INFO0004-2: Anagram Generator

Pr Laurent MATHY, Cyril SOLDANI

March 2, 2019

1 Introduction

In this assignment, you will design and implement an anagram generator.

Your anagram generator will use the words of a dictionary file to find all the unique combinations of words that use the same letters as a given input string.

E.g., looking up the anagrams of “hikswy” could return “whisky” and “ski why”, amongst others.

2 Interface

Your anagram generator will be used through two functions:

```
Dictionary create_dictionary(const std::string& filename);

std::vector<std::vector<std::string>>
anagrams(const std::string& input, const Dictionary& dict, unsigned max);
```

- the `create_dictionary` function takes the name of the dictionary file as argument and returns the corresponding dictionary in whatever form is appropriate for use in the `anagrams` function.

  The dictionary file contains a single word per line, and these words are in alphabetical order.

  To simplify, you can assume that

  - all letters in both your input and dictionary are lowercase ASCII letters;
  - your input contains only letters and spaces.

- the `anagrams` function takes the input as a `string`, the dictionary created from the dictionary file by the `create_dictionary` function, and an `unsigned` parameter indicating the maximum number of words in combinations making up an anagram. A value of 0 for this last parameter indicates no restriction on the number of words allowed.

  This function will return *all* the unique anagrams, containing the specified maximum number of words or fewer, for the input. By *unique anagram*, we mean a combination of words *in alphabetical order*. Each such combination will be stored as a `vector<string>` and all these will be stored in a `vector<vector<string>>`.

These function declarations, as well as a `typedef` defining the `Dictionary` type, must be contained in a header file called `anagrams.hpp`. This file may also contain other lines of code, *but only the strict minimum required* to support the use of this interface.

All the code required for the implementation of this interface must be in a file called `anagrams.cpp`.

You can consult the Standard C++ Library Reference for further documentation about the STL data structures and algorithms, at `http://cppreference.com/`.

3 Remarks

3.1 No new classes or structures

In this project, you **must not** define any new `class` or `struct`. Also try to reuse the STL algorithms as much as possible, rather than re-implementing them.

3.2 Respect the interface

Submission must scrupulously **follow the interface** defined above. You can of course define auxiliary functions as you see fit, but these should be properly hidden from the users of the interface.

3.3 Efficiency

You should pay attention to efficiency. We will test your code with various inputs, using fairly large dictionaries that can contain several hundred thousands of words. Any performance issues will be exposed in this context. Unresponsive code will be **severely penalized**. You must therefore carefully choose your internal data structures and algorithms.

3.4 Readability

Your code must be readable:

- Make the organisation of your code as obvious as possible. Remember you can create as many auxiliary functions as you see fit.
- Use descriptive names for functions and globals.
- Complement your self-documenting code with comments, where appropriate.
- Choose a coding convention, and stick to it. *Consistency* is key.

3.5 Robustness

Your code must be robust. The `const` keyword must be used correctly, sensitive variables must be correctly protected, memory must be managed appropriately, the program must run to completion **without crash** (even if the user provides invalid input).

3.6 Warnings

Your code must compile **without error or warning** with `g++ -std=c++11 -Wall` on `ms8??` machines. However, we advise you to check your code also with `g++ -std=c++11 -Wall -Wextra` or, even better, `clang++ -Weverything -Wno-c++98-compat`.

3.7 Evaluation

Your code will be evaluated based on all the above criteria. Failure to comply with any of the points mentioned in **bold face** can (and most often will) result in a mark of zero!

4 Submission

Projects must be submitted through the submission platform before **Sunday March the 17th, 23:59 CET**.

You will submit a `s<ID>.tar.xz` archive of a `s<ID>` folder containing `anagrams.{c,h}pp`, where `s<ID>` is your ULiège student ID.

The submission platform will do basic checks on your submission, and you can submit multiple times, so check your submission early!