Linköping Institute of Technology
Department of Science and Technology/ITN
Aida Nordman

Programming in C++

TNG033

Lab 4

Course goal

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

- to use different container classes;
- to use iterators;
- to use algorithms; and
- to use **`string`**-class as a container class.

Preparation

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

- Fö 13: **`test_clock_algorithms.cpp`** and class **`Clock`**,
- Fö 13: slides 24 to 29, **`timetable.cpp`** and class **`Clock`**,
- Fö 13: slides 29 to 32, **`stream_iterators.cpp`** and class **`Clock`**.

Also read parts 2.8, 5.11, 12.1, 12.2, 12.5.1, and 12.5.2 in the course book.

It may also be useful to read the library documentation (**`algorithm`** och **`iterator`**) about function template **`back_inserter`** och **`back_inserter_iterator`**.

Demonstration

You should demonstrate your solution orally during your redovisning lab session on **week 51**.

Late labs can only be presented during a redovisning lab session, if time allows. This means that during the redovisning lab session in week 51, priority is given to presenting lab 4. Moreover, if you want to attend a lab session then you **must** go to the lab room that is scheduled for the lab class you signed for in the beginning of the course (see lab groups registrations if you have forgotten to which lab class you signed up for). The schedule for the labs is available from the course web site. We also remind you that at most two late labs can be presented in a lab session.

If you have any specific question about the exercises, then drop me 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: …**".

Your code should satisfy the following requirements.

- Be structured with small functions.
- Use of global variables is not allowed, but global constants are accepted.
- 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 display data stored in a container (to `cout`);
    - to transform the characters in a string.
- **Challenge:** at most one common C++ loop (such as `for`-loop, `while`-loop, or `do`-loop) may be used in every program requested in the exercises below.

Description

The lab is about handling words in a text file, as described below.

- In the first task, a frequency table should be created for the words in a given text file. The words in this table should only contain lower-case words. The file contains no special characters (such as å, ä, ö, ü or similar).
- In the second exercise, all anagrams in a file with English words should be found and then displayed.
- In the last exercise, the anagrams should be sorted such that the word with the highest number of possible anagrams is printed out first.

There are short test files and then real files with a large quantity of words to handle.

Exercise 1

In this exercise the words in a text file should be processed. A word may contain letters and numbers but not punctuation signs (`.,!?:\"();`). Genitive apostrophe (`'` as in. `Dave's`) is possible though.

Reading may be done via redirection of input, as in lab 1, or by opening a file. For every read word, all **upper case** letters should be transformed **to lower case** letters and all **punctuation signs** should be removed. Words are separated by white spaces.

- All read words should be added to a frequency table. Thus, this table keeps for each word the number of occurrences. Easiest way is to use (`map`).
- The frequency table is displayed, in alphabetical order.
- Finally, display the frequency table, by decreasing order of words' frequency. As lecture 13 shows, one cannot sort a `map` on anything other than the key value. Therefore, all pairings of word and frequency should be stored in a vector. Since words with the highest frequency should be displayed first, the vector should be sorted to reflect this.

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.

Uppgift 2

An **anagram** is word whose letters can be rearranged to make up a new word. Example of an anagram is "*listen*" and "*silent*". Both these words are anagrams, and the first word that the other can be created from will be called *subject* below.

In this exercise all anagrams in a text file with English words should be identified. Here too is easiest to use **`map`** to store the anagram for every *subject*. When this is done, the different anagrams for each *subject* should be displayed.

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.

Uppgift 3

The different anagrams should be displayed such that the anagrams for the subject with the highest number of anagrams should appear first. Save the program for this exercise in a file named **`uppgift3.cpp`**.

The files **`uppgift3_kort.txt`** and **`uppgift3.txt`** should be used to test the program and the expected output is in the files **`uppgift3_kort_out.txt`** and **`uppgift3_out.txt`**, respectively.

Good luck!!