Linköping University
Department of Science and Technology/ITN
Aida Nordman

Programming in C++

TNG033

Lab 3

Course goals

To write programs in C++ using **STL**, **S**tandard **T**emplate **L**ibrary, and define templates. Specifically,

- to use different container classes;
- to use iterators;
- to use algorithms;
- to use `string`-class as a container class; and
- to get acquainted with the online library documentation,
    - cppreference.com (see sections under "*Containers library*", "*Algorithms library*", "*Iterators library*", and "*Numerics library*")
- To define and use function objects to customize algorithms from the standard library.

Preparation

When using **STL**, we advise not to use "`using namespace std;`" in the beginning of the program. Instead, you should use the prefix "`std::`" when using items from the library, as illustrated in lectures 11 to 13.

Lectures 11, 12, and 13 are relevant for this lab, and it is recommended that you look over the examples from these lectures.

- Read the library documentation about function template `std::back_inserter` and `std::back_inserter_iterator` and pay attention to the examples provided there. You are going to need to use `back_inserter` (and `back_inserter_iterator`) in this lab.
- Do the self-study exercises of set 5. Pay special attention to exercises 2, 3, and 5.
- Do exercise 1 before your lab session on week 50.
- Write down the most important questions that you want to discuss with your lab assistant.

Demonstration

The exercises are compulsory and you should demonstrate your solution orally during your **first RE** lab session on **week 51.** After this occasion, if your solution for lab 3 has not been approved then it is considered a late lab. Note that a late lab can be presented provided there is time in a **RE** lab session.

We have booked an extra **RE** lab session on week 51. If by then you have been approved in all labs then you can skip this extra lab session. Otherwise, **one** late lab can be presented during the redovisning time slot assigned to your group.

All groups that have more than one late lab to present on the extra **RE** lab session of week 51 must place themselves in a queue written on the board in the **beginning** of the lab session, at 13:15. When time allows the lab assistant will give opportunity for the next group in the queue to present a second (or third) late lab. However, we stress once more that a late lab can be presented provided there is time in the extra **RE** lab session of week 51.

Necessary requirements for approving your lab 3 are given below.

- Use of global variables is not allowed, but global constants are accepted.
- The code must be readable, well-indented, and follow good programming practices.
- Your solution must not have pieces of code that are near duplicates of each other.
- The compiler must not issue warnings when compiling your code. In Visual Studio, you should set the compiler's warning level to **level4** (`\W4`).
- Modification of the `std` namespace is not allowed.
- Standard library components (e.g. `std::vector`, `std::map`, etc) shall be used whenever possible.
- Standard algorithms must be used, instead of common C++ loops (such as `for`-loop, `while`-loop, or `do`-loop). As an example, make sure to use **STL**-algorithms
  - to copy data from one container to another;
  - to sort;
  - to write data stored in a container to a stream (*e.g.* `std::cout`);
  - to transform the characters in a string.

If you have any specific question about the exercises, then send us an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "`TNG033: ...`".

Exercise 1

You are requested to write a program that creates a frequency table for the words in a given text file. The words in this table should only contain lower-case letters and digits, but no punctuation signs (*e.g.* `.,!?:\"();`). Genitive apostrophe (`'` as in `china's`) is possible though.

The input file contains no special characters (such as å, ä, ö, ü or similar) but punctuation signs and words with upper and lower-case letters may occur in the file (e.g. `China's`). For every word read from the file, all upper-case letters should be transformed to lower-case letters and all punctuation signs should be removed. As usual, words are separated by white spaces.

Your program should perform the following tasks by the indicated order.

1. Read words and add them to a frequency table. Thus, this table keeps for each word the number of occurrences. Easiest way is to use a `std::map`.
2. The frequency table is displayed, in alphabetical order.
3. Finally, display the frequency table, by decreasing order of words' frequency. As lecture 12 shows, a map is always sorted and the order is only based on the key value (i.e. a word). Therefore, all pairings of word and frequency should be stored in a

`std::vector`. Since words with the highest frequency should be displayed first, the vector should be sorted to reflect this.

At most one usual C++ loop (such as `for`-loop, `while`-loop, or `do`-loop) may be used in solution of this exercises.

The files `uppgift1_kort.txt` and `uppgift1.txt` should be used to test the program and the expected output is in the files `uppgift1_kort_out.txt` and `uppgift1_out.txt`, respectively.

Exercise 2

An **anagram** is word whose letters can be rearranged to make up a new word. Example of an anagram is "*listen*" and "*silent*".

A **subject** for a word `w` is the string obtained from `w` by sorting alphabetically its characters. For instance, the subject of word "*listen*" is "*eilnst*". Obviously, anagrams such as "*listen*" and "*silent*" have the same subject.

In this exercise all anagrams in a text file with English words should be identified. Here too, it is easiest to use a `std::map` to store the list of anagrams for every subject. When this is done, the different anagrams for each subject should then be written to an output stream `std::ostream` (like `std::cout` or an output text file).

All words in the input file are in lower-case letters and there are no punctuation signs.

The output should contain the total number of words read from the input file. In addition, only the cases of subjects with two or more anagram words should be listed in the output, with indication of the number of words in each list of anagrams.

No (zero) common C++ loops (such as `for`-loop, `while`-loop, or `do`-loop) can be used in the solution of this exercise.

The files `uppgift2_kort.txt` and `uppgift2.txt` should be used to test the program and the expected output is in the files `uppgift2_kort_out.txt` and `uppgift2_out.txt`, respectively.

Good luck!!