Editor
CMPUT 175 Assignment 3

Due Date: December 3

"**ed**" is a line-oriented editor for text. It was developed in the late 1960s as part of the creation of the UNIX computer operating system. (Linux is the popular open-source descendent of UNIX.) Your assignment is to build an **ed**-like editor. The commands are based on those of **ed** but have been simplified both in terms of the number of commands supported and the complexity of the commands. Once this assignment is complete, you will have a functional text editor.

Note that because of the pared down functionality, some editing operations are awkward. Of course, after submitting the assignment, you are free to enhance its capabilities!

Example

The following is sample output from running your editor. Text beginning with a # are comments to help explain what is happening. Commands are followed by 0, 1 or 2 arguments, each separated by spaces.

Note in the following that the line last accessed in the file is considered the "current" line. Any commands that talk about a relative position in the file do so relative to the current line.

```
Welcome to ed379
>l A3-sample.txt                          # Read in a file
A3-sample.txt (10)                        # file name (#lines read)
>1                                        # Go to line 1
>p 100                                    # Print line 1 and the next
  1: It was the best of times,            # 100 lines (it does not
  2: it was the worst of times,           # matter if the number is
  3: it was the age of wisdom,            # bigger than the number of
  4: it was the age of foolishness,       # lines in the file
  5: it was the epoch of belief,
  6: it was the epoch of incredulity,
  7: it was the season of light,
  8: it was the season of darkness,
  9: it was the spring of hope,
 10: it was the winter of despair.
>5                                        # Go to line 5
>                                         # Null command. Print the
  6: it was the epoch of incredulity,     # next line
>                                         # Null command
  7: it was the season of light,
>                                         # Null command
  8: it was the season of darkness,
>p -4                                     # Print the 4 previous lines
  4: it was the age of foolishness,       # to the current line and
  5: it was the epoch of belief,          # then the current line
  6: it was the epoch of incredulity,
  7: it was the season of light,
  8: it was the season of darkness,
>a                                        # Append text after current
```

```
[In Edmonton, the season is six months      # line (8)
long, but in Hawaii it is much less!]
                                            # An empty line ends input
>p -5                                       # Print the current line and
  5: it was the epoch of belief,            # the 5 previous ones
  6: it was the epoch of incredulity,
  7: it was the season of light,
  8: it was the season of darkness,
  9: [In Edmonton, the season is six months
 10: long, but in Hawaii it is much less!]
>8                                          # Go to line 8
>i                                          # Insert text before
[In Edmonton, May, June, July, and August   # current line
are our season of light. Sadly, it is not
long enough!]
                                            # An empty line ends input
>1                                          # Go to line 1
>p 100                                      # Print
  1: It was the best of times,
  2: it was the worst of times,
  3: it was the age of wisdom,
  4: it was the age of foolishness,
  5: it was the epoch of belief,
  6: it was the epoch of incredulity,
  7: it was the season of light,
  8: [In Edmonton, May, June, July, and August
  9: are our season of light. Sadly, it is not
 10: long enough!]
 11: it was the season of darkness,
 12: [In Edmonton, the season is six months
 13: long, but in Hawaii it is much less!]
 14: it was the spring of hope,
 15: it was the winter of despair.
>8                                          # Go to line 8
>d 2                                        # Delete current line + 2
>p                                          # Print current line
  8: it was the season of darkness,
>                                           # Null command
  9: [In Edmonton, the season is six months
>d 1                                        # Delete current line + 1
>p                                          # Print current line
  9: it was the spring of hope,
>1                                          # Go to line 1
>p 100                                      # Print
  1: It was the best of times,
  2: it was the worst of times,
  3: it was the age of wisdom,
  4: it was the age of foolishness,
  5: it was the epoch of belief,
  6: it was the epoch of incredulity,
  7: it was the season of light,
  8: it was the season of darkness,
  9: it was the spring of hope,
 10: it was the winter of despair.
>? belief                                   # Search backward for belief
```

```
  5: it was the epoch of belief,            # only find 1st occurrence
>? "best of times"                          # Search backwards
  1: It was the best of times,
>? worst                                    # Search backwards, wrapping
  2: it was the worst of times,             # around
>/ hope                                     # Search forward
  9: it was the spring of hope,             # only find 1st occurrence
>/ was                                      # Search forward
 10: it was the winter of despair.
>/ was                                      # Search forward, wrapping
  1: It was the best of times,              # around
>r best BEST                                # Replace “best” with
  1: It was the BEST of times,              # “BEST”
>r "of times" "OF EVERYTHING!"              # Replace
  1: It was the BEST OF EVERYTHING!,
>i                                          # Insert at current line (1)
With apologies to Charles Dickens...

>p 100                                      # Print
  1: With apologies to Charles Dickens...
  2: It was the BEST OF EVERYTHING!,
  3: it was the worst of times,
  4: it was the age of wisdom,
  5: it was the age of foolishness,
  6: it was the epoch of belief,
  7: it was the epoch of incredulity,
  8: it was the season of light,
  9: it was the season of darkness,
 10: it was the spring of hope,
 11: it was the winter of despair.
>s                                          # Sort (ascending)
>p -100                                     # Print
  1: It was the BEST OF EVERYTHING!,
  2: With apologies to Charles Dickens...
  3: it was the age of foolishness,
  4: it was the age of wisdom,
  5: it was the epoch of belief,
  6: it was the epoch of incredulity,
  7: it was the season of darkness,
  8: it was the season of light,
  9: it was the spring of hope,
 10: it was the winter of despair.
 11: it was the worst of times,
>w A3-newfile.txt                           # Write out file
A3-newfile.txt (11)                         # file name (#lines written)
>q                                          # Quit

Good-bye
```

Commands

This section provides a list of the commands that your program will support. Commands can take the following forms, with each part separated by one or more spaces. A command can be preceded by white space.

- <> An empty line; no non-blank characters.
- <n> Line number (n), a positive integer

- <c> Command (c), one of “a”, “d”, “i”, “”l”, p”, “q”, “r”, ”s”, “w”, “/”, “?”
- <c> <i> Command (c) and an integer (i) offset (can be negative)
- <c> <text1> Command (c) and a text parameter (text1)
- <c> <text1> <text2> Command and two text parameters (text1, text2)

Below is a table summarizing the commands. The last line accessed by a command is called the “current line.” Some commands result in moving around in the file, causing the current line to change. How a command changes the current line is documented in the table.

c	Action	Usage	Semantics	Current Line
a	Add text	a	Add lines of text after current line. End input with an empty line (no characters on it). Can add lines to an empty file.	Last line added
d	Delete	d	Delete current line. An error if the file is empty.	Next line; last line if at file end
		d n	Delete current line and next n. An error if the file is empty. There is no error message if n goes beyond file end.	Next line; last line if at file end
		d -n	Delete previous n lines and current line. An error if the file is empty. There is no error message if n goes before file start.	Next line; first line if at file start
i	Insert	i	Insert lines of text before current line. End input with empty line (no characters on it). Can insert lines to an empty file.	Last line inserted
l	Load	l fname	Read file named “fname”. An error if the open fails. No check if the user failed to write out previous file.	Last line of file
p	Print	p	Print current line. No error if file is empty.	No change
		p n	Print current line and n more. There is no error message if n goes beyond file end. No error if file is empty.	Last line printed
		p -n	Print n previous lines and current line. There is no error message if n goes before file start. No error if file is empty.	No change
q	Quit	q	Exit the program. No check if the user failed to write out previous file.	
r	Replace	r text1 text2	If “text1” is present in current line, replace it with “text2” (only once). No error if file is empty or if replace fails.	No change
		r text1	If “text1” is present in current line, replace it with “” (empty). No error if file is empty or if replace fails.	No change
s	Sort	s	Sort the lines in the file into ascending order (don’t add descending order). No error if file is empty.	Last line of file
w	Write	w fname	Write to file “fname”. An error if the open fails.	Last line of file
		w	Write to file (use name from load command). An error if the open fails.	Last line of file
/	Search forward	/ text1	Search from next line for “text1”, wrapping around to file head if needed. Only find first occurrence of “text1”. When cycle	If match, line of the match; If not, no change

			around to current line, stop without matching. No error if search fails.	
?	Search backward	? text1	Search from previous line for “text1”, wrapping around to file tail if needed. Only find first occurrence of “text1”. When cycle around to current line, stop without matching. No error if search fails.	If match, line of the match; If not, no change
<n>	Line number	<n>	Go to line <n>. It is an error to go beyond the end of file or before the first line in the file.	Line <n>; No change if error
<>	Print next	<>	Print next line. No printing if next line is end of file.	Line printed else no change

A few semantic notes:

- The program does not need to read in a file as its first command. The program should start up with empty text, which the user could start adding to.
- When reading in a file, the previous file is “thrown away” (get it garbage collected). Your program does not have to check to see if the previous file was changed. It is the user’s responsibility to write the file out before reading in a new one.
- The above comment also applies for the quit command. It is the user’s responsibility to write the file out before quitting the program.
- The original **ed** program was sparse on error reporting (as is this program). Give appropriate messages for all command formatting errors and I/O errors. Also report all meaningful semantic errors – errors that if executed would cause a problem (e.g., asking to go to a line number beyond the end of file). Some commands can have minor errors – errors that have no impact (e.g., asking to print 100 lines of a file, but there are only 10 lines in the file; a search does not find a matching line; a replace does not do a text replacement). These messages can be ignored.

Implementation

You will implement a `TextFile` using a doubly linked list (the code is provided to you). It is essentially the same as the course notes and as seen in Lab 7 (with some getter/setter methods added). Each linked list entry will contain one line of text from the file that is being edited.

The linked list code provided is not complete; you need to implement the `insert` method. Note that it differs from the course version in that this method takes an extra parameter:

```
insert(self, current, item, where)
```

The new node is added relative to `current`, with `where` indicting whether it is inserted before or after.

You should add the doubly linked list code to your program. Do not import it since the file given to you does not have a complete `insert` method.

You are to create a new class called `TextFile`. The `TextFile` class contains the following public methods.

- `__init__(self,fname)` – create a `TextFile`, recording the name of the file read.
- `load(self,name)` – read in text.
- `write(self,name)` – write out file’s text.
- `print(self,offset)` – print line(s), with offset indicating the number of lines before or after the current line to be printed.
- `linenum(self,lineno)` – set the current line to be the one at line #.
- `add(self,where)` – `where` is “insert” (before) or “add” (after) the current line.
- `delete(self,offset)` – delete line(s), with offset indicating the number of lines

before or after the current line to be deleted.

- `search(self,text,where)` – where is to look “before” or “after” the current line.
- `replace(self,text1,text2)` – in the current line, replace “text1” with “text2”, if possible.
- `sort(self)` – sort the entire file.

- `getName(self,fname)` – get the name of the file.
- `setName(self,fname)` – set the name of the file.
- `getCurr(self)` – get the current line.
- `setCurr(self,current)` – set the current line.
- `getLine(self)` – get the line # of the current line.
- `setLine(self,line)` – set the line # of the current line.

You are allowed to have internal state in your `TextFile`, appropriate getter and setter methods, and supporting methods.

Your program needs to keep track of the current line in a file. You also need to keep track of the line # of the current line. Some commands change the current line, and you will have to update the line number. Two things you must not do:

- Do not associate a line # with each line in the file. That is expensive to maintain. For example, every time you add or delete a line in the file, you may have to traverse the file to update the line #s.
- Do not find a line # by always searching from the head (or tail) of the file. Again, this is an expensive operation.

The solution is to maintain this information incrementally. For example, if your current line is line # 5 and the user executes the command “p 4”, you will print out the current line and the next four. This results in the current line moving to the line last printed, which is line # 9 (=5+4).

For the sorting command, use selection sort. Your implementation will only support putting the data into ascending order.

The substitute command is more powerful if it can replace phrases, not just single words. `Split()` is useful to break text into words, but it does not understand quotes. The `shlex` routine `split()` understands quotes. The following code illustrates the difference

```
import shlex
line = input()
print( "LINE", line )
l = line.split()
print( "SPLIT", l )
l = shlex.split(line)
print( "SHLEX", l )
```

For the input:

```
one "two three" four
```

The output is:

```
LINE one "two three" four
SPLIT ['one', '"two', 'three"', 'four']
SHLEX ['one', 'two three', 'four']
```

You are allowed to import `shlex` into your code, but no other imports are allowed.

Testing

You will be provided with the following files:

- `A3-DLinkedList.txt` – a file containing the code for doubly linked lists.
- `A3-input.txt` – a series of editing commands for your program to execute (the set of commands shown in the example above).
- `A3-newfile2.txt` – the input file above produces a file called `A3-newfile.txt`. `A3-newfile2.txt` is a copy of this file for you to compare to see if your answer is correct.
- `A3-sample.txt` – a text file that will be read into your editor.
- `A3-output.txt` – the results of running your editor with the input from `A3-input.txt`. The output should be identical to that given in this file. As well the file written (`A3-newfile.txt`) should be identical to `A3-newfile2.txt`.

It is your responsibility to thoroughly test your program, including potential error scenarios.

Rubric

- Code quality and adherence to the specifications: 10%
- `DLinkedList` class (including insert): 10%
- `TextFile` I/O (load and write): 5%
- `TextFile` searching (forward and backward): 10%
- `TextFile` adding text (add and insert): 10%
- `TextFile` delete text command: 5%
- `TextFile` replace command: 5%
- `TextFile` other commands: 5%
- `TextFile` printing commands: 5%
- `TextFile` sort command: 10%
- Proper maintenance of the current line: 10%
- Use of asserts/exceptions: 5%
- Main program (e.g., command correctness): 10%
- Bonus extension (see below): 10%

Hints for Getting Started

- Create a `TextFile` class.
- Implement the `load` method. Implement a simplified version of the `print` method to print out the entire file (to verify that you have read it in correctly).
- Implement the `write` method. It should produce a file whose contents are the same as the file you read in.
- Implement the `line number` command and verify that you are setting the current line correctly.
- Enhance the `print` method to support positive and negative offsets.
- Now try the search forward (`/`) and search backward (`?`) commands. Make sure the current line is being set correctly.
- The `add` and `insert` commands will require that you implement the `DLinkedList insert` method.
- Finish the rest of the commands.

Program Extensions

There are obvious extensions to this assignment that you can consider implementing:

1. For bonus marks... have the program prompt the user if they are about to discard a file that has changed but not been written out. They should have the chance to save their text. This can occur if the user reads in a file or quits the program before doing a write.

2. Have commands work for ranges of numbers. For example, “10,20 p” would print lines 10 to 20 (inclusive).
3. Have commands work for relative ranges of numbers. For example, “-5,+5 p” would print 11 lines (5 before the current line, the current line, and 5 after the current line).
4. Enhance the search and replace commands so that they can apply to the entire file. For example, “r text1 text2 g” would be a replace “global” resulting in all occurrences of text1 being replaced by text2. “s text1 g” would find all occurrences of text1 in the file.

Submission Instructions

Please follow these instructions to correctly submit your solution:

- All your code should be contained in a single Python file: **editor.py**.
- Make sure that you include your name (as author) in a header comment at the top of **editor.py**, along with an acknowledgement of any collaborators/references.
- Please submit your **editor.py** file via eClass before the due date/time.
- Do not include any other files in your submission.
- **Late submissions will not be accepted**. You can make as many submissions as you like before the deadline – only your last submission will be marked. Submit early; submit often.

REMINDER: Plagiarism will be checked for

Just a reminder that, as with all submitted assessments in this course, we use automated tools to search for plagiarism. In case there is any doubt, you **CANNOT** post this assignment (in whole or in part) on a website like Chegg, Coursehero, StackOverflow or something similar and ask for someone else to solve this problem (in whole or in part) for you. Similarly, you cannot search for and copy answers that you find already posted on the Internet. You cannot copy someone else’s solution, regardless of whether you found that solution online, or if it was provided to you by a person you know. **YOU MUST SUBMIT YOUR OWN WORK**.