Linköping university
ITN

TNG033: Programming in C++
Lab 3

Course goals

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

- to use different container classes;
- to use iterators;
- to use algorithms;
- 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;
- to define classes satisfying a set of given requirements.

Preparation

Perform the following tasks before *Lab2 HA*.

- Review lecture 11, lecture 12, and lecture 13.
- Do exercises 2, 3, and 5 of the set 5 of exercises.
- Download the files for this lab exercises from the course website. Then, you can use use CMake to create a project with the files set.hpp, set.cpp, and test.cpp. For how to use CMake, you can see this short guide.
- Read exercise 1, exercise 2, and exercise 3 descriptions.
- Do exercise 1 before the start of lab session *Lab3 HA*.

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 STL library.

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`: …".

Exercise 1

Define a class named `Polynomial` to represent polynomials with integer coefficients. Thus, each instance of the class represents a function of the form $\sum_{i=0}^{n} a_i x^i$, where a_i $(0 \leq i \leq n)$ is an integer and $n \geq 0$ is the polynomial's degree. Each $a_i x^i$ is a polynomial's term, where a_i is the term's coefficient and i is the term's degree.

Class `Polynomial` should implement a polynomial by using a `std::map`, named `coefficients_table`, to store the polynomial's terms such that `coefficients_table[i]` stores the coefficient a_i. Note that only non-zero terms should be stored in the map.

Class `Polynomial` should have the following (basic) functionality.

- Constructor(s) that create a polynomial with one term, as shown below.

```
Polynomial p1{2, 3}; // p1 = 2x^3
Polynomial p2{-8};   // p2 = −8
Polynomial p3;       // p3 = 0
```

- A constructor that creates a polynomial from a vector of terms, as shown below. The given vector is not sorted in any specific way.

```
// Create polynomial p = 2X^1 - 5X^2 + X^6
Polynomial p{{{1, 2}, {2, -5}, {6, 1}}};
```

- Implicit conversations from `int` to an instance of class `Polynomial` should be supported.
- Instances of class `Polynomial` should be copiable and assignable.
- A member function named `degree` that returns the degree of a polynomial.
- A type conversion operator supporting conversion from `Polynomial` to `std::string`, though implicit conversions should not be supported. An example is given below. Pay attention to the placement of the white spaces.

```
-8X^0 + 2X^1 - 5X^2 + 1X^6
```

- The following operators should be overloaded.
  - `operator+=`
  - `operator-=`
  - `operator*=`
  - `operator+`
  - `operator-`
  - `operator*`
  - `operator==`
  - `operator<<`
- Mixed-mode arithmetic expressions involving `Polynomial`s and `int`s should be supported, as illustrated below.

```
std::cout << 2 * p - p + 100;

p -= 1;
```

Design your class with care and follow the given specifications.

Add your code to the files `polynomial.h` and `polynomial.cpp`. File `test.cpp` contains a `main` function to test your class. The expected output is as follows.

```
** Test 1: constructors, conversion to string, and degree()

** Test 2: operator+=

** Test 3:  operator-=

** Test 4:  operator*=

** Test 6:  operator+, operator-, operator*, operator<<

p4 = 256X^2 - 1408X^3 + 1936X^4 + 800X^5 - 2840X^6 + 3616X^7 - 4479X^8 - 1320X^9 + 4180X^10 - 2640X^11 + 3744X^12 + 432X
^13 - 1748X^14 + 400X^15 - 1010X^16 - 40X^17 + 124X^18 + 1008X^19 - 1240X^20 - 300X^21 + 970X^22 - 1392X^23 + 2005X^24 +
 120X^25 - 450X^26 + 240X^27 - 648X^28 - 12X^29 + 90X^30 + 60X^32 + 30X^34 - 180X^35 + 225X^36 + 36X^39 - 90X^40 + 9X^44

** Test 7: mixed-mode operations

Success
```

Requirements for exercise 1

The implementation of class Polynomial must satisfy the following code requirements.

- Special member functions[1] to be allowed, and the compiler can generate, shall be defaulted.
- Class instances should only store non-zero terms.
- `std::for_each` cannot be used.
- Constructors implementation should not have any loops, i.e. no `for`-loops, nor `while`-loops, nor `do`-loops, nor range-based loops. Otherwise, only range-based loops or loops using iterators can be used. Nested loops cannot be used.
- No other functionality other than the one described is allowed. Feel free to add private member functions, if needed.
- You can add extra tests to the `main` function. However, the tests you add must be clearly indicated in the code. You cannot remove any of the given tests, though.

Exercise 2

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 nor quotes. Genitive apostrophe (as in `china's`)[2] is possible, though.

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

Your program should perform the following tasks by the indicated order. The function template `std::back_inserter` can be useful.

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). Count also the number of words read from the input file (use the integer variable named `counter` for this purpose[3]).
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. For words with the same frequency, sort them in alphabetical order.

Note that the `main` provided for this exercise ends by calling a function (named `test`) which tests whether the map (step 1 above) and the vector (steps 2 and 3 above) have the correct contents.

The files `uppgift_kort.txt` and `uppgift.txt` should be used to test the program. Files `out_uppgift_kort.txt` and `out_uppgift.txt` show the contents of the map and the

[1] Copy constructor, assignment operator, and destructor.
[2] Thus, in this case, "`china's`" should be seen as a single word.
[3] Variable `counter` is used as actual argument of the `test` function provided.

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.

The expected output is as follows.

```
Text file: uppgift1.txt
Frequency table sorted alphabetically created ...
Frequency table sorted by frequence created ...

Passed all tests successfully!!
```

Add your code for this exercise to the file `exerc2.cpp`.

Requirements for exercise 2

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.

Exercise 3

The formula below can be used to compute π.

$$\pi = \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left(\frac{1}{16} \right)^n$$

The file `exerc3_given.cpp` contains a program that requests to the user the number of terms $n > 0$ and then computes an approximation of π using the formula above. Understand the given program.

Re-implement the given program but use STL by replacing the `for`-loops by library algorithms. Note that the new program should perform the same steps as the given program and should not contain any loops (e.g. `for`-loop, `while`-loop, etc). In addition, `std::for_each` algorithm should **not** be used. [**Hint**: `std::inner_product` algorithm can be useful.]

Add your code for this exercise to the file `exerc3.cpp`.

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.

- The code must be readable, well-indented, and follow good programming practices.
- The compiler must not issue warnings when compiling your code.
- The proposed solutions must pass all given tests and satisfy the written requirements.