COSC2430 HW2: Linked Lists
Reversing words in a text file

Instructor: Carlos Ordonez

1 Introduction

You will write a C++ program to reverse a test file containing diverse strings, including words. You must program doubly linked lists in order to process the input file. It is preferred, but not necessary, that your algorithm is as efficient as possible, both in time to complete as well as memory management.

The purpose of this homeworks is unsderstanding how linked lists help managing and processing data with dynamic size. Also, there will be specific programming requirements to manipulate the linked list. Therefore, **you are not allowed to use the linked list template in the STL Library.**

The goal is to print the words in the file in reverse order. In Phase 1 you will print every string and in Phase 2 you will "clean" the list removing numbers and then printing only words.

With this homework you will gain understanding on how an editor or word processor manipulates text files or documents. Notice a file is truly a very dynamic object of any size with diverse content.

2 Input

The input is one text file, with strings, where a string is a sequence of letters (simple words) or a sequence of digits (i.e. simple integers). Strings are separated by spaces, commas (repeated spaces should be considered as one space). Each line has a line feed (\n) in Unix format (be careful if you develop your program in Windows and you port it later to Unix). You can assume each string will be at most 30 characters long and it is acceptable to use any data type or class for such string (i.e. char[] array, string class, char *, etc).

Notice that words can be followed by punctuation signs and that some of them can be capitalized (upper case). This should not interfere with your results: a capitalized word is the same as a non-capitalized word for the purposes of this homework. Uppercase letters can be anywhere in the word, for ex.: "Hungry", "hungry" and "hUngry" should be counted as the same word. Punctuation and separators should be ignored.

3 Program input and output specification

The main program should be called "reverse". Therefore, your .cpp file should be called reverse.cpp.

The input is a plain text file with a sequence of words, where each word is a string of one or more letters. You can create text file with any editor like nano or vi.

Call syntax from the Unix command line (default for clean parameter is clean=n, if not specified):

```
reverse inputfile=<input.txt>;clean=<y/n>
```

Notice that the file name will not necessarily be the same every time. Therefore, your program will have to take that into account. You can use the Command Line Parser that is provided in the TAs' homepage.

The processed output should be sent to a fixed file "output.txt" with only strings separated by one space, but maintaining the same lines as the original file.

Important clarifications. Notice that the output word should always be in lowercase, regardless of how the words appear in the input file (i.e. convert words to lower case). Notice the file will have a mix of words, numbers, separators and perhaps other symbols like mathematical operators, brakets, parentheses, etc (simply ignore them by negation). Notice words and numbers may not have a separator, but will need to be separated in the output file.

3.1 Example

Example of input file (t.txt):

```
xyz abc
fun sad book 123,678+1000
a AA aaa123, $%
[X]
```

Example of result file output.txt (fixed file name in this homework) with all words and numbers, removing all other symbols (clean=n or not specified):

```
x
123 aaa aa a
1000 678 123 book sad fun
abc xyz
```

Example of result file output.txt (fixed file name in this homework) with only words and program call "reverse inputfile=t.txt;clean=y":

```
x
aaa aa a
book sad fun
abc xyz
```

4 Requirements

- General requirements

  Your program must compile and run with GNU C++. It is discouraged to use any other compilers because of pointer and file manipulation: a program working in one operating system may not work on a different operating system with unexpected errors at run time. Keep in mind C++ is not as portable as Java.

  Your source code must be individual. Copying or modifying source code from another person is forbidden. Any significant similarity between your source code and source code from another student (including past editions of the course) will be treated as cheating and reported to the Undergrad Director. Any program downloaded or copied from the Internet should be disclosed to avoid false alarms (keep in mind another studenrt had the same idea).

  Follow the call syntax exactly as specified (name of program, input/output parameters). Avoid using other names for the main program or the program parameters since this will make your program fail automated testing. Always test your program ahead of the due date with the sample scripts and files provided by the TAs. Ask any questions or point any inconsistencies in this specification well before the deadline (pointing out confusing aspects on the last day is useless).

You must include a README.TXT file explaining the main classes of our program and specifically where you programmed your required data structure or algorithm. This requirement will remain for every homework.

Future homeworks will use similar logic for input/output and testing. Try to write your C++ classes and functions in a way that can be used and maintained in the future.

Don't forget to comment your code, especially with a short explanation at the top of each source code file or complicated logic (e.g. pointers, nested loops).

Leave the source code in your home folder. Avoid leaving your last version in subfolders. Follow TAs' instructions. Always keep in mind your programs will be automatically tested and your source code will be later manually checked to make sure you implemented the requirements.

Test cases. Your program will be tested with 10 test cases, going from easy to difficult, where each test case is 10 points. You can assume 80% of test cases will be clean, valid input files. You are not expected to handle files with many errors or input not complying with this specification.

Test your program well with many files, especially corner cases like empty files, empty lines, many words per line, numbers, strange separators. Your program is not expected to handle every potential case in the input file, but should not fail if there are minor issues in the input file (e.g. the TAs made a minor mistake).

Your program should write error messages to the screen, not to the output file. Keep in mind that not following the output file format will make your program fail test cases.

Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.

Resubmission or fixing code is not allowed after the deadline. Therefore, ask before and test well your program.

Grade (out of 100 points): A program not finished by the deadline is zero points. A non-compiling program or a program that does not produce any meaningful output is worth 10 points. A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 60 or higher. Correctness is more important than speed, but speed will gain importance as you move forward in the course.

- Specific requirements

  Phase 1: each line of input file stored as doubly linked list. Functions: insertion, traversal, print. It is acceptable to have an array of linked lists, assuming the file has up to 1000 lines, but keep in mind Phase 2 where the input file may be much larger. Your program must display on the screen the number of lines in the file and the number of bytes used by the program (it is expected each student will have slight differences). It is expected different files will use different memory size. The input file may have up to 1000 lines. It is important NOT to include any messages or in the output file. Your README file must explain in which file and class you programmed the doubly linked lists (preferably one file). The TAs will specificaly review your source code to make sure you do not use arrays to store each line. Any program using arrays or the STL list classes to store each line will receive a 20/100 as maximum grade. Phase 1 cannot be resubmitted after grades are released. However, Phase 2 will give you an opportunity to fix bugs or issues from Phase 1, but test cases will be different.

  Phase 2: the entire file is now stored as a linked list of linked lists. That is, each line is a linked list and has pointers to the previous and next lines. Functions: deletion of individual line list, deletion from global list not necessary. The entire file is processed traversing the "global" linked list of lines

and ”individual” linked list for each line. The user has the option to remove numbers (clean=y): in this case the program must traverse the list and delete nodes with numbers. Input files may be much larger, up to 100k lines. Phase 2 will be used to fix any bugs or omissions from Phase 1.