CSE 232 Spring 2014

Programming Project #7

Assignment Overview
In this assignment you will practice writing your own template, generic algorithms, along with more experience with things like maps, vectors, file input and output etc.
This assignment is worth 50 points (5.0% of the course grade) and must be completed and turned in before 11:59PM on Monday, March 17th .

Background
We are going to do some document comparison, not unlike what we did in the Python class. The overall idea is to read in two small files and report the following:
1. The frequency of word occurrence each file
2. The words common to the two files
3. The words unique to each of the two files

Restrictions
You are not permitted to use the STL `set` container. You are also not permitted to use either the STL `set_intersection` or `set_difference` algorithms. You can test with them, but ultimately what you turn in cannot use them.

Common Words
It should be pretty obvious that the most common word in almost any English document would be words like "a", "and", "the", "this" or the like. They are so common and impart so little information that they are called **stop words** and are often ignored in text counts. I provide a file called stopWords.txt to be used as a source of stopwords. Do not count any word in the stopWords.txt file in any operations on your files.

The General Idea
The general idea is to:
- read in each word from the file
    - for each word, lower case it and strip away any non-alphanumeric at either end of the word
    - check to make sure the word is not in the stop_word list.
- put the words from each file in a map for that file, with the key being the word and the value being the frequency of that word in the file.
- do set intersection to find the common words between two maps of word frequency.
- do set difference to find the unique words in two maps of word frequency.

Assignment
You may remember that the STL provides a `set` data structure, but that data structure does not have methods for intersection or difference. However, the STL does provide *generic algorithms* for that purpose. To gain more experience with generic algorithms, you must write **your own set intersection and set difference generic algorithms** for the task at hand. You can test with the STL versions, but you must write your own for this project.

Functions

- `string` **`lower_and_strip`**`(string s)`
  - takes the string `s`, strips any non-alphanumeric character from the beginning or end of the string (only the ends, not it the middle), lowercases the resulting string and returns it.
- `void` **`read_stopwords`**`(vector<string> &v, string file_name)`
  - reads in the words from the stopWords.txt file and stores them in the vector.
- `void` **`read_file`**`(map<string, long> &m,`
  `vector<string> &stop_list,`
  `string file_name)`
  - reads the words from `file_name`
  - applies `lower_and_strip` to each word
  - if the word is not in the `stop_list`
    - adds the word to the map and updates the map frequency
- `string` **`print_map`**`(map<string, long> &m,`
  `string order="alpha")`
  - if order == "alpha", return a string of the map pairs in alphabetic order
  - if order== "count", return a string of the map pairs in frequency order
- `vector<string>` **`sorted_words`**`(map<string, long> &m)`
  - returns a vector of the words from the map in sorted order.
  - you can use the STL `sort` function
- `template<typename Itr1, typename Itr2, typename Out>`
  `void` **`my_set_intersection`**`(Itr1 source1_begin, Itr1 source1_end, Itr2 source2_begin, Itr2 source2_end, Out dest)`
  - `Itr1` and `Itr2` are input iterators associated with two, sorted, containers
  - `Out` is an output iterator where the results will be placed.
  - Finds those elements common in the two containers
  - collects the common elements in the `dest` container
- `template<typename Itr1, typename Itr2, typename Out>`
  `void` **`my_set_difference`**`(Itr1 source1_begin, Itr1 source1_end, Itr2 source2_begin, Itr2 source2_end, Out dest)`
  - `Itr1` and `Itr2` are input iterators associated with two, sorted, containers
  - `Out` is an output iterator where the results will be placed.
  - Finds those elements unique in the first container.
  - collects the unique elements in the `dest` container

Requirements

As mentioned, we have provided a `main.cpp` which will test each of your functions. Your code must compile and produce correct output using our main **without modification.** We have provided three files:

- Gettysburg.txt (the Gettysburg address)

- Preamble.txt (First two full paragraphs of the Declaration of Independence)
- stopWords.txt (stopword list, one word per line).

Deliverables

`functions.h` and `functions.cpp` -- your source code solution (remember to include your section, the date, project number and comments). Remember, you ***do not provide*** `main.cpp`

1. Please be sure to use the specified file names
2. Save a copy of your file in your CSE account disk space (H drive on CSE computers).
3. You will electronically submit a copy of the file using the "handin" program: http://www.cse.msu.edu/handin/webclient

Assignment Suggestions

The elements below are suggestions on how to accomplish your goals. Feel free to implement the provided function declarations any way you want, but these might be of some help. Remember, you can test with STL set or STL set_intersection and STL set_difference, but ultimately you must not use them in what you turn in.

- `lower_and_strip` : Here are a list functions that will help with this. Assume a string `s = ",,AbC- "` (space at the end)
  - there is no special variable string that contains all the alphanumeric characters (like there was in Python). So do it yourself. Make a string with all the alphanumeric characters and name it `target`.
  - `s.find_first_of(target)`. Find the index of the first character in `s` that is contained in target (index 2 in this example)
  - `s.find_last_of(target)`. Find the index of the last character in `s` that is contained in target (index 4 in this example)
    - methods return `string::npos` if no target character occurs in `s`
  - `s.substr(position, length)` . Make a substring starting a `position` that is of length `length` .
  - `#include<cctype>` , `tolower(ch)` . This is a C function, not a C++ function so no using statement. Just do the include. `tolower(ch)` returns a character that is the lower case version of `ch` . Look at pg. 92 of the book.

- `read_stopwords` : The stopword file has one word per line. All the words are lower case. If you use >> from the file object you get each word with the `'\n'` already stripped.
- `read_file` : Couple things to note:
  - if you apply `lower_and_strip` to a word you could be left with an empty string (particularly in Gettysburg.txt). Check for that.

    - you can use the `find` algorithm to determine if a file word is in the `stop_list`. Look up the form of the `find` algorithm (online, Example 9.1, in book pg. 376, 871)
- `print_map`: This is going to be some work but let's break it down
    - Easiest to work with a vector, so make a vector of pairs (what type?) and collect all the pairs from the map
    - sort the vector
        - the sort for alphabetic just works, using the first element of each pair (a string) to sort the vector
        - sorting on the frequency requires that you write a function for the sort that uses the second element of each pair as the comparison element. If you want to get fancy, sort on the string if the counts are equal. Look online, example 9.3, book pg 386, 876
    - convert the sorted vector to a string
        - use the transform algorithm
        - output to an ostringstream (it is an output stream like cout)
        - trim and return
        - Look online, examples 10.*
- `sorted_words` : Not very hard given what has been done. Collect the words from the map in a vector and sort
    - if you want to use the STL `set_intersection` and `set_difference` algorithms for comparison (and you **absolutely should**) they require a sorted list.
    - whether you take advantage of that in **your** intersection and difference algorithms is up to you.
- `my_set_intersection` : this is surprisingly short but it takes some care. Things to note
    - first 2 args are the begin and end iterator of the first vector of string
    - second 2 args are the begin and end iterator of the second vector of string
    - the last arg is an output iterator
    - for each word in the first vector you use the `find` algorithm to see if it also occurs in the second vector
    - if the word occurs in both, you set the output iterator (a pointer type remember) to have that value and then increment the iterator.
- `my_set_difference` : virtually the same as intersection