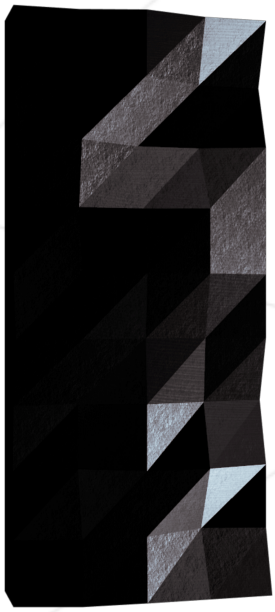

C++ - Module 09

STL

Summary:
This document contains the exercises of Module 09 from C++ modules.

Version: 3.1

Contents

Chapter I

Introduction

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes" (source: Wikipedia).

The goal of these modules is to introduce you to **Object-Oriented Programming**. This will be the starting point of your C++ journey. Many languages are recommended to learn OOP. We decided to choose C++ since it's derived from your old friend C. Because this is a complex language, and in order to keep things simple, your code will comply with the C++98 standard.

We are aware modern C++ is way different in a lot of aspects. So if you want to become a proficient C++ developer, it's up to you to go further after the 42 Common Core!

Chapter II

General rules

Compiling

- Compile your code with `c++` and the flags `-Wall -Wextra -Werror`
- Your code should still compile if you add the flag `-std=c++98`

Formatting and naming conventions

- The exercise directories will be named this way: `ex00, ex01, ... , exn`
- Name your files, classes, functions, member functions and attributes as required in the guidelines.
- Write class names in **UpperCamelCase** format. Files containing class code will always be named according to the class name. For instance: `ClassName.hpp`/`ClassName.h`, `ClassName.cpp`, or `ClassName.tpp`. Then, if you have a header file containing the definition of a class "BrickWall" standing for a brick wall, its name will be `BrickWall.hpp`.
- Unless specified otherwise, every output message must end with a newline character and be displayed to the standard output.
- *Goodbye Norminette!* No coding style is enforced in the C++ modules. You can follow your favorite one. But keep in mind that code your peer evaluators can't understand is code they can't grade. Do your best to write clean and readable code.

Allowed/Forbidden

You are not coding in C anymore. Time to C++! Therefore:

- You are allowed to use almost everything from the standard library. Thus, instead of sticking to what you already know, it would be smart to use the C++-ish versions of the C functions you are used to as much as possible.
- However, you can't use any other external library. It means C++11 (and derived forms) and `Boost` libraries are forbidden. The following functions are forbidden too: `*printf()`, `*alloc()` and `free()`. If you use them, your grade will be 0 and that's it.

- Note that unless explicitly stated otherwise, the `using namespace <ns_name>` and `friend` keywords are forbidden. Otherwise, your grade will be -42.
- **You are allowed to use the STL only in Modules 08 and 09.** That means: no **Containers** (vector/list/map, and so forth) and no **Algorithms** (anything that requires including the `<algorithm>` header) until then. Otherwise, your grade will be -42.

A few design requirements

- Memory leakage occurs in C++ too. When you allocate memory (by using the `new` keyword), you must avoid **memory leaks**.
- From Module 02 to Module 09, your classes must be designed in the **Orthodox Canonical Form, except when explicitly stated otherwise**.
- Any function implementation put in a header file (except for function templates) means 0 to the exercise.
- You should be able to use each of your headers independently from others. Thus, they must include all the dependencies they need. However, you must avoid the problem of double inclusion by adding **include guards**. Otherwise, your grade will be 0.

Read me

- You can add some additional files if you need to (i.e., to split your code). As these assignments are not verified by a program, feel free to do so as long as you turn in the mandatory files.
- Sometimes, the guidelines of an exercise look short but the examples can show requirements that are not explicitly written in the instructions.
- Read each module completely before starting! Really, do it.
- By Odin, by Thor! Use your brain!!!

Regarding the Makefile for C++ projects, the same rules as in C apply (see the Norm chapter about the Makefile).

You will have to implement a lot of classes. This can seem tedious, unless you're able to script your favorite text editor.

You are given a certain amount of freedom to complete the exercises. However, follow the mandatory rules and don't be lazy. You would miss a lot of useful information! Do not hesitate to read about theoretical concepts.

Chapter III

Module-specific rules

It is mandatory to use the standard containers to perform each exercise in this module.

Once a container is used you cannot use it for the rest of the module.

`It is advisable to read the subject in its entirety before doing the exercises.`

`You must use at least one container for each exercise with the exception of exercise 02 which requires the use of two containers.`

You must submit a `Makefile` for each program which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`.

You must use c++, and your Makefile must not relink.

Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.

Chapter IV

AI Instructions

Context

This project is designed to help you discover the fundamental building blocks of your 42 training.

To properly anchor key knowledge and skills, it's essential to adopt a thoughtful approach to using AI tools and support.

True foundational learning requires genuine intellectual effort — through challenge, repetition, and peer-learning exchanges.

For a more complete overview of our stance on AI — as a learning tool, as part of the 42 training, and as an expectation in the job market — please refer to the dedicated FAQ on the intranet.

Main message

- Build strong foundations without shortcuts.
- Really develop tech & power skills.
- Experience real peer-learning, start learning how to learn and solve new problems.
- The learning journey is more important than the result.
- Learn about the risks associated with AI, and develop effective control practices and countermeasures to avoid common pitfalls.

Learner rules:

- You should apply reasoning to your assigned tasks, especially before turning to AI.

- You should not ask for direct answers to the AI.
- You should learn about 42 global approach on AI.

Phase outcomes:

Within this foundational phase, you will get the following outcomes:

- Get proper tech and coding foundations.
- Know why and how AI can be dangerous during this phase.

Comments and example:

- Yes, we know AI exists — and yes, it can solve your projects. But you're here to learn, not to prove that AI has learned. Don't waste your time (or ours) just to demonstrate that AI can solve the given problem.
- Learning at 42 isn't about knowing the answer — it's about developing the ability to find one. AI gives you the answer directly, but that prevents you from building your own reasoning. And reasoning takes time, effort, and involves failure. The path to success is not supposed to be easy.
- Keep in mind that during exams, AI is not available — no internet, no smartphones, etc. You'll quickly realise if you've relied too heavily on AI in your learning process.
- Peer learning exposes you to different ideas and approaches, improving your interpersonal skills and your ability to think divergently. That's far more valuable than just chatting with a bot. So don't be shy — talk, ask questions, and learn together!
- Yes, AI will be part of the curriculum — both as a learning tool and as a topic in itself. You'll even have the chance to build your own AI software. In order to learn more about our crescendo approach you'll go through in the documentation available on the intranet.

✓ Good practice:

> I'm stuck on a new concept. I ask someone nearby how they approached it. We talk for 10 minutes — and suddenly it clicks. I get it.

✗ Bad practice:

> I secretly use AI, copy some code that looks right. During peer evaluation, I can't explain anything. I fail. During the exam — no AI — I'm stuck again. I fail.

Chapter V

Exercise 00: Bitcoin Exchange

	Exercise: 00
Bitcoin Exchange	
Directory: *ex*00/	
Files to Submit: `Makefile, main.cpp, BitcoinExchange.{cpp, hpp}`	
Forbidden: `None`	

You have to create a program which outputs the value of a certain amount of bitcoin on a certain date.

This program must use a database in csv format which will represent bitcoin price over time. This database is provided with this subject.

The program will take as input a second database, storing the different prices/dates to evaluate.

Your program must respect these rules:

- The program name is btc.
- Your program must take a file as an argument.
- Each line in this file must use the following format: "date | value".
- A valid date will always be in the following format: Year-Month-Day.
- A valid value must be either a float or a positive integer, between 0 and 1000.

`You must use at least one container in your code to validate this exercise. You should handle possible errors with an appropriate error message.`

Here is an example of an input.txt file:

```
$> head input.txt
date | value
2011-01-03 | 3
2011-01-03 | 2
2011-01-03 | 1
2011-01-03 | 1.2
2011-01-09 | 1
2012-01-11 | -1
2001-42-42
2012-01-11 | 1
2012-01-11 | 2147483648
$>
```

Your program will use the value in your input file.

Your program should display on the standard output the result of the value multiplied by the exchange rate according to the date indicated in your database.

If the date used in the input does not exist in your DB then you must use the closest date contained in your DB. Be careful to use the lower date and not the upper one.

The following is an example of the program's use.

```
$> ./btc
Error: could not open file.
$> ./btc input.txt
2011-01-03 => 3 = 0.9
2011-01-03 => 2 = 0.6
2011-01-03 => 1 = 0.3
2011-01-03 => 1.2 = 0.36
2011-01-09 => 1 = 0.32
Error: not a positive number.
Error: bad input => 2001-42-42
2012-01-11 => 1 = 7.1
Error: too large a number.
$>
```

Warning: The container(s) you use to validate this exercise will no longer be usable for the rest of this module.

Chapter VI

Exercise 01: Reverse Polish Notation

	Exercise: 01
RPN	
Directory: *ex*01/	
Files to Submit: `Makefile, main.cpp, RPN.{cpp, hpp}`	
Forbidden: `None`	

You must create a program with these constraints:

- The program name is RPN.
- Your program must take an inverted Polish mathematical expression as an argument.
- The numbers used in this operation and passed as arguments will always be less than 10. The calculation itself but also the result do not take into account this rule.
- Your program must process this expression and output the correct result on the standard output.
- If an error occurs during the execution of the program an error message should be displayed on the standard error.
- Your program must be able to handle operations with these tokens: "+ - / *".

`You must use at least one container in your code to validate this exercise.`

You don't need to handle brackets or decimal numbers.

Here is an example of standard usage:

```
$> ./RPN "8 9 * 9 - 9 - 9 - 4 - 1 +"
42
$> ./RPN "7 7 * 7 -"
42
$> ./RPN "1 2 * 2 / 2 * 2 4 - +"
0
$> ./RPN "(1 + 1)"
Error
$>
```

Warning: The container(s) you used in the previous exercise are forbidden here. The container(s) you used to validate this exercise will not be usable for the rest of this module.

Chapter VII

Exercise 02: PmergeMe

	Exercise: 02
PmergeMe	
Directory: *ex*02/	
Files to Submit: `Makefile, main.cpp, PmergeMe.{cpp, hpp}`	
Forbidden: `None`	

You must create a program with these constraints:

- The name of the program is PmergeMe.
- Your program must be able to use a positive integer sequence as an argument.
- Your program must use the merge-insert sort algorithm to sort the positive integer sequence.

To clarify, yes, you need to use the Ford-Johnson algorithm. (source: Art Of Computer Programming, Vol.3. Merge Insertion, Page 184.)

- If an error occurs during program execution, an error message should be displayed on the standard error.

You must use at least two different containers in your code to validate this exercise. Your program must be able to handle at least 3000 different integers.

It is strongly advised to implement your algorithm for each container and thus to avoid using a generic function.

Here are some additional guidelines on the information you should display line by line on the standard output:

- On the first line you must display an explicit text followed by the unsorted positive integer sequence.
- On the second line you must display an explicit text followed by the sorted positive integer sequence.
- On the third line, you must display an explicit message indicating the time taken by your algorithm, specifying the first container used to sort the positive integer sequence.
- On the last line you must display an explicit text indicating the time used by your algorithm by specifying the second container used to sort the positive integer sequence.

The format for the display of the time used to carry out your sorting is free but the precision chosen must allow to clearly see the difference between the two containers used.

Here is an **example** of standard use:

```
$> ./PmergeMe 3 5 9 7 4
Before:  3 5 9 7 4
After:   3 4 5 7 9
Time to process a range of   5 elements with std::[..] :  0.00031 us
Time to process a range of   5 elements with std::[..] :  0.00014 us
$> ./PmergeMe `shuf -i 1-100000 -n 3000 | tr "\n" " "`
Before:  141 79 526 321 [...]
After:   79 141 321 526 [...]
Time to process a range of 3000 elements with std::[..] : 62.14389 us
Time to process a range of 3000 elements with std::[..] : 69.27212 us
$> ./PmergeMe "-1" "2"
Error
$> # For OSX USER:
$> ./PmergeMe `jot -r 3000 1 100000 | tr '\n' ' '`
[...]
$>
```

The indication of the time is deliberately strange in this example. Of course you have to indicate the time used to perform all your operations, both the sorting part and the data management part.

Warning: The container(s) you used in the previous exercises are forbidden here.

The management of errors related to duplicates is left to your discretion.

Chapter VIII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behavior change, a few lines of code to write or rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific timeframe is defined as part of the evaluation.
You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may vary from one evaluation to another for the same project.