CPSC 131 Fall 2018

Project 5: Concordance

(using std::multimap<T, U, Comparator>), use #include <map>

Introduction

The Standard Library provides sets, multisets, maps, and multimaps, which are all typically implemented as balanced binary search trees. Maps and multimaps are associative containers, that associate a key with a value. Multimaps are different from maps in that they allow multiple values for a given key.

Concordance

A concordance is an alphabetical list of the words in a text, with citations of the passages concerned. It is more useful if the words are first sorted by length, then alphabetically. Here is a smaller sample output of the program you will write. (Compare it to your own writing.)

```
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
// MAKING CONCORDANCE with FILE: hp1_page1.txt......
a: 2 3 4 6 7 8 10
as: 2 5 7 10
at: 9 10
be: 1 5 7
he: 2 7 10
if: 3 5
in: 1 2 4 5
it: 3 5
mr: 1 2 7 9 10
no: 2
of: 1 2 8 10
on: 2 7 9
or: 1
so: 2
to: 1 5 7 9
up: 7 9
all: 7
and: 1 2 3 5 7 9 10
any: 2
big: 2
boy: 2 6
but: 3 6 9
car: 10
did: 2
for: 3 6 7
got: 10
had: 2 3 6
her: 2 4 5
him: 6
his: 7 9 10
man: 2
met: 3
mrs: 1 2 3 4 7 9
now: 10
our: 7
out: 3 7 10
owl: 8
say: 1 5
she: 2 4 7
sky: 7
son: 2 6
the: 1 2 3 5 6 7 8 9 10
```

The numbers next to each word indicate the line numbers at which the word was found.

Objective
You are given a partial implementation of a concordance class. It contains the class declaration, the minimum and maximum length of words to analyze (e.g., from 4 letter words to 12 letter words), a multimap (std::multimap<std::string, size_t, StringLenCmp>), and a vector (std::vector<size_t>). The StringLenCmp class is provided. The vector counts how many unique words of each length occurred (it doesn't count the number of times that word occurred). The Comparator class only defines operator(), sometimes called the function call operator. As you can see from the code, its only job is to tell the multimap to sort the entries first by word length, then alphabetically. Nothing to it!

Your job is to complete the concordance class (concordance.h), so that it correctly calculates the number of words in three separate pieces of text, all from the book: *Harry Potter and the Sorceror's Stone*. The first piece of text is the first paragraph from the book. The second piece is the first page (and a bit more to finish the paragraph). The last piece is the entire first chapter from the book.

Source Code Files
You are given "skeleton" code files with declarations that may be incomplete and without any implementation. Implement the code and ensure that all the tests in main.cpp pass successfully.

- concordance.h: This is to be completed
    - Note: the functions to read from a file and split a line into words is already given to you. You need to implement the remaining 4 public member functions.
- main.cpp: This calls a test function (in main.cpp) that tests the output of your concordance class. You may wish to add additional tests, based on the ones already provided.
- README.md: You must edit this file to include the name and CSUF email of each student in your group. Do this even if you are working by yourself. This information will be used so that we can enter your grades into Titanium.

Hints
As you start implementing the concordance class, comment out the tests in main.cpp that you haven't implemented yet in your class. As you fill in the code for your class, uncomment the matching tests to make sure your class is working. Keep testing your code as you implement it. It is much easier to debug one method in your class than all of the methods at the same time.

Do not wait till the very end to test your code. Speak to your teachers if you need help designing your approach, or are having trouble compiling or debugging your code. const errors can frequently prevent your code from compiling. Note also that it is possible to be 95% of the way finished, but have the impression you are miles away. Your instructor can help you get over that final 5% if you need help. Don't hesitate to ask.

Expected output
There are two types of output: 1) the counts of words of each length and 2) the list of words sorted by length along with their page numbers.

Word counts

When you complete your project, all of the tests of word counts should run correctly. Expected output is shown below, for the concordance of all three input files.

```
hp1_paragraph words with length...
PASSED ...para words with length  1: expected and received 0
PASSED ...para words with length  2: expected and received 6
PASSED ...para words with length  3: expected and received 5
PASSED ...para words with length  4: expected and received 11
PASSED ...para words with length  5: expected and received 4

PASSED ...para words with length  6: expected and received 6
PASSED ...para words with length  7: expected and received 3
PASSED ...para words with length  8: expected and received 3
PASSED ...para words with length  9: expected and received 1
PASSED ...para words with length 10: expected and received 1

PASSED hp1_para total words: expected and received 40


hp1_page words with length...
PASSED ...page words with length  1: expected and received 1
PASSED ...page words with length  2: expected and received 15
PASSED ...page words with length  3: expected and received 30
PASSED ...page words with length  4: expected and received 48
PASSED ...page words with length  5: expected and received 30

PASSED ...page words with length  6: expected and received 40
PASSED ...page words with length  7: expected and received 20
PASSED ...page words with length  8: expected and received 17
PASSED ...page words with length  9: expected and received 9
PASSED ...page words with length 10: expected and received 2

PASSED ...page words with length 11: expected and received 0
PASSED ...page words with length 12: expected and received 1
PASSED ...page words with length 13: expected and received 0
PASSED ...page words with length 14: expected and received 0
PASSED ...page words with length 15: expected and received 0

PASSED hp1_page1 total words: expected and received 213


hp1_chapter words with length...
PASSED ...chapter words with length  1: expected and received 3
PASSED ...chapter words with length  2: expected and received 28
PASSED ...chapter words with length  3: expected and received 96
PASSED ...chapter words with length  4: expected and received 222
PASSED ...chapter words with length  5: expected and received 205

PASSED ...chapter words with length  6: expected and received 207
PASSED ...chapter words with length  7: expected and received 184
PASSED ...chapter words with length  8: expected and received 122
PASSED ...chapter words with length  9: expected and received 62
```

```
PASSED ...chapter words with length 10: expected and received 26

PASSED ...chapter words with length 11: expected and received 12
PASSED ...chapter words with length 12: expected and received 7
PASSED ...chapter words with length 13: expected and received 3
PASSED ...chapter words with length 14: expected and received 1
PASSED ...chapter words with length 15: expected and received 0

PASSED hp1_chapter1 total words: expected and received 1179

PASSED Total tests passed: expected and received 43

     ...done.
```

Printing the concordance

In addition to the automated tests of word counts, you are also expected to print the list of words and page numbers by iterating through the multimap. The challenge is to print only once and not to repeat page numbers. Sample output is shown in this file.

Obtaining and submitting code

Click the assignment link to fork your own copy of the skeleton code to your PC.

https://classroom.github.com/g/iLilP4cS

Development environment

The test platform is Linux with the g++ -std=c++14 compiler. For this reason, the recommended development platform is Linux.

Linux environment

To attempt to compile the test program, use the following command:

```
clang++ -g -std=c++14 main.cpp -o test
```

To attempt to run the compiled test program, use the following command:

```
./test
```

Grading rubric

Your grade will be comprised of two parts, *Form* and *Function*. *Function* refers to whether your code works properly as tested by the main function (80%). *Form* refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is **November ~~28th~~ 30th at 11:55pm**.