CMSC 202 Spring 2020
Project 5 – Word Clouds

Assignment: Project 5 – Word Clouds

Due Date: Thursday, May 7, 2020 at 8:59pm

Value: 80 points

1. Overview

In this project, you will:

- Practice basic C++ syntax including branching structures
- Write classes and instantiate those classes using a constructor
- Create a templated data structure (linked list)
- Implement copy constructor and assignment operators
- Use simple file input
- Use overloaded operators to access templated data structure

2. Background

When you search for something online or when you click on an advertisement in an email you got, that action is tracked. This data is being aggregated and used to target marketing campaigns specifically catering to your history and projected interests. Companies then use data visualizations (like charts, graphs, infographics, and more) in order to communicate important information at a glance. The action of visualizing the data is something that helps us try and make sense of it.

For this project, we are going to generate the data for a simple data visualization. A word cloud is a simple type of data visualization that highlights important textual data points by analyzing the frequency of a word. Given some text, formatted in a specific way, we should be able to generate a list of all words and their frequency in the text. Then, by exporting the list, we can use one of the many online word cloud generators to visualize our data.

In Figure 1 below, a list of words generated from our proj5_test1.txt file has been visualized by removing all common words (we remove them) and all words that have frequency less than 2 (the word cloud tool removes them).

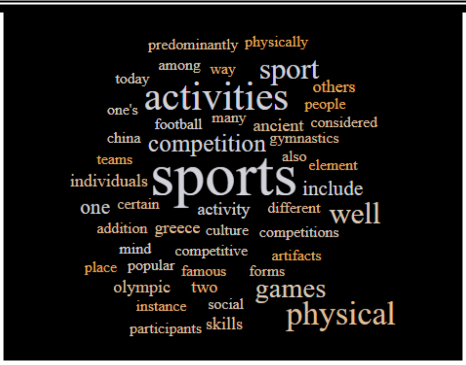
predominantly physically
among way sport
today others
activities
one's people
football many ancient considered
china competition gymnastics
teams also element
individuals sports include
one certain activity different well
addition greece culture competitions
mind competitive artifacts
place popular famous forms
olympic two games
instance social
participants skills physical

Figure 1. Word Cloud using Test1 (min frequency 2)

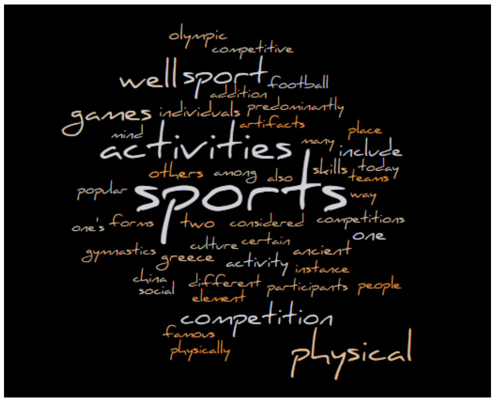

Figure 2. Stylized Word Cloud using Test1 (min frequency 2)

3. Assignment Description

Class 1 – LL – This is a very important class. The project uses a linked list to hold templated pairs. The Nodes of the linked list hold a pair (T,int) meaning whatever T is and always an integer. It templated so that it could store the information about whatever it is designed to hold (our project will only hold strings) and a corresponding quantity. The LL works like a set though – everything in the LL is sorted and whatever the "T" is will always be the key. You cannot have duplicate keys in the linked list. When you insert a new node into the LL, you will need to insert it in order. Therefore, the insert function is by far the most complicated of the project. After a word is inserted into the linked list, if any additional instances of that word are found, instead of adding the word again, the quantity is increased.

You must implement the copy constructor and assignment operator in this class (even if we don't use them for this project). See the LL.cpp file for additional details for other functions that you need to implement including remove. There should be absolutely **NO** references to anything about word clouds or words in this LL!

You should implement this class by itself and then test it completely before using it. There is sample test code at the bottom of LL.cpp. You can uncomment it as you code functions to test it incrementally. Do not forget how we must implement templated classes!

Class 2: WordCloud – For this project, a WordCloud will read in a single file and process it. The processing involves reading all words from a file and inserting them into the linked list. If a word already exists in the linked list, then the quantity of that word is increased instead of adding it again. All punctuation must be removed from the words except if the word requires additional punctuation such as a contraction. Additionally, there is a list of common words (such as "the", "and", or "us") included in the WordCloud.h that needs to be removed from the word cloud. Finally, the user is asked if they would like to remove all words with a frequency of 1. If they respond with YES, Y, yes, or y all words with a frequency (quantity) of 1 are removed from the linked list.

4. Requirements:

This is a list of the requirements of this application. For you to earn all the points you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standards (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **`<iostream>, <fstream>, <iomanip>, <vector>, <cmath>, <ctime>, <cstdlib>, <sstream>`**, and **`<string>`**. You should only use **`namespace std`**.
- You may use **`toupper`**, **`tolower`**, or **`ispunct`** for evaluating individual characters. All words in the **`LL`** and the output files should be lower case.
- Using the provided files, **`LL.cpp, WordCloud.h, proj5.cpp, makefile, proj5_test1.txt, proj5_test2.txt, proj5_test3.txt, and proj5_test4.txt (and a variety of sample runs)`** write the program as described.
- As a reminder, **`LL.cpp`** is templated and all functions must exist in ONE file (**`LL.cpp`**). **`Class`** uses **`LL`** to manage the roster and waitlist but could easily be modified to use practically any type of data. **`LL`** must include a functioning **`copy`** constructor and **`assignment`** operator. The **`LL`** must insert based on the key of the pair (**`first`**). While the LL class COULD hold anything where the comparison operators are overloaded, this project will really only hold strings (even when the input files hold things that look like numbers).
- You can copy the files from my directory in **`/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj5`**.
- Class member variables must be **`private`** for project 5.
- All loops for vectors or strings must use iterators.

- All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.
- The code must not have memory leaks.

5. Recommendations

- We will test your **`LL`** file separately from your application (as well as part of it). It must be fully implemented with a destructor, copy constructor, and an assignment operator. We provided several tests – these are in no way exhaustive but rather just a starting point. You may want to test additional things! Test it first using something like strings. Make sure that the insert function inserts everything in order!
- After the **`LL`** works (including all tests), you can start on **`WordCloud`**. Make sure you can read in the file as part of **`LoadFile`**. Once **`LoadFile`** works, then you can add **`RemovePunct`** (which removes extra punctuation from the front or back of a string). **`RemoveCommon`** can then be implemented. Finally, **`Export`** can be finalized. Don't forget that you can incrementally test your code by using **`Start`** to test one function at a time.
- Finally, after **`RemoveCommon`** works, look at **`RemoveSingles`**. This function (which can use a normal **`for`** loop to iterate over the LL) uses the overloaded [] to see if a node has a frequency of 1. If it does, it removes that node from the LL. For the **`RemoveSingles`** function, when you are iterating over the list the size of the linked list is going to keep changing. If you get jump errors, it is recommended that you reset your for-loop counter to 0 after something is removed.
- Once you have everything running, you can copy your output files into https://worditout.com/word-cloud/create Click on "table" and paste your output file into it and then on "generate" You can see your word cloud now!

6. Sample Input and Output

For this project, there are no input files. Command line arguments have been included (as well as the **`makefile`**) so you can use **`make run1`** or **`make val1`** (all 4 have been included: **`make run1`**, **`make run2`**, **`make run3`**, and **`make run4`** including their val counterparts) to test your code. Here is a sample run:

```
[jdixon@linux5 proj5]$ make run1
./proj5 proj5_test1.txt
Welcome to UMBC Word Clouds
Would you like to remove all words with a frequency of 1?
no
```

```
0 words removed.
1896:1
ability:1
activities:12
activity:3
addition:2
advantageous:1
**A-W Removed for Space**
weightlifting:1
well:8
winning:1
work:1
worldwide:1
wrestling:1
END
What would you like to call the export file?
proj5_test1_output.txt
Output to exportFile proj5_test1_output.txt
238 unique words exported
```

Below is a sample run for proj5_test2.txt.

```
Welcome to UMBC Word Clouds
Would you like to remove all words with a frequency of 1?
yes
627 words removed.
100:3
15:6
20:2
25:2
30:6
4:2
50:3
93:2
97:2
```

```
absolutely:4
absorb:5
absorbed:4
**Removed from document for space**
without:3
work:4
works:2
world:2
wronger:3
years:4
zinc:4
END
What would you like to call the export file?
proj5_test2_output.txt
Output to exportFile proj5_test2_output.txt
320 unique words exported
```

There are several sample runs available in the project directory:

`proj5_output_test1_singles_removed.txt`

`proj5_output_test1.txt`

`proj5_output_test2_singles_removed.txt`

`proj5_output_test2.txt`

`proj5_output_test3.txt`

`proj5_output_test4.txt`

Don't forget to test your copy constructor and assignment operator even though we don't use them in this project!

7. Compiling and Running

As above, there are several make run and make val options available in the provided makefile.

Because we are using dynamic memory for this project, you are required to manage any memory leaks that might be created. Anything that you use "new" for needs to be deleted. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **`valgrind`** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **`valgrind`**:

```
valgrind ./proj5 proj5_test1.txt
```

Note: If you accidently use valgrind make run, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file.

If you have no memory leaks, you should see output like the following:

```
==5606==
==5606== HEAP SUMMARY:
==5606==     in use at exit: 0 bytes in 0 blocks
==5606==   total heap usage: 87 allocs, 87 frees, 10,684 bytes allocated
==5606==
==5606== All heap blocks were freed -- no leaks are possible
==5606==
==5606== For counts of detected and suppressed errors, rerun with: -v
==5606== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```

The important part is "in use at exit: 0 bytes 0 blocks," which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on **`valgrind`** can be found here: http://valgrind.org/docs/manual/quick-start.html

8. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

For this project, you should submit these files to the **`proj5`** subdirectory:

`LL.cpp, proj5.cpp`

`WordCloud.h, WordCloud.cpp`

For this project, you are allowed to edit **`LL.cpp`** but no other header (.h) files.

As you should have already set up your symbolic link for this class, you can just copy your files listed above to the submission folder.

a. cd to your project 5 folder. An example might be cd **`~/202/projects/proj5`**

b. **`cp WordCloud.h WordCloud.cpp LL.cpp proj5.cpp ~/cs202proj/proj5`**

You can check to make sure that your files were successfully copied over to the submission directory by entering the command. Please note: You are responsible for turning in all required files.

```
ls ~/cs202proj/proj5
```

You can check that your program compiles and runs in the **`proj5`** directory, but please clean up any **`.o`** and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here. Uploading of any **`.gch`** files will result in a severe penalty.

For additional information about project submissions, there is a more complete document available in Blackboard under "Course Materials" and "Project Submission."

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj5 folder, it is because the due date has passed. You should be able to see your proj5 files but you can no longer edit or copy the files in to your proj5 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to **`~/cs202proj/proj5-late1`**
- If it is 24-48 hours late, copy your files to **`~/cs202proj/proj5-late2`**
- If it is after 48 hours late, it is too late to be submitted.