CSCI203/CSCI803 ASSIGNMENT 1

(10 marks + 2 demo marks)

Step-1 demo due during your Week-4 Lab class
Final submit due Week-6, Fri 11:59pm 6 Sept.

You are to write a word processing program in one file (only) named: ass1.cpp or ass1.java, etc. Your main() procedure should record the start time of your program and, at the end of processing, display the total run time of your program (in seconds). You should do this work in steps, as described below. You may use any data structures and algorithms that have been presented in class up to the end of week 4. If you use other data structures or algorithms, appropriate references must be provided. **Note: You will only receive the 2 marks for step-1 if you complete and demo Step-1 in your lab class in week-4.**

Step-1 *(Week-4 demo, 2 marks)*

Write a program that reads the dictionary file (named: "dictionary.txt) and places the words in an appropriate array and then prints the number of words read on the screen. (You can assume that the dictionary contains no more than 400,000 words and no word is longer than 35 characters.) Then write a function that uses a linear search to find the first 10 palindromes in the dictionary array and display them on the screen. (Note: A palindrome is a word larger than 1 character that reads the same both forward and backward like: "racecar".) Example output:

```
Number of words in dictionary: 370103
First 5 palindromes:
aa : aa
aaa : aaa
aas : saa
ab : ba
aba : aba

Total run time (secs): 1
```

Note: You will receive the 2 demo marks no matter how your search for the palindromes in the array is done. Make sure your search stops when the first 5 palindromes are found.

Step-2 *(More palindromes, 2 marks)*

Before you commence Step-2, estimate how long it would take for your step-1 linear search to find all the palindromes in the dictionary? Now, replace the linear search with a binary search and estimate the time to find ALL the palindromes in the dictionary. Write your estimate of the speedup factor in the report in Step-5. Print on the screen: the first 10 palindromes found (similar to step-1) and the longest palindrome found. (Note: if there is more than one palindrome with the longest length, print the first one only.) Implement other speed enhancements if you can think of any.

Step-3 *(Spell-check, 3 marks)*

Read the input data file: "sample.txt", pre-process it (as explained below) and search for each word (once only) in the dictionary. Then print on the screen the total number of valid words read, the number of unique words read, and the number of unique words read that were found in the dictionary. You should store all unique words that were found in the dictionary in a suitable array.

Note: To pre-process the words read from "sample.txt", all capitals should be converted to lower case. Punctuation marks in words should be removed and treated as a single word. Thus, *it's* will become *its*, *you'll* will become *youll* and *loop-hole* will become *loophole*. Any word that becomes zero characters as a result of the pre-processing should be rejected it as an invalid word.

Step-4 *(Anagrams, 3 marks)*

Use the dictionary to find anagrams of the unique words stored in the array from step-3. Print the first 10 anagram words together with their anagrams in alphabetic order e.g.:

```
admire: armied damier dimera merida
after: afret frate trefa
....
```

Also, print the word with the most anagrams, the longest word with anagram(s), the total number of words with anagram(s) and the total number of anagrams found. Optimise your code so that it completes this task as quickly as possible.

Step-5 *(Specifications, 2 marks)*

In a comment block at the bottom of your program, write the following details in no more than 20 lines of text. State the run time of your final program and what machine this was on. Quote the speedup achieved in step-2. List the data structures and algorithms used by your program to spell-check, find palindromes and find anagrams. Also include any other enhancements you did to speed up your program (if any). For this step, marks will be awarded based on how accurately and clearly you describe your program.

Marking Guide:

Programs must compile and run on banshee under gcc (C programs), g++ (C++ programs), javac (Java programs) or python (python programs). Programs which do not compile and run on banshee will receive no marks. Marks may be deducted for untidy or poorly designed code. Appropriate comments should be provided where needed. All coding and comments must be your own work. Inbuilt standard libraries or data structures and algorithms, such as STL should not be used. You may use *string* or *String* type for storing the words, if you wish.

Submission:

Assignments should be typed into a single text file named "ass1.*ext"* where *"ext"* is the appropriate file extension for the chosen language. You should run your program and copy and paste the output into a text file named: "output.txt"

Submit your files via the *submit* program on banshee:

`submit -u user -c csci203 -a 1 ass1.ext output.txt`

- where *user* is your unix userid and *ext* is the extn of your code file.

Late assignment submissions without granted extension will be marked but the points awarded will be reduced by 1 mark for each day late. Assignments will not be accepted if more than five days late. 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 via SOLS before the submission deadline. Supporting documentation should accompany the request for any extension.