

TNG033: Programming in C++ Lab 3

Course goals

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;
- to use `std::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 (and/or lambda functions) 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 the following and pay attention to the examples provided there.
 - Function template [`std::back_inserter`](#) and [`std::back_inserter_iterator`](#).
 - Function template [`std::inserter`](#) and [`std::insert_iterator`](#).
- Do exercises 2, 3, and 5 of the [set 5 of exercises](#).
- Do [exercise 1](#) before the start of lab session *Lab3 HA*.
- Write down the most important questions that you want to discuss with your lab assistant.

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 English or Swedish. Add the course code to the e-mail’s subject, i.e. “TNG033/ED3: ...”.

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 from a given input text file and add them to a frequency table. Thus, this table keeps for each word the number of occurrences and is sorted alphabetically. Use a `std::map` to implement the frequency table (name the map variable as `table`, in your code).
2. Copy all pairings of word and frequency from the map created in the point above into a `std::vector` (name the vector variable as `freq`).
3. Finally, sort the vector decreasingly by the words' frequency.

➡ **At most one** usual C++ loop (such as `for`-loop, `while`-loop, or `do`-loop) may be used in the solution of this exercise. In addition, `std::for_each` algorithm should not be used.

Note that the `main` provided for this exercise ends by calling a function (named `test`) which tests whether the table and the vector created in steps 1 and 3 above have the correct contents.

The files `uppgift1_kort.txt` and `uppgift1.txt` should be used to test the program. Files `out_uppgift1_kort.txt` and `out_uppgift1.txt` show the contents of the map and the vector, for each of the input text files. The contents of the map appear first, followed by a separation line "`STOP -111`", followed by the contents of the vector.

Exercise 2

An **anagram** is a 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.

This exercise has two tasks.

- a) 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). [Hint: `std::for_each` algorithm can be used.]

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 written to the output stream, 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 task.

The files `uppgift2_kort.txt` and `uppgift2.txt` should be used to test the program and the expected output is in the files `anagrams_uppgift2_kort.txt` and `anagrams_uppgift2.txt`, respectively.

- b) Write a test function, similar to the test function used in exercise 1, to check whether the map created in a) has the correct contents. As you probably will notice, the table created for the input file `uppgift2.txt` is quite long to be manually checked that it's correct.

Each line of the files `out_uppgift2_kort.txt` and `out_uppgift2.txt`¹ lists all anagrams for a subject (cases of subjects with one anagram are also included). The lines are listed by alphabetical order of the subject. Note that the subjects are not in these files. Thus, the test function should load the contents of these files into a map. An assertion should then be used to compare this map with the map built in exercise a).

 **At most one** usual C++ loop (such as `for`-loop, `while`-loop, or `do`-loop) may be used in the solution of this task. In addition, `std::for_each` algorithm should not be used.

Demonstration

The exercises in this lab are compulsory and you should demonstrate your solutions during the lab session *Lab3 RE*. Read the instructions given in the [labs web page](#) and consult the course schedule. We also remind you that your code for the lab exercises cannot be sent by email to the staff.

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.
- 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.

Good luck!

¹ File `out_uppgift2_kort.txt` has the map contents created for the input text file `uppgift2_kort.txt`, while file `out_uppgift2.txt` has the map contents created for the input file `uppgift2.txt`.