Cosc 1P03 Assignment 2

(Due date Feb 18th 16:00 est, Late date Feb 21st 16:00 est)

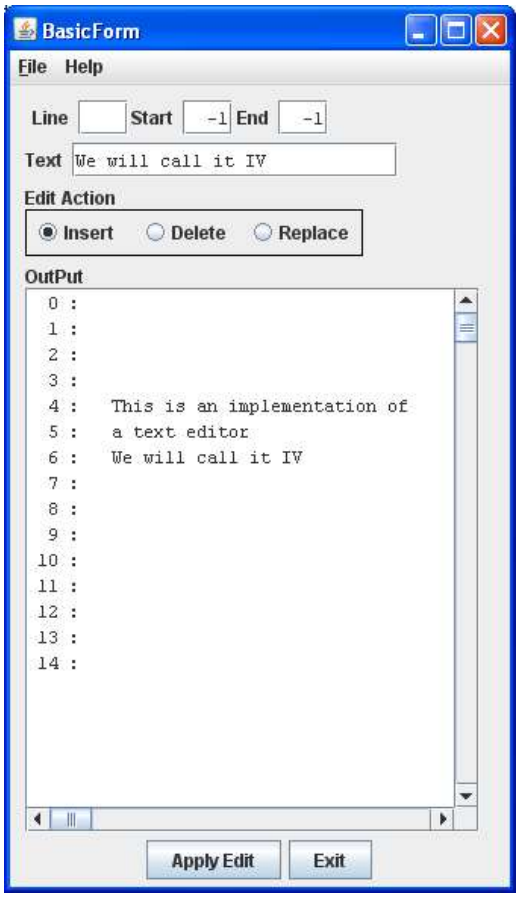
Back Ground

The need for text editors have been around from the first days of computing. In those days, graphical user interfaces have not been invented. As a result, programmers had to enter text using a primitive line editor. In effect, the programmer would specify which line of text they wanted to edit, by specifying a line number, and then inserting, deleting or replacing text within that line. This was a very cumbersome method to enter program text, but it worked, and for the time, that was all there was.

One of the most popular text editors of the time was VI (vee eye). Current versions of this editor are called VIM. VI ran in two modes, a command mode and an editing mode. It was completely keyboard based and did not require a mouse or any GUI systems. It could run on the most primitive text based terminals.

Fast Forward to Today.

We are not going to recreate VI, but program a simple text editor based on a similar concept. The editor will only implement a few basic operations which will allow the user to enter and edit text. We will call our editor IV (eye vee).



The Assignment

You are given the Basic Form to the right. There are only 3 commands, insert, delete and replace, selectable with the radio buttons. Variations of these commands can be derived by adding the parameters: Line, Start, End and Text. We will assume that the user will always enter valid commands within the specified behavior of the editor. Below are the commands and behavior which will need to be implemented. The parameters are abbreviated L, S, E, and T. Missing parameters below can be assumed not used, if S or E are not used a value of -1 is assumed. All text strings use 0 based indexing.

Insert L=valid line number, T=Some text String. Inserts a new line above L, all other lines are moved down 1 line. The text T is entered on the new line.

Insert L=valid line number, S= some index into the line, T=Some text String. Will insert T into line L before character index S. If S=0 we insert at front, if S is greater then the current length of line L we insert at the end, otherwise T is inserted before the character indexed by S. This allows insertion of new text anywhere in line L.

Delete L=valid line number. The line L is deleted; all lines after L are moved down.

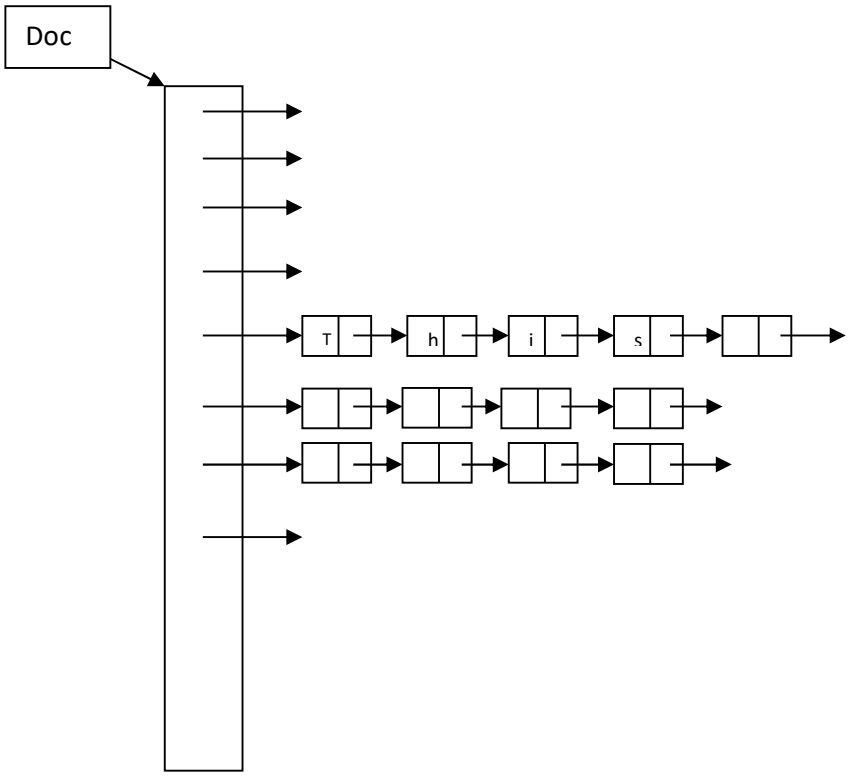
Delete L=valid line number, S=index of a character within the line. Deletes a single character from line L, indexed by S. If S indexes a character past the end of the line (S>line length), there is no effect.

Delete L=valid line number, S=start index, E=end index. Removes the characters from index S to E. If S>line length then no effect. If S indexes a valid character and E > line length, then all characters from S to EOL are removed.

Replace L=valid line number, S=start index, E=end index, T=text to insert. This is a hybrid command of delete L S E and insert L S T. Effect is to delete the characters from S to E and insert T before S.

These are 6 basic commands which can be used to edit a program. The user will need to physically count to obtain the S and E indexes, but otherwise quite usable. We won't worry about saving and loading a document (program file), this is not necessary for this assignment.

The assignment objective is to practice the mechanics of linear linked data structures. A document (what is being edited) will be represented in memory as an array of node pointers. These pointers represent list heads to linked lists of characters. Each node in the linked list will contain a primitive char which represent a character in the character text of the document, and a next pointer (used to link to the next node in the list). Below is an example of the data structure.



The length of the array is arbitrary; the size will determine how many lines the document can contain. For the purpose of this assignment set this array to at least 25 (declare this as a constant). Each node contains exactly 1 character. Each time a modification is made to the document, it is to be printed to the text area (Output), (see basic form example). Below is the form build method which you can freely use.

```
private void BuildForm(){
      String[] RadioLabel = {"Insert","Delete","Replace"};
      form = new BasicForm("Apply Edit","Exit");
      form.addTextField("L","Line",4,15,10);
      form.addTextField("S","Start",4,80,10);
      form.addTextField("E","End",4,150,10);
      form.addTextField("T","Text",30);
      form.addRadioButtons("action","Edit Action",false,RadioLabel);
      form.addTextArea("O","OutPut",20,40);
}
```

You will need at least 2 classes defined, a node class and a main class.

Your final implementation should conform to the assignment specifications. Deviation from the assignment specification will result in a significant loss of marks. Note that the purpose of the assignment is to practice linked list mechanics.

Where do I begin?

This is a linked list assignment, and thus will be the first time many of you have programmed using dynamic data structures. Your most common error will be, NullpointerExceptions. It is advisable that you program in small steps and test every method fully before commencing. Here is how I approached the development of this assignment.

1. Create a project with 2 classes, node class as specified and the main class. Both should be in a package called IV.
2. Write the methods to implement an Event loop which will drive the Basic Form, see 1p02. Test this with the given basic form.
3. Create the basic data structure, array to hold the document. Write a method to traverse this data structure and print it out to the Output field of the BasicForm each time Apply Edit is pressed. This is important since it will allow you to see what effects the editing commands had on your document.
4. Implement each of the commands in order, testing all cases for each to ensure behavior compliance.
5. Most importantly, start early. This assignment should not be started the day or night before.