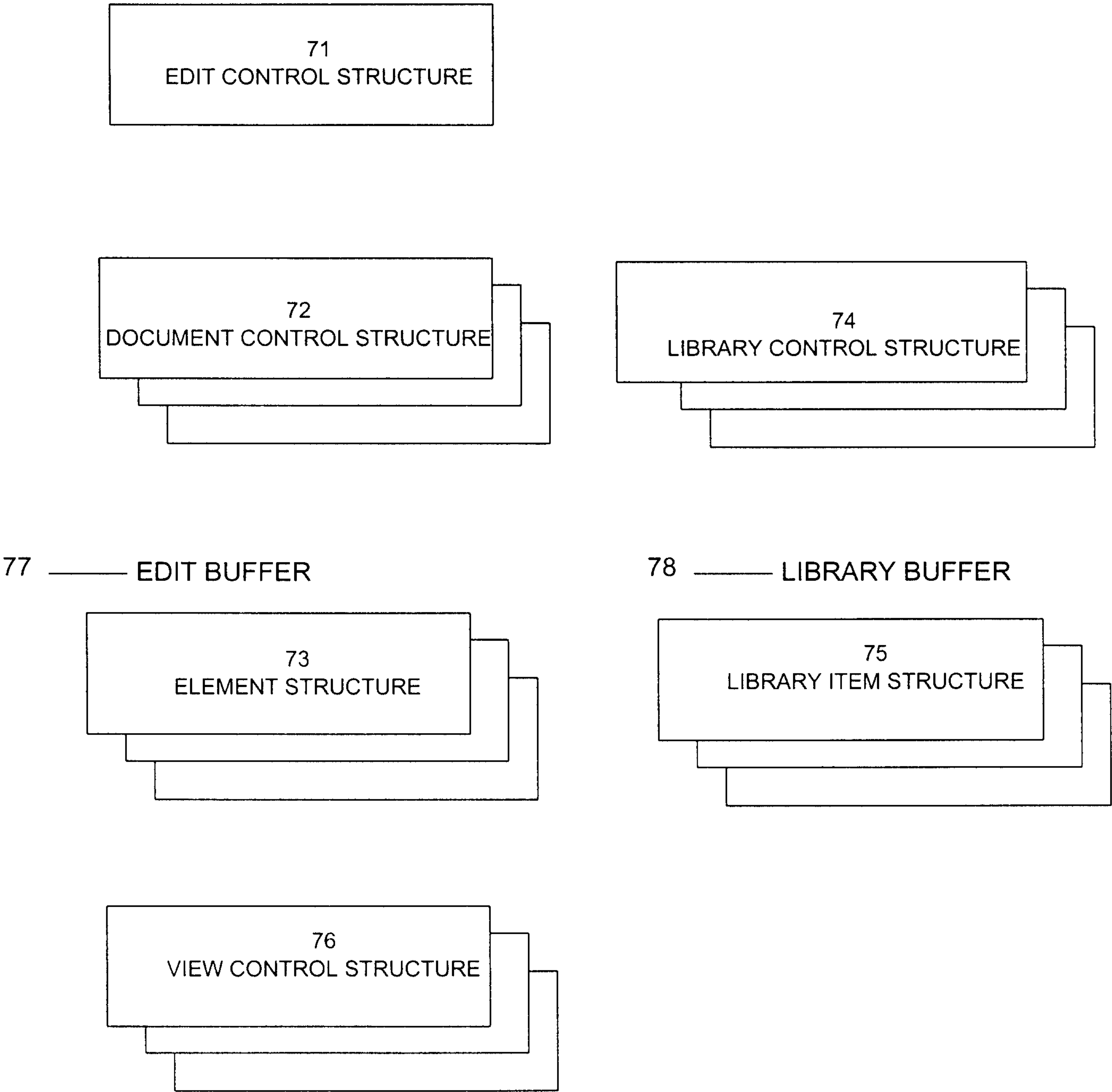


FIG. 8a

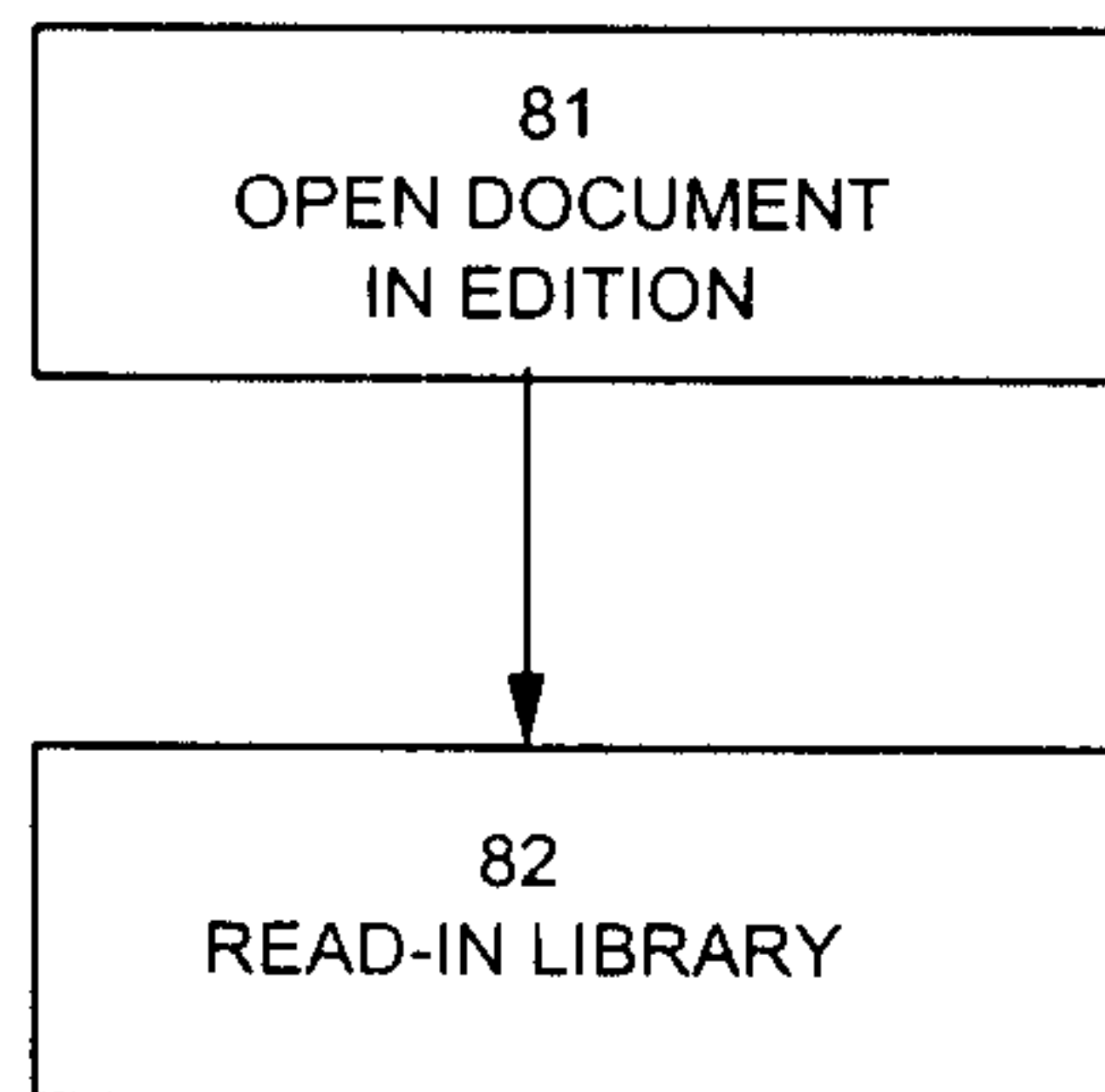


FIG. 8b

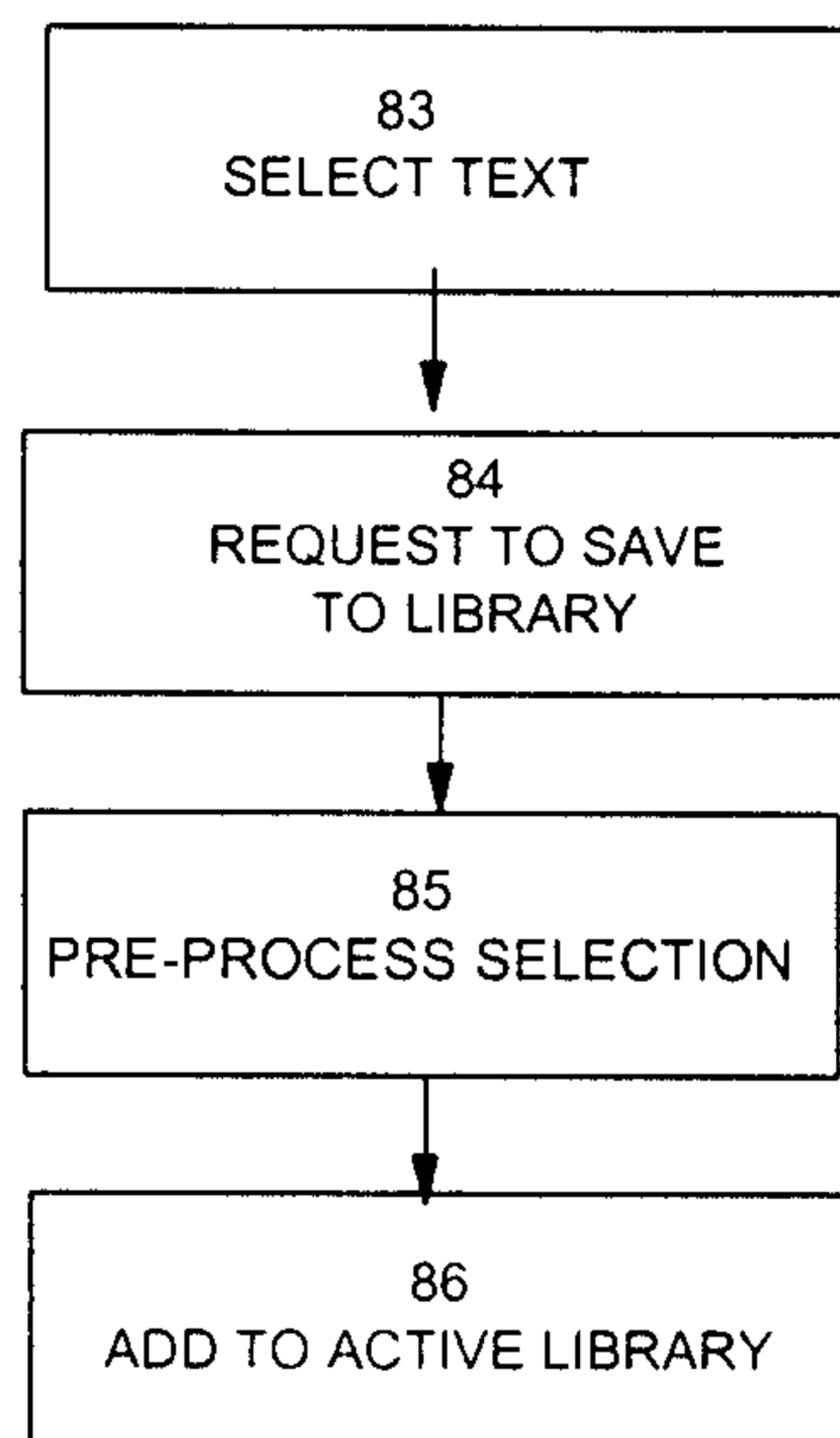
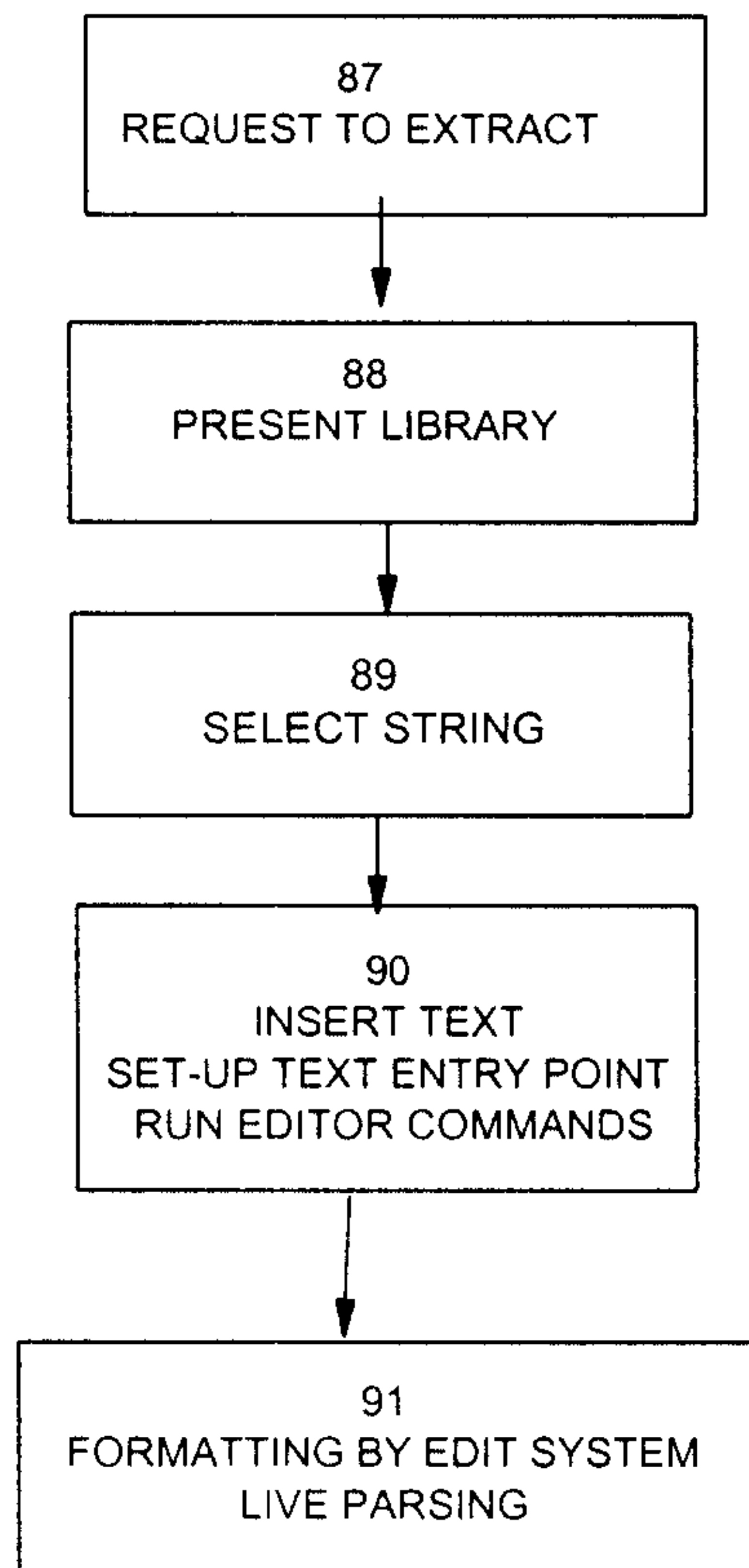


FIG. 8c**FIG. 8d**