CS 564, Spring 2022: Assignment 1 - Word Locator in C++
Due: Feb 4, 2022, at 11:59 pm

Problem Description

The goal of this assignment is help you brush up your C++ programming skills, and exercise your skills in data structure and algorithm design (the stuff that you know from CS 367). In this assignment, you are to develop a "word locator" program written in C++, which will allow a user to check if a specified (re)occurrence of a specified query word appears in the input text file.

Program Specification

Your program *must* be named "wl". When started, your program will display a prompt ">" (printed on *stdout/cout*) and will then be ready to accept one of the following commands:

1. "**load <*filename*>**": This command loads the specified file. The file may be specified by either an absolute or a relative pathname. Running this command should result in your program parsing and storing the words in this file in a data structure that can be queried using the locate command (described below). A word is defined as a sequence of upper and lower case letters in the English alphabet (i.e. characters 'a' to 'z', and 'A' to 'Z'), numbers, and the apostrophe. All other characters are considered as white space and will therefore be treated as terminating a word. Two successive load commands should be treated as if there is an intermediate "new" command (see below) in between the two commands.

2. "**locate <*word*> <*n*>**": This command outputs the number of the word, counting from the beginning of the file, of the *n*^{*th*} occurrence of the word. Word numbering starts from 1, so the first word in the load file has a word number of 1. *The locate command is case insensitive*, i.e. to match the word in the locate command with a word in the load file you should use a case-insensitive string comparison method. If there are no matches for the locate command, print "`No matching entry".`

   The syntax of the locate command is "locate <word> <n>". The "<word>" parameter will have a whitespace before and after it, and "<n>" should be an integer greater than 0.

   As an example, the following are legal commands: "`locate sing 3`**"** and "`locate  sing  3`**"** Both locate the 3rd occurrence of "sing", but the second command has a few additional blank spaces around the parameter "sing".

   The following commands are not legal: "`locate sing3`**"**, "`locate sing 3q`**"**. The first command does not specify a parameter <n>, and in the second command the parameter <n> is not an integer.

   Please note that the command "`locate sing23 4`**"** is a legal command for locating the fourth occurrence of the word "sing23".

3. "**new**": This command resets the word list to the original (empty) state.

4. "**end**": This command terminates the program.

Your program should respond to incorrect commands in the following ways:

- If a bad command is entered, print the precise string "`ERROR: Invalid command`", and go to the next prompt. Examples of bad commands are: `"find word 7"` and `"locate song".` Other examples of bad command include the locate command having a word that is not legal as per the definition above. For example "ra#s" and "rats!" are invalid word parameters.
- Note that if an incorrect load command is entered, such as `"load"` (no filename) then your data structure should **not** be reset. In other words, if you have a previously loaded file, subsequent locate commands should still query that previously loaded file. Similarly if the load command specifies an

invalid file name, then you should not reset the data structure. In both cases of the invalid load command outlined above, please print the standard error message “`ERROR: Invalid command`”.[1]

- If there is extraneous content in the command, such as “`locate word 5 17`” or “`new 12`”, print out the standard error message: “`ERROR: Invalid command`”
- All the command keywords are case insensitive, so “LoCATe sing 2” is a valid command, and should be treated as “locate sing 2”.

Example

Given the following sample text file, sixpence.txt:

```
Sing a song of sixpence,
A pocket full of rye;
Four and twenty blackbirds
Baked in a pie.
When the pie was opened,
They all began to sing.
Now, wasn’t that a dainty dish to set before the King?

The King was in the countinghouse,
Counting out his money;
The Queen was in the parlor
Eating bread and honey.
The maid was in the garden,
Hanging out the clothes.
Along there came a big black bird
And snipped off her nose!
```

The following is a sample run:

```
>load sixpence.txt
>locate song 1
3
>locate Song 1
3
>locate SoNg 1
3
>locate pie 1
18
>locate pie 2
21
>locate pie 3
No matching entry
>locate prince
ERROR: Invalid command
>locate prince 1
No matching entry
>new
>locate song 1
No matching entry
>end
```

Format your programs responses exactly as in the above example. The programs will be graded by an automaton, and to assure proper credit, your program must respond exactly as above. It is essential that nothing else be output to *stdout/cout*. Any extraneous messages should be removed before turning in the program. It is also expected that your program will not crash, no matter what the input is, either in the loaded text file, or in the sequence of commands given to the program.

[1] The error handling component of this assignment has *deliberately* been simplified to ease the programming load. In a real application, one should print error messages that identify the specific error and help the user in correcting their input. You will get sufficient practice in such error handling methods when you work on the BadgerDB assignments.

Design Task

Your main design task is to pick a **tree-based** data structure (so do not use a hash-based index structure!) that allows *efficient* execution of the locate command. You may have several design choices, and I want you to pick the most efficient data structure that you can think of. You can't use in-built C++ STL data structures for this key data structure.

Hint: One possible solution is to create a Binary Search Tree (BST) for each input. What should you store at each node to quickly locate the number of the word for the requested occurrence? If you choose to implement a BST make sure to compare to other possible tree types in your design report.

For this assignment, we are not concerned with the efficiency of the load command. However, you do have a restriction on the amount of space that you can use for running your program. The memory footprint of your program, which includes the memory used by your code and the data structure that you build, should not exceed four times the size of the input load file, when measured in bytes. Don't worry about exceeding this limit on very small files. For example, it is okay if your program exceeds this limit when loading the small sample load file sixpence.txt, but on the large file "wrnpc.txt" it should meet this requirement. You can use the command "ps –l" to check the program size. If you are not familiar with "ps", please read the man page.

Getting Started

The necessary files for this assignment (can be downloaded from Canvas):

1. wl.h and wl.cpp: Your solution code has to added here. You will create these files.
2. Makefile: A sample makefile.
3. sixpence.txt: A sample text file.
4. sixpence.cmd: A sample command file.
5. sixpence.out: Sample output when the command "wl < sixpence.cmd" is run.
6. wrnpc.txt: Another sample text file (sample command and outputs are not provided for this file).

Development Platform

- **Platform:** The stages will be compiled and tested on the CS department machines running Ubuntu 20.04 LTS Linux. See https://csl.cs.wisc.edu/docs/csl/2012-08-16-instructional-facilities/ for the list of machines. We plan to use the snares-XX.cs.wisc.edu or equivalent machines. You are free to do some development using other platforms (or even set up a Docker on your laptop) but you must make sure that your project works with the official configuration. If you are new to the CS environment, you will need a CS account https://csl.cs.wisc.edu/docs/csl/2012-09-04-about-your-instructional-account/ for details.
- **Warnings:** One of the strengths of C++ is that it does a lot of compile time checking of the code (consequently reducing run-time errors). Try to take advantage of this by turning *on* as many compiler warnings as possible.

Your Assignment

Start by copying the files above in to your workspace. In this directory you will find all the files mentioned above. To complete your assignment, follow the directions in the section above. You should end up with the solution code in the files wl.h and wl.cpp *(please do not create any other files)*, and a report explaining the design choices of the data structure that you use. Your code should be fully commented following the specs for Doxygen (www.**doxygen**.org*)*. In other words, you should be able to generate documentation for your code using doxygen.

Coding and Testing

Your program must be written in C++. Your coding style should have well-defined classes and clean interfaces. The code should be well-documented. Each file should start with a header describing the purpose of the file and should also contain **your name, student id, and UW email address**. The email address must be your official UW email (not your CS address).

Testing for correctness involves more than just seeing if a few test cases produce the correct output. There are certain types of errors (memory errors and memory leaks) that usually surface after the system has been running for a longer period of time. You should use *valgrind* to isolate such errors. This can be done by simply adding "valgrind" to the command line, for example "valgrind ./wl < sixpence.cmd" You will get a listing of memory errors in your program. If you have programmed in Java you should keep in mind that C++ does not have automatic garbage collection, so each *new* must ultimately be matched by a corresponding *delete*. Otherwise all the memory in the system might be used up. Valgrind can be used to detect such "memory leaks" as well.

More information about valgrind can be found at: http://www.valgrind.org/docs/manual/index.html

In addition to the test driver that we provide, your assignment will be tested against our (more comprehensive) test driver. You are encouraged to develop additional tests on your own.

Submitting Your Assignment

You will submit this assignment electronically using Canvas. In addition to submitting your code, you will also submit a design report that describes the following design and program criteria:

1. Explain your choice of the data structure that you implemented. Did you consider any other data structures besides the one that you implemented? How did you arrive at your final choice of the data structure?
2. What is the best, average, and worst case complexity of your implementation of the locate command in terms of the number of words in the file that you are querying? *(you need to provide all three - best, average, and worst-case analysis).* For the complexity, we are only interested in the big-O analysis.
3. What is the average case *space* complexity of your data structure in terms of the number of words in the input file? In other words, using the big-Oh notation what is the expected average size of your data structure in terms of the number of words.

You must also turn in a copy of the two files "wl.cpp" and "wl.h".

Please submit the assignment using Canvas using the following instructions:

1. Name the project directory: firstname_lastname_P1 (e.g. elon_musk_p1)
2. Run: **make clean** from inside the directory (to make the submitted file small)
3. Run: **tar -czvf firstname_lastname_P1.tar.gz /path-to-the-project/firstname_lastname_P1**
4. Submit the tar file

As a check, you can uncompress the tar file (using the command: tar - xzvf __P1.tar.gz). You should see the Makefile and all the source code files (e.g. wc.cpp and wc.h) inside the directory.

If you do not adhere to this standard, 5 points will be deducted.

Grading

This assignment is worth 5% of your grade. The maximum score on this assignment is 100. For this assignment, 60% of the grade is for correctness (correct results, no memory leaks, memory consumption within the requested specification), 30% for your design report, 10% for your programming style. We are not going to run speed tests for this assignment, as I don’t want you to get into detailed code optimization methods. Rather the purpose of this assignment is to make sure that you have a good algorithmic and code design in place. You will be penalized if the algorithmic complexity of your solution is poor – for example, if your algorithm takes on average $O(n^2)$ time to execute the locate command.

The programming style points are to make sure that you follow good programming practices. Your software should have a modular design, and should be well commented and structured so that any other programmer can easily understand your code and design.

As for all assignments and projects in this class, there are no late days.