Word Frequencies using a Linked List *(updated)*

50 points - Due Wednesday, September 27, 11:59 PM

Assignment

You should know by now (after watching all the fun in class) how to use structs, pointers, and a dash of logic to create a linked list. If not, there is some code posted on the website (or will be soon!) to create a simple integer linked list. This assignment builds on those skills. You're going to create a simple word frequency counter that accepts word input from the command line, and builds a linked list that stores each word and the number of times it was encountered. So each node of the linked list will have a string (the word), an integer (the count), and of course a pointer to the next node.

So here's the scoop. You will write a C++ program with a main loop that:

1. Reads a single word from the user.

2. Searches for that word in the linked list. If the word is found, its counter is incremented. If the word is not found, then it is inserted **alphabetically** into the linked list, with its counter is set to 1. Then goes back to step 1, unless the end-of-file has been reached.

3. When end-of-file is reached (see below for more information on this), the program should print out each word in the list and the number of times it occurred, in alphabetical order. (You shouldn't need to do any kind of sorting here, since each word will be inserted in the correct order in the first place).

4. Before terminating, the program should clean up the linked list, making sure that everything which was created dynamically (e.g. the linked list nodes) gets properly deleted.

Assignment Notes

- There's code on the website to read in one word at a time until the end-of-file marker is hit (EOF is a special state that means there is no more input). Feel free to use this code, or to write your own. (When you're typing in your own input in Linux, you can give the computer EOF using Ctrl+D on its own line. On Windows, it's Ctrl+Z, followed by the Enter key.)

- There's always the potential for buffer overflow when using `cin >> word;` as the sample code does. You can ignore this possibility for now and use a fixed-sized buffer, both when getting input from the user and in the linked-list nodes, although in both cases the buffer should be large enough to hold a decent-sized word.

- You may be wondering what the point of this "end-of-file" nonsense is. It allows us to use Unix "pipes", which are an easy way to use the output of one program as input to another. The benefit for you is that you don't have to repeatedly type stuff in.

  An example: the unix program `cat` simply prints out the contents of a file. But if we pipe `cat`'s output into our own program, using the vertical bar (|) symbol, like this:

  ```
  cat gettysburg.txt | ./myProgram
  ```

  ... then if your program is working properly, it will automatically do word frequency counts on the contents of that file. You can play with pipes using the posted sample code, if you want. (Note that you still need to use the `./` in front of your program name when doing this.) This works exactly the same way on Windows, but you need to use the "type" command instead of "cat".

- gettysburg.txt is available for download on the class website as test input. It's a copy of the Gettysburg Address, all lower-case and without punctuation (see the Extra Credit section for the reason why). Or, feel free to use whatever input you want.

- This assignment will require loops, strings, dynamic memory allocation, pointers and pointer accesses. C functions that might come in handy: `strcmp`, `strncpy` and `ispunct`. Google these if you're not sure about how to use them.

What to turn in: a directory containing...

- Your code, of course! A single C++ file should be plenty.

- A README file containing any relevant info, such as your name and compilation instructions (include the exact command you used to compile your code). Again, please include in the README the approximate amount of time you spent on this assignment.

Extra Credit

The assignment as specified doesn't deal well with varying case or punctuation. "GOOD" is treated differently than "good", for example, although they are both the same word. Also, "happy." (with a period) is treated as a different word than "happy" (with no period)... and a dash surrounded by spaces would be read in as its own word. None of this is what we want. The extra credit, then, is this: after each word has been read in (but before it is inserted or updated in the list), convert it to lowercase and strip out any punctuation. And if the "word" consists entirely of punctuation, have your program ignore it entirely (i.e. don't insert it into the linked list).

Non-specific Notes

- Start early; ask questions.

- Refer back to the syllabus for hints about writing good C++ programs (or programs in any language): only work on small pieces of code at a time, and test them well before moving on. Compile lots, and fix the warnings and errors. Write lots of comments.

- **Be sure your code compiles on the CS department's Linux machines before turning it in**. Even if you write it elsewhere (on a different OS, a different IDE, etc), give it a quick compile and test on a lab machine before you turn it in. Remember, if your code does not compile, it will get (at most) 50% credit.

- You must submit your code using the department's electronic submit procedure. A link to instructions for doing this is on the class website. Our course number (for the purposes of the submit program) is **c109sb**.

- Just a reminder: the CLAS academic (dis)honesty policies will be enforced. You're allowed to discuss the assignment with others, but not to collaborate or share code. Nor are you allowed to find or use code off the Internet.