University of Wollongong

School of Computing and Information Systems

CSCI251/851 **Advanced Programming** **Autumn 2017**

Assignment 4 **(Due: 11.59pm Week 10, Tuesday 9 May)** **8 marks**

Aim:

This assignment is to familiarise you with the use of STL container classes in your programs.
On completion you should know how to:

- Write C++ code using STL container class objects.
- Devise programs requiring data manipulation with STL containers.
- Gain experience writing and debugging complex C++ programs incrementally.

Prerequisites:

Before you undertake this assignment please review the week 5 lecture notes on templates and the week 6&7 lecture notes on containers and the Standard Template Library (STL). Also, download and study the week 7&8 STL example programs in the Examples folder. The following links may also prove useful for working with STL containers:

http://www.cplusplus.com/reference/stl/
http://www.cprogramming.com/tutorial/stl/stlintro.html
http://www.sgi.com/tech/stl/stl_introduction.html

If you are unsure on how to do something with STL, try googling "STL" with "set" or "map" or "vector" etc.

Requirements:

For this assignment you are to implement a C++ program that can read a text file and display any known words on the screen together with their frequency and position in the file. Below shows a snippet of what the output should look like:

```
    Word     Count    Position(s)
       a       7      22 65 87 96 130 148 165
     all       1      50
     and       9      19 70 125 134 138 157 177 247 261
     are       1      74
      as       2      169 214
automata       1      73
      be       1      99
   begin       1      106
 between       1      132
    cell       5      97 131 156 180 238
cellular       1      72
  create       1      143
  cursor       2      119 124
   eight       1      171
   every       2      155 237
   first       2      20 110
```

A "known" word is a word that is in the English dictionary which is defined by the words listed in the file named "dictionary.txt". The word "Positions" are determined by all the places where each word occurs in the data file. For example, in the above screen output, the word "first" occurred twice, at the 20th and 110th word position in the test data file.

A partially completed "WordStat" class is provided in "wordstat.h" and "wordstat.cpp". A driver program is also provided in main.cpp. The file: "output.txt" shows the output your completed program should produce when run with redirected input from "input.txt". You should do this assignment incrementally by completing the following steps.

Step 1 (1.5 marks)

Implement the ReadDictionary() and DisplayDictionary() public member function in the "wordstats" class in "wordstat.h & "wordstat.cpp". ReadDictionary should open "dictionary.txt" and read all the words into the private "Dictionary" member:

```
      set<string> Dictionary;
```

If you are unsure on using the STL set container, take a look at "08-set.cpp" in the examples folder. DisplayDictionary() should display the contents of the Dictionary on the screen 10 words at a time. The screenshots below shows an example interaction of reading and displaying the dictionary using menu options '1' & '2'.

```
Command > 1

25133 words in dictionary.

Command > 2

A&M
A&P
AAA
AAAS
AAU
ABA
AC
ACM
ACS
AK
Continue? (y/n): y

AL
AMA
ANSI
APS
AR
ARPA
ASTM
AT&T
AZ
Aarhus
Continue? (y/n): n
```

Step 2 (1.5 marks)

Implement the ReadTextFile() public member function. ReadTextFile() should ask the user for the text filename, read the contents of the file into the "KnownWords" and "UnknownWords" data members of the "WordStats" class:

```
      WordMap KnownWords;
      WordMap UnknownWords;
```

Please note that "WordMap" is typedefed in "wordstat.h" as:

```
   typedef map<string, vector<int> > WordMap;
```

Any words that are found in the Dictionary are placed in the "KnownWords" WordMap. Any words that are not found are placed in the "UnknownWords" WordMap. After this is done, the number of words read into the "KnownWords" and UnknownWords" containers should be displayed on the screen. Before attempting to lookup a word in the Dictionary, you should first preprocess the word by converting all characters to lower case and remove any tailing punctuation marks like full stop (.), comma (,), question mark (?), etc.
If you are unsure on working with map containers look at "08-map.cpp" in the example folder.

Step 3 (1.5 marks)

Implement the DisplayKnownWordStats(); public member function. This function should iterate the "KnownWords" WordMap and display the word stats on the screen as shown on page 1. DisplayKnownWordStats() can be tested with menu option '4'.

Step 4 (0.5 marks)

Implement the "DisplayUnknownWordStats()" public member function by declaring a private member function named: "DisplayWordStats(WordMap &WMap);" and placing the code from "DisplayKnownWordStats()" into this function. By changing "KnownWords" to "Words" you should be able to call this function within the "DisplayKnownWordStats()" and "DisplayUnknownWordStats()" function and by passing the appropriate map container to it.

Step 5 (1.5 marks)

Implement the "DisplayMostFreqKnownWords()" public member function. This function should display the 10 most frequently occurring words in the "KnownWords" container. E.g.

```
Command > 6

                Word    Count
                 the     21
                live     10
                  of      9
                 and      9
                  or      7
                  in      7
                   a      7
                  to      6
                  is      6
                cell      5
```

To do this declare a local multimap<int,string>; container. You will need to iterate the "KnownWords" container and insert the size of the vector (as the key) and the word into the local mutlimap. This multimap can then be iterated to display the 10 most frequent words on the screen.

Step 6 (0.5 mark)

Implement the "DisplayMostFreqUnknownWords()" public member function. Do this the same as you did with Step 4. i.e. by declaring another private function:

```
        void DisplayMostFreqWords(WordMap &Words)
```

and copying the code from "DisplayMostFreqKnownWords()" into this function, etc.

Step 7 (1.0 mark)

Implement the "DisplayOriginalText()" public member function. To do this declare a local map<int,string>; container and do the same as you did with Step 6, except you should also iterate the vectors in the KnownWords and UnknownWords containers and add the <position, word> pair for all words and their positions. This will sort the words into their original text order based on the word position numbers. Test this function with menu option '8'. The screen output should look something like below.

the game of life written by ian sharpe the game of life was invented by the mathematician john conway and first reached a wide public when it was written up in scientific american in 1970 or thereabouts in those days it was mostly played on squared paper nowadays computers take all the hard work out of this fascinating invention to some it's nothing more than a toy to others it and related cellular automata are subjects for serious study in this implementation the screen is divided into a grid of cells 40 wide by 24 deep a cell may be live (red or dead (white) you begin by creating the first generation of live cells or seed . . .

Step 8 (for CSCI851 students. 1 mark deduction if not done.)

Provide and implement menu options ‘9’ for adding a new word to the dictionary. You should prompt the user for the new word and check that the entered word is not already in the dictionary. If it is in the dictionary, display an error message, otherwise add the new word to the dictionary and save the dictionary words to the file: “dictionary.txt”.

Submit:

Submit your files (and the output produced (copy & paste) when the text in input.txt is entered in response to the user prompts) using the submit facility on UNIX as shown below:

```
$ submit -u login -c CSCI251 –a4 main.cpp wordstats.h wordstats.cpp output.txt
```

where 'login' is your UNIX login ID
Note: CSCI851 should also submit to –c CSCI251.

<u>You must demonstrate your program in the lab by week 11. Failure to demo on time will result in a 1 mark deduction for each week late.</u>

Deductions will be made for untidy work or for failing to comply with the submission instructions. Requests for alternative submission arrangements will only be considered before the due date. An extension of time for the assignment submission may be granted in certain circumstances. Any request for an extension of the submission deadline must be made to the Subject Coordinator before the submission deadline. Supporting documentation must accompany the request for any extension. Late assignment submissions without granted extension will be marked but the marks awarded will be reduced by 1 mark for each day late. Assignments will not be accepted if more than three days late.