CSCI 200 - Fall 2023
Foundational Programming Concepts & Design

A3 - Green Eggs and Ham Classes

Home | HW Sets | Schedule | Files | Help | Links

→This assignment is due by Tuesday, October 10, 2023, 11:59 PM.←

→ As with all assignments, this must be an individual effort and cannot be pair programmed. Any debugging assistance must follow the course collaboration policy and be cited in the comment header block for the assignment.←

→ Do not forget to complete the following labs with this set: L3A, L3B ←

→ Do not forget to complete zyBooks Assignment 3 for this set.←

Jump To: **Rubric Submission**

In this assignment, we will focus on classes, vectors, strings, File I/O, and Functions!

Overview

Have you ever finished a book and wondered, "Geez, I wonder how many times each word occurs in this text?" No? This assignment illustrates a fundamental use of collections: storing related values in a single data structure, and then using that data structure to reveal interesting facts about the data.

For this assignment, you will read in a text file containing the story Green Eggs and Ham (plus some others). You will then need to count the number of occurrences of each word & letter and display the frequencies. You'll be amazed at the results!

RMP-H

The Specifics

For this assignment, download the **starter code pack**. This zip file contains several files:

- `main.cpp` - the predetermined main.cpp. This file shows the usage and functionality that is expected of your program. You are not allowed to edit this file. You will not be submitting this file with your assignment.
- `processor.h` - declaration of function to execute your classes in the expected order
- `processor.cpp` - definition of function to execute your classes in the expected order
- `Makefile` - the preset Makefile to build with your program.
- `input/aliceChapter1.txt` - the first chapter of Alice in Wonderland in text format.
- `input/greeneggsandham.txt` - the contents of Green Eggs and Ham in text format.
- `input/romeoandjuliet.txt` - the contents of Romeo and Juliet in text format.
- `solutions/aliceChapter1.out` - the expected output when running your program against the `aliceChapter1.txt` file
- `solutions/greeneggsandham.out` - the expected output when running your program against the `greeneggsandham.txt` file
- `solutions/romeoandjuliet.out` - the expected output when running your program against the `romeoandjuliet.txt` file

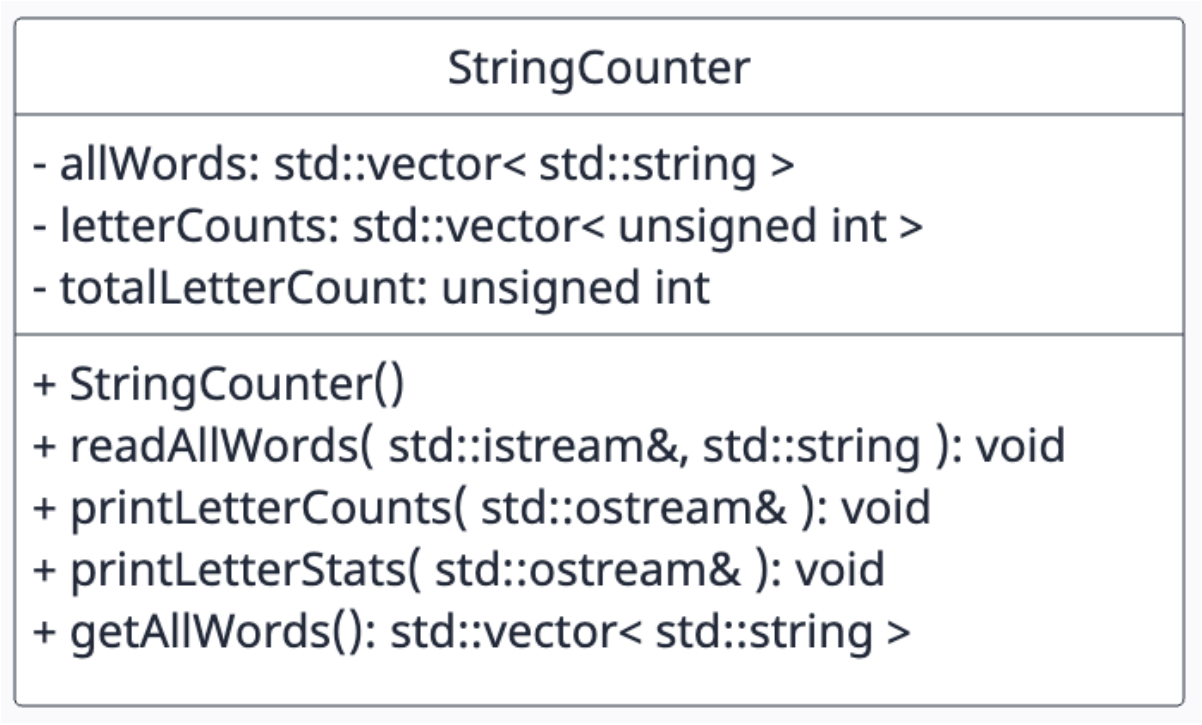
Object Oriented Programming

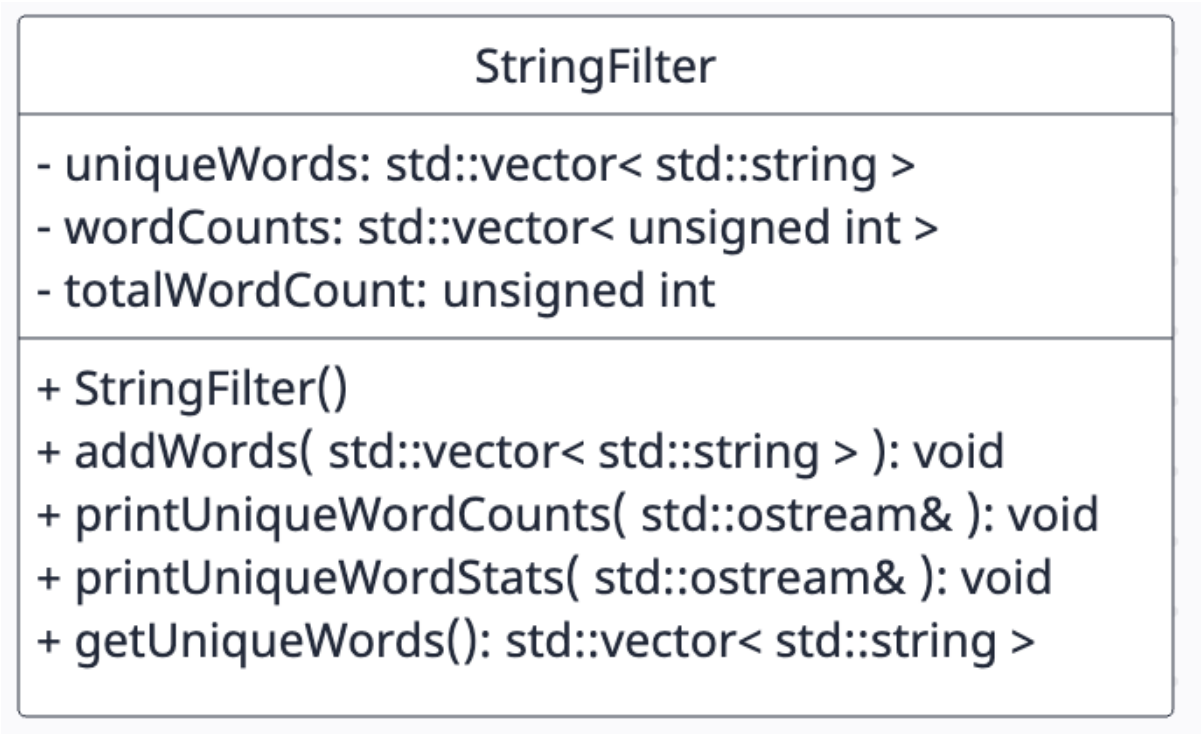
Referring to the implementation in `processor.cpp`, take note how the program reads as a series of subtasks and the provided comments are redundant. The code is "self documenting" with the function names providing the steps that are occurring. Your task is to provide the implementations for all the called functions. You will need to create four files: `StringCounter.h` & `StringCounter.cpp` and `StringFilter.h` & `StringFilter.cpp` to make the program work as intended.

You will want to make your program as general as possible by not having any assumptions about the data hardcoded in. Three public input files have been supplied with the starter pack. We will run your program against a fourth private input file.

Class Requirements

The UML of each class is given below.





The input, output, and task of each member function is described below as well. The functions are:

1. **StringCounter::StringCounter()**
2. **StringCounter::readAllWords()**
3. **StringCounter::printLetterCounts()**
4. **StringCounter::printLetterStats()**
5. **StringCounter::getAllWords()**
6. **StringFilter::StringFilter()**

7. **StringFilter::addWords()**
8. **StringFilter::printUniqueWordCounts()**
9. **StringFilter::printUniqueWordStats()**
10. **StringFilter::getUniqueWords()**

`StringCounter::StringCounter()`

Input: None

Output: N/A

Task: Initializes private data members to sensible values (there are no letters present)

`StringCounter::readAllWords()`

Input: (1) Reference to the input stream (2) a string of characters to remove from any read words

Output: None

Task: Read all the words that are in the input stream and store in the private vector of all words. For each word, remove all occurrences of all the punctuation characters denoted by the punctuation string and convert each character to its upper case equivalent.

`StringCounter::printLetterCounts()`

Input: Reference to the output stream

Output: None

Task: For each letter, print out the letter and its corresponding count to the provided output stream. Format the output as follows:

```
A: #C
B: #C
...
Y: #C
Z: #C
```

Notice how there are two columns. We want the values aligned in each column. The columns correspond to the following values:

1. `A` - The letter
2. `#C` - The corresponding count of the letter. Right align all values. Allocate enough space for the length of the most frequent letter present in the file. (Assume there will be at most 10^{10} occurrences of each letter.)

An example (based on singing Happy Birthday to Bjourne) is shown below:

```
A:  8
B:  5
C:  0
D:  4
E:  1
F:  0
G:  0
H:  8
I:  4
J:  1
K:  0
L:  0
M:  0
N:  1
O:  8
P:  8
Q:  0
R:  5
S:  0
T:  8
U:  4
V:  0
W:  0
X:  0
Y: 11
Z:  0
```

Refer to the solution files for longer examples on the expected formatting.

`StringCounter::printLetterStats()`

Input: Reference to the output stream

Output: None

Task: Print out the two letters that occur least often and most often to the provided output stream. If there is more than one letter that occurs the same number of times, print the one that comes first alphabetically. Print out the following pieces of information:

1. The letter
2. The number of occurrences
3. The frequency of appearance as a percentage to 3 decimal places

Format the output as follows:

```
 Most Frequent Letter: Z #C (#P%)
Least Frequent Letter: A #C (#P%)
```

Notice how there are three columns of values. The columns correspond to the following values:

1. `A` - The letter.
2. `#C` - The corresponding count of the letter. Right align all values. Allocate enough space for the length of the most frequent letter present in the file. (Assume there will be at most 10^{10} occurrences.)
3. `#P` - The frequency of the letter. Right align all values. Print to three decimal places.

An example with actual values is shown below:

```
 Most Frequent Letter: Y 11 ( 14.667%)
Least Frequent Letter: C  0 (  0.000%)
```

Refer to the solution files for longer examples on the expected formatting.

`StringCounter::getAllWords()`

Input: None

Output: A vector of strings containing all the words

Task: The function will return the private vector of strings.

`StringFilter::StringFilter()`

Input: None

Output: N/A

Task: Initializes private data members to sensible values (there are no words present).

`StringFilter::addWords()`

Input: A vector of strings containing all the words

Output: None

Task: The function will compute the unique set of words present in the input vector. It will also count the number of occurrences of each unique word in the entire text. The private vectors will be the same size with element positions corresponding to the same word and count.

`StringFilter::printUniqueWordCounts()`

Input: Reference to the output stream

Output: None

Task: For each word, print out the word and its corresponding count. Format the output as follows:

```
WORD1 : #C
WORD2 : #C
...
WORDN : #C
```

Notice how there are two columns. We want the values aligned in each column. The columns correspond to the following values:

1. `WORD` - The word. Left align all values. Allocate enough space for the length of the longest word present. (Assume the longest word will be at most 20 characters long.)
2. `#C` - The corresponding count of the letter. Right align all values. Allocate enough space for the length of the most frequent letter present in the file. (Assume there will be at most 10^{10} unique words.)

An example (based on singing Happy Birthday to Bjourne) is shown below:

```
HAPPY    : 4
BIRTHDAY : 4
TO       : 4
YOU      : 3
BJOURNE  : 1
```

Refer to the solution files for longer examples on the expected formatting.

`StringFilter::printUniqueWordStats()`

Input: Reference to the output stream

Output: None

Task: Print out the two words that occur least often and most often. If there is more than one word that occurs the same number of times, print the one that is encountered first. Print out the following pieces of information:

1. The word
2. The number of occurrences
3. The frequency of appearance as a percentage to 3 decimal places

Format the output as follows:

```
 Most Frequent Word: WORD1 #C (#P%)
Least Frequent Word: WORD2 #C (#P%)
```

Notice how there are three columns of values. The columns correspond to the following values:

1. `WORD#` - The word. Left align all values. Allocate enough space for the length of the longest word present. (Assume the longest word will be at most 20 characters long.)
2. `#C` - The corresponding count of the word. Right align all values. Allocate enough space for the length of the most frequent letter present in the file. (Assume there will be at most 10^{10} occurrences.)
3. `#P` - The frequency of the word. Right align all values. Print to three decimal places.

An example with actual values is shown below:

```
 Most Frequent Word: HAPPY   4 ( 25.000%)
Least Frequent Word: BJOURNE 1 (  6.250%)
```

Refer to the solution files for longer examples on the expected formatting.

`StringFilter::getUniqueWords()`

Input: None

Output: A vector of strings containing all the unique words

Task: The function will return the private vector of strings.

Extra Credit

For extra credit, sort the unique words and their associated counts. Sample outputs are provided and denoted by `solutions/*_xc.out`. The sample output for singing Happy Birthday is below:

```
BIRTHDAY : 4
BJOURNE  : 1
HAPPY    : 4
TO       : 4
YOU      : 3
 Most Frequent Word: BIRTHDAY 4 ( 25.000%)
Least Frequent Word: BJOURNE  1 (  6.250%)
```

Notice the additional change in the most frequent word selected.

Functional Requirements

- You may not make use of the standard library functions `sort()`, `find()`, `any_of()` or anything else from `#include <algorithm>`. You must implement your own functions.
- DO NOT use global variables.
- You must use parameters & class members properly.
- Mark parameters and member functions as const appropriately if the function is not modifying a value.
- For this assignment, the output must match the example solutions exactly. The public provided test files are expected to match the provided output files exactly. The private test file will need to generate the expected output as well.

Hints

- Do not wait until the day before this is due to begin.
- The first step is to create the files and class function stubs to get the program to compile and run.

- The second step is to implement each function one at a time. Verify the function is correct before moving on to the next function.
- **Do not** just dive into the assignment. Create a mental plan of what tasks your program needs to accomplish. Convert this to pseudocode. Tackle the first task (eg, "can I open the file ok?") and conduct a sanity check. Then tackle the next task (eg, "can I read all the words in the file, and store the frequencies of each word?") and conduct another sanity check. We strongly suggest writing your program (one step at a time!)
- You may modify `main.cpp` or `processor.cpp` to verify each step is working properly but you will not be submitting either of these files. Be sure your classes work with the expected provided files.
- You may add additional functions to assist if you deem it necessary. A common task is determining how many digits are present in an integer.

Grading Rubric

Your submission will be graded according to the following rubric:

Points	Requirement Description
0.5	Submitted correctly by Tuesday, October 10, 2023, 11:59 PM
0.5	Project builds without errors nor warnings.
2.0	**Best Practices** and **Style Guide** followed.
0.5	Program follows specified user I/O flow.
0.5	Public and private tests successfully passed.
2.0	Fully meets specifications.
6.00	**Total Points**

Extra Credit Points	Requirement Description
+0.5	Words sorted upon output.

Submission

Always, **always**, **ALWAYS** update the header comments at the top of your main.cpp file. And if you ever get stuck, remember that there is LOTS of **help** available.

Zip together your `StringCounter.h, StringCounter.cpp, StringFilter.h, StringFilter.cpp` files and name the zip file `A3.zip` . Upload this zip file to Canvas under A3.

→This assignment is due by Tuesday, October 10, 2023, 11:59 PM.←

→ As with all assignments, this must be an individual effort and cannot be pair programmed. Any debugging assistance must follow the course collaboration policy and be cited in the comment header block for the assignment.←

→ Do not forget to complete the following labs with this set: L3A, L3B ←

→ Do not forget to complete zyBooks Assignment 3 for this set.←

Last Updated: 10/06/23 14:57

Any questions, comments, corrections, or request for use please contact jpaone {at} mines {dot} edu.

[Jump to Top] [Site Map]