Read Me File for Linelist Program

The goal of this program is to create a simple line editor that reads lines in a file and allows the user to add and delete lines from the file. The program uses linelist.cpp and linelist.h as the working files and client.cpp is the client file that the user is really interacting with. The client takes in a file name as an argument and edits that file. Once the user is done editing, the new lines are written and saved to the original file. The files are simple text files and the program limits itself to deleting lines and adding new lines.

# Design

The program uses the following libraries: iostream, fstream, cstdlib and cstdio. It also uses the linelist class, which was written for this program. The iostream library is used for asking for user input and displaying on the console for the user to see something, such as the “1> “ that is shown before each line while editing. The fstream library is used to read and write files. The cstdlib and cstdio are included to allow the use of the “exit(1)” function, in the case the user doesn’t type a filename when trying to execute the program.

The linelist class which is used in the program includes several private variables and public functions, which are used and implemented in the client, either directly or indirectly. The private variables of the linelist class are \*currLine of type LineNode, currLineNum of type int and length of type int. the currLine pointer is used to reference different LineNodes in the linelist when the user is using the program. currLineNum and length are used to support the functions in if statements and other logic. The LineList object consists of multiple LineNodes linked together to form a doublely linked list with a topBugger and bottomBuffer used to designate the beginning and end of the linelist.

The functions are all void, except for the getter methods for currLineNum, Length and currLine. The method goToTop sets currLine and currLineNum to reference the top of the list. The goToBottom function does the same but references the bottom of the linelist.

insertLine takes a string variable, which is used for the info the new LineNode, and the new LineNode is inserted after the current line being reference and before the next line. If the user is at the top of the list, then the new line is inserted at the top of the list as line 1, right after the topMessage buffer.

deleteLine takes no parameters and deletes the current line. After deleting the current line, the current line becomes the next line. If the user is at the end of the linelist, then the last line is delete and the current line becomes the new last line or the previous back of the deleted line.

MoveNextLine and movePrevLine allow the user to change the current line being referenced in the linelist. If the user is at the top of the list, then calling moveNextLine will not do anything because the user is already at the top of the linelist and cannot reference the top buffer. The same applies if the user is at the bottom of the list and tries to call movePrevLine.

printList is used to print the entire contents of the linelist excluding the top buffer and bottom buffer. This method is good for the user to check if a line they inserted or deleted was correct or not or see the current progress of their file before they exit and save.

# Algorithm Outline

1. Program is executed by calling the executable file followed by the filename of the file to edit
   1. If a filename is not enter, the program will exit with code exit(1)
2. Create a LineList based on the contents of the file
3. Print the LineList contents and set the current line to the last line
4. Continually ask for user input until “E” is enter
   1. If user enters I, prompt the user for a message for the new line, then insert the line after the current line. The current line is now the new line and the length has increased by 1.
   2. If user enters D, delete the current line, if any are available. If the user deletes the last line, then the current line will point to the new last line of the LineList and will not point to the bottom buffer.
   3. If the user enters P, move up one line. If the user is already at the top line, then don’t move up.
   4. If the user enters N, then move down one line. If the user is already at the bottom line, then don’t move down.
   5. If the user enters L, print the whole list, excluding the buffers. If the LineList is empty, then don’t print anything.
   6. If user enters E, exit the loop and finishing executing the program.
5. Save the contents of the LineList to the file that was originally specified by the user. Each NodeLine will have its own line in the file. Ignore the buffers and “close” the file.