COSC2320: Data Structures
Recursion with Linked Lists: a Simple Editor

Instructor: Carlos Ordonez

1 Introduction

You will create a C++ program that can edit a text file with line and word insertions and deletions. The file consists of a sequence of lines, where each line has words or numbers separated by spaces. Notice these commands are sufficient to produce any edited file, mimicking how the user manipulates the keyboard.

The goal is to manipulate the file with linked lists. Specifically, the file will be a list of lines and each line will be a list of words or numbers, separated by spaces (one space).

Editing the file will be done with a sequence of line and word insertions and deletions.

2 Program input and output specification

The main program should be called "editor". The output should be sent to the standard output (so that it can be visualized or redirected).

The input file is a regular text file, where each line is terminated with an end-of-line. There is a second file containing editing commands.

Call syntax at the OS prompt:

```
editor "input=<filename>;commands=<filename>"
```

Technical details about program:

- The input and command files are plain text files, where each line ends with end of line.
- Separators: Spaces (one space between two words/numbers). Separators do not count as words and thus they are ignored to determine positions.
- Commands:

```
insertword=linenumber:searchword/insertword
deleteword=linenumber:searchword
insertline=linenumber
deleteline=linenumber
```

insert/delete one word (both require searching for a word). Insertion is done after the "search" word. Deletion deletes the 1st word occurrence. If there are repeated words process the insert/delete command for the 1st word occurrence.

Insert/delete one line (no word specified). The line number must be $1 \ldots N$, where N is the total number of lines.

- There may be multiple occurrence of insert/delete commands, in any order (you cannot assume commands are sorted by line number).
- The edited file is printed to the console (stdout). Remember the user may redirect the output to a file.

3 Example

Example of program call:

```
editor "input=xyz.txt;commands=mycommands.txt"
```

Input file example:

```
This is is an an icorrect
XSSD
A 2nd phrase
```

Example of command file (notice commands are not sorted by line number):

```
deleteline=2
insertline=3
insertword=3:End
deleteword=1:icorrect
insertword=1:an/incorrect
insertword=1:This/sentence
deleteword=1:is
deleteword=1:an
deleteword=1:an
```

Output example

```
This sentence is incorrect
A 2nd phrase
End
```

4 Requirements

- Correctness is the most important requirement: TEST your program with many input files. Your program should not crash or produce exceptions. You should track the recursive calls eaither with the debugger or printing periodically the call stack.
- All functions to search, compute list length and display the list must be recursive. Insertion and deletion of one node are not recursive, but should call the recursive search function. Therefore, using while/for loops to manipulate the list is not allowed. We emphasize that recursive functions are a requirement only for the linked lists, not for the other parts of the program (like reading the file, parameters, etc).
- Your program should perform a sequence on insertions and deletions following the command file. Do not display line numbers when producing the output edited file (this will hinder atuomated testing).

- You should verify the correctness of your output with any editor (vi in Unix, notepad in Windows). Double check the line number. Notice the line number is *not* shown in the output file (the line number is displayed by the editor).

  To show line numbers in Unix use "cat -n <outputfile>".

- Editing commands: you can assume all commands will be given in lowercase. You should do a basic validation to reject invalid commands (mispelled, undefined, bad arguments).

  line insertion: a blank line, with no word of separator.

  word insertion: separated by one space.

  You cannot assume commands come "sorted" by line number. That is, you can expect commands to refer to any valid number of line within the file, but do a basic check of valid line number.

- Linked lists and recursive functions are a major requirement. The input sentence (possibly a paragraph) is maintained as a doubly linked list, where each node will store a word. Notice you can store the separator as a second string in the node together with the word or in a separate node or not store it at all (make a comment about your decision). The file (sequence of lines) can be manipulated as a singly or doubly linked list. For each line doubly linked lists are required: A program using arrays will get 50% max.

- You do not have to use recursive calls to read the input text file, but your "insert" function cannot have a loop. You do not have to store the input parameters in a linked list; in fact, it is encouraged to reuse your previous C++ code.

- The updated edited document should be printed only at the end, *after* processing the last editing command. In order to debug your program you can write the contents of the linked lists to a text file of your own (e.g. "debuglist.txt"). You should not display the output file to the console after executing every command since this will hinder proper testing. In short, avoid mixing the output file with debugging information.

- Memory allocation and deallocation.

  It is unacceptable to use static memory allocation (e.g. arrays) or to "forget" to clean memory. Your program must use "new" to allocated each node and must free memory (using the "delete" operator) before main() ends (e.g. in the last object destructor). A program with memory management bugs will likely crash at the end.

- Limits: You can assume input text lines can have up to 256 characters, but that should not be a limit in the list. The minimum number of lines in an input file is zero (empty). You cannot assume a maximum number of lines in the input file.