# Data Structures
# Project 2: Line Text Editor (LTE)
# Assigned: September 15, Due: October 4 by 13:59:59

## 1 Description

Write a line text editor (LTE) using Java. The command syntax is described below. An internal copy of a file is maintained as a list of text lines (strings). In order to move freely up and down the file, the internal data structure is a doubly linked-list of lines (Java Strings). The linked-list methods you are to implement are described in this document. The editor commands are also described in this document.

The application will actually maintain two lists of lines. The first is the lines of text read from the file. The second list of lines is a "clipboard" used for cut-copy-paste operations.

### 1.1 Editor Commands

The following is a list of editor commands

| Command | Arguments | Description |
|---|---|---|
| `h` | | Display help |
| `r` | `filespec` | Read a file into the current buffer |
| `w` | | Write the current buffer to a file on disk |
| `f` | `filespec` | Change the name of the current buffer |
| `q` | | Quit the line editor |
| `q!` | | Quit the line editor without saving |
| `t` | | Go to the first line in the buffer |
| `b` | | Go to the last line in the buffer |
| `g` | `num` | Go to line num in the buffer |
| `-` | | Go to the previous line |
| `+` | | Go to the next line |
| `=` | | Print the current line number |
| `n` | | Toggle line number displayed |
| `#` | | Print the number of lines and characters in the buffer |
| `p` | | Print the current line |
| `pr` | `start stop` | Print several lines |
| `?` | `pattern` | Search backwards for a pattern |

| / | pattern | Search forwards for a pattern |
|---|---|---|
| s | text1 text2 | Substitute all occurrences of text1 with text2 on current line |
| sr | text1 text2 start stop | Substitute all occurrences of text1 with text2 between start and stop |
| d | | Delete the current line from buffer and copy into the clipboard (CUT) |
| dr | start stop | Delete several lines from buffer and copy into the clipboard (CUT) |
| c | | Copy current line into clipboard (COPY) |
| cr | start stop | Copy lines between start and stop into the clipboard (COPY) |
| pa | | Paste the contents of the clipboard above the current line (PASTE) |
| pb | | Paste the contents of the clipboard below the current line (PASTE) |
| ia | | Insert new lines of text above the current line until "." appears on its own line |
| ic | | Insert new lines of text at the current line until "." appears on its own line (REPLACE current line) |
| ib | | Insert new lines of text after the current line until "." appears on its own line |

## 1.2 Command Descriptions

**HELP (Command: `h`)**

This command will display a help screen in a tabular format similar to the command table listed above.

**READ FILE (command: `r filespec`)**

This command will read the contents of the file named at *filespec* and store it into the current buffer. If *filespec* does not exist, then print the message ``==>> FILE DOES NOT EXIST <<==`` on a separate line and then (go back and) prompt for the next input command. If the buffer already has lines of text, then clear the current buffer (delete all lines) before reading in the file.

**WRITE FILE (Command: `w filespec`)**

This command will write the contents of the current buffer into a file specified by the string *filespec*. If the buffer is empty, the print the message `''==>> BUFFER IS EMPTY <<==''` and do not create an empty file (0 bytes).

**CHANGE FILE (Buffer) NAME (Command: `f filespec`)**

This command will change the name of the current buffer to the name specified by the string *filespec*.

**QUIT (Command: `q`)**

This command will quit the line editor application. If lines in the current buffer have been modified, the quit command will prompt the user to save the buffer to a file before exiting.

**FORCE QUIT (Command: `q!`)**

This command will quit the line editor without saving the contents of buffer. If the current buffer has been modified, those changes will be lost.

**TOP (Command: `t`)**

This command will go to the first line in the buffer. If the buffer is empty, then the top command will display the message ‘‘`==>> BUFFER IS EMPTY <<==`’’.

**BOTTOM (Command: `b`)**

This command will go to the last line in the buffer. IF the buffer is empty, then the bottom command will display the message ‘‘`==>> BUFFER IS EMPTY <<==`’’.

**GOTO LINE (Command: `g num`)**

This command will go to line number specified by the argument *num*. The argument must be an integer and in the range [1..$n$] where $n$ is the number of lines in the buffer. If the line number is out of range, then print the error message ‘‘`==>> RANGE ERROR - num MUST BE [1..`$n$`] <<==`’’. Replace $n$ in the error message with the number of lines in the buffer.

**GOTO PREVIOUS LINE (Command: `-`)**

This command will go to the previous line from the current position in buffer. If the buffer is empty, then your editor should display the message ‘‘`==>> BUFFER IS EMPTY <<==`’’. If the buffer is not empty, and the current position is already at the top of the buffer (line 1), then display the message ‘‘`==>> ALREADY AT TOP OF BUFFER <<==`’’.

**GOTO NEXT LINE (Command: `+`)**

This command will go to the next line from the current position in buffer. If the buffer is empty, then your editor should display the message ‘‘`==>> BUFFER IS EMPTY <<==`’’. If the buffer is not empty, and the current position is already at the last line of the buffer (line $n$), then display the message ‘‘`==>> ALREADY AT BOTTOM OF BUFFER <<==`’’.

**TOGGLE LINE NUMBER (Command: `n`)**

This command will toggle the display of the line number when displaying the prompt and printed lines from the buffer.

**PRINT LINE (Command: `p`)**

This command will print the current line in the buffer to standard output. If the buffer is empty, then display the message ‘‘`==>> BUFFER IS EMPTY <<==`’’.

**PRINT RANGE (Command: `pr start stop`)**

This command will print a range of lines in the buffer to standard output. If the buffer is empty, then display the message ‘‘`==>> BUFFER IS EMPTY <<==`’’. The values of *start* and *stop* must be in the range [1..$n$] where $n$ is the number of lines in the buffer. If any of the value of *start* or *stop* are outside the range, print the message ‘‘`==>> RANGE ERROR - start stop MUST BE [1..`$n$`] <<==`’’. Replace $n$ in the error message with the number of lines. Otherwise, the range parameters are good and display buffer lines *start* through *stop* (inclusive) to standard output. The current position in the buffer must not change.

**SEARCH BACKWARDS (Command: `? pattern`)**

This command will search for the first occurrence of the string *pattern* in lines prior to the current line. If the pattern is found, the current position will move to that line. If the pattern is not found, display the message ‘‘`==>> STRING` *pattern* `NOT FOUND <<==`’’, and

the current position does not change. If the buffer is empty, then display the message ‘‘==>> BUFFER IS EMPTY <<==’’. If the current position is at line 1, then display the message ‘‘==>> ALREADY AT TOP OF BUFFER <<==’’.

**SEARCH FORWARDS (Command: / pattern)**

This command will search for the first occurrence of the string *pattern* in the current line and lines below current. If the pattern is found, the current position will move to that line. If the pattern is not found, display the message ‘‘==>> STRING *pattern* NOT FOUND <<==’’, and the current position does not change. If the buffer is empty, then display the message ‘‘==>> BUFFER IS EMPTY <<==’’.

**SUBSTITUTE TEXT (Command s text1 text2)**

This command will search the current line for all occurrences of *text*1 and replace them with *text*2.

**SUBSTITUTE TEXT IN RANGE (Command sr start stop text1 text2)**

This command will search the range of lines [*start..stop*] for all occurrences of *text*1 and replace them with *text*2.

**DELETE CURRENT LINE (Command d)**

This command will delete the current line from the buffer and copy it into the clipboard. If the current buffer is empty, display an error message ‘‘==>> BUFFER IS EMPTY <<==’’ and continue with the next command. The deleted line is copied into an empty clipboard. If the clipboard is not empty, clear it and then paste the line into the clipboard.

**DELETE RANGE (Commnad dr start stop)**

Similar to the delete command (d), but deletes a range of lines from the buffer and pastes them into the clipboard. The start and stop arguments must be valid indices. If not, display an error message ‘‘==>> INDICES OUT OF RANGE <<==’’.

**COPY CURRENT LINE (Command c)**

This command will copy the current line from the buffer and copy it into the clipboard. If the current buffer is empty, display an error message ’’==>> BUFFER IS EMPTY <<==’’ and continue with the next command. The copied line is inserted into an empty clipboard. If the clipboard is not empty, clear it, and then paste the line into the clipboard.

**COPY RANGE (Commnad cr start stop)**

Similar to the copy command (c), but copies a range of lines from the buffer and pastes them into the clipboard. The start and stop arguments must be valid indices. If not, display an error message ‘‘==>> INDICES OUT OF RANGE <<==’’.

**PASTE ABOVE (Command pa)**

This command will paste the contents of the clipboard into the lines above the current line. If the clipboard is empty, then display an error message ‘‘==>> CLIPBOARD EMPTY <<==’’.

**PASTE BELOW (Command pb)**

Similar to paste above, this command will past the lines below the current line.

**INSERT ABOVE CURRENT (Command ia)**

This command will read lines from standard input until a single line containing a ‘‘.’’ is found and insert them into the buffer above the current line.

**INSERT AT CURRENT (Command `ic`)**

`This command will read lines from standard input until a single line containing a ‘‘.’’ is found and insert them into the buffer at the current line (side effect is that current is pushed down the list).`

**INSERT BELOW CURRENT(Command `ib`)**

`This command will read lines from standard input until a single line containing a ‘‘.’’ is found and insert them into the buffer below the current line (the current line remains at the same index).`

## 2 CommandLine Class

Write a Java class called `CommandLine` (stored in the module `CommandLine.java`) that will read a command from the standard input, determine if it is valid command, and parse the input line into command and argument tokens. Commands will have either 0, 1, 2, 3, or 4 arguments. You will need to write a test program to verify this is working correctly.

## 3 Doubly Linked-List Class

Implement a doubly linked-list class that uses Java generics. This implementation of a doubly linked-list class has many methods and data members. You are to implement the primitives (methods) of this ADT and write a test program to verify that each of the methods works correctly.

A sample TestDLList.java test program is also attached. This test program is incomplete, but should be used to test your list primitives.

## 4 Buffer Class

Write a Java class called `Buffer` (stored in the module `Buffer.java`) that will create a buffer. Two buffers will be used in this project: the main buffer and a clipboard. A buffer consists of three things: a String that stores the filespec (file name) of the current buffer, a boolean called `dirty` that is set if the current buffer is changed in any way, and a doubly-linked list of strings. Your main LTE class will create to objects of type Buffer: buffer, and clipboard.

## 5 LTE Class

Write a line text editor class called `LTE` (stored in the module `LTE.java`). This will be the class file where the `main()` method will reside for this project. The main program will first check if there is a single (extra) command line argument. If there is, LTE will assume it is a file name and try to open the file and read the lines of text into the buffer. If there is nothing passed on the command line, then LTE will start with an empty buffer. The `process_command` method will print a prompt, get a command string from standard input, parse the command string int a command and arguments using your `CommandLine` class, and execute the command (suggestion is to use a switch). Each LTE command should have its own “driver method” that separately performs that operation on the the buffer. The user input prompt for LTE should display the filespec, command number, and line number: example `LTE:foo.java:57:22>_` File name is foo.java, command number 57, line 22 in the buffer.

## 6 Hints, Error Checking, and Other Comments

- Help should display a nicely formatted help message of all commands for this project. The help message should be limited to 23 lines and 80 columns to the help text does not scroll off the screen.
- Most text editors keep track of any changes that have been made to the file during editing. Many times the text editor will prompt the user to save changes before exiting, etc. Thus, any modification to the buffer should set the "dirty bit" to true.
- Reading a file will overwrite the contents of the current buffer. If the dirty bit is set, prompt the user to save the contents of the current buffer before reading the file.
- Write will always write the buffer to the disk using the current file name only if the dirty bit is set. All write operations will reset the dirty bit to false.
- Quitting the program will exit back to the shell. If the dirty bit is set, prompt the user to save the buffer to disk before exiting. The ``q!'' command exits the program regardless of the status of the dirty bit.
- Any command that requires a number as an argument must be a valid number (i.e. digits) and the argument must be in range of the number of lines in the current buffer. Any number out of range or invalid is an error and the command is aborted. Prompt for the next command to not ``exit()''.
- All ranges of line numbers (i.e. start stop) are inclusive.
- Display an appropriate error message when traversing the buffer using - and + when the current line is either the top or bottom line in the buffer.
- Searching a the buffer for a pattern will be a continuous search operation stopping and displaying the current line number and column number where the pattern was found. The user will have the option of continuing the search. If the search is continued, the search resumes on the current line after the preceding pattern was found.
- The delete and copy commands will fill the clipboard with line(s) of text. The clipboard is always flushed before these operations take place.
- The paste commands will copy the current contents of the clipboard into the line before or after current. If the clipboard is empty, display an appropriate error message. The contents of the clipboard remains unchanged if the buffer is changed to a different file. This will allow copying contents of one file into another.
- Starting the application with the name of a single file will automatically load that file into the buffer. More than one file name on the command line is an error and the program should display a "usage" message and exit gracefully. Otherwise the program will start with an empty buffer.

## 7 Assessment, Submission, and Grading

- This is an individual project.

- Your program must be written in Java.
- Your program will be tested using your computer's terminal screen.
  - `linux$ javac CommandLine.java`
  - `linux$ javac DLList.java`
  - `linux$ javac Buffer.java`
  - `linux$ javac LTE.java`
  - `linux$ java LTE test1.txt`
- Your program must read from standard input (keyboard) and write to standard output (terminal screen).
- The output must formatted exactly as as specified and shown in this document.
- You must use good software design. In this project, the design is fixed.
- Comment and document all code submitted and follow the documentation guidelines (forthcoming) as specified by your instructor.
- Follow the submission guidelines and procedures (GitHub Classroom).
- Commit and push your changes regularly using `git`.
- This project is worth 100 points.