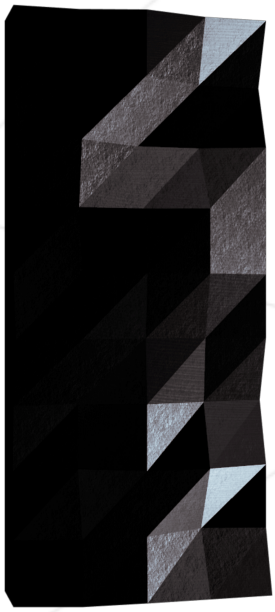

# ft_containers

## C++ containers, easy mode

*Summary: The multiple available containers in C++ all have very different usage. To make sure you understand them, let's re-implement them!*

*Version: 4*

# Contents

# Chapter I

# General rules

- Any function implemented in a header (except in the case of templates), and any unprotected header means 0 to the exercise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names.
- Remember: You are coding in `C++` now, not in `C` anymore. Therefore:
  - The following functions are FORBIDDEN, and their use will be punished by a `0`, no questions asked: `*alloc`, `*printf` and `free`.
  - You are allowed to use everything in the standard library. HOWEVER, it would be smart to try and use the C++-ish versions of the functions you are used to in C, instead of just keeping to what you know, this is a new language after all.
  - Since your goal here is to recode the STL library, you of course cannot use the containers themselves.
- Actually, the use of any explicitly forbidden function or mechanic will be punished by a `0`, no questions asked.
- Also note that unless otherwise stated, the `C++` keywords `"using namespace"` and `"friend"` are forbidden. Their use will be punished by a `-42`, no questions asked.
- Files associated with a class will always be `ClassName.hpp` and `ClassName.cpp`, unless specified otherwise.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description. If something seems ambiguous, you don't understand `C++` enough.
- Since you are allowed to use the `C++` tools you learned about since the beginning, you are not allowed to use any external library. And before you ask, that also means no `C++11` and derivates, nor `Boost`.
- Read each project FULLY before starting it! Really, do it.
- The compiler to use is `clang++`.

- Your code has to be compiled with the following flags : `-Wall -Wextra -Werror -std=c++98`.
- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on, obviously.
- In case you're wondering, no coding style is enforced during in `C++`. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code she or he can't grade.
- Important stuff: You will NOT be graded by a program. However, DO NOT be lazy, you would miss a LOT of what they have to offer!
- It's not a problem to have some extraneous files in what you turn in, you may choose to separate your code in more files than what's asked of you. Feel free, as long as the result is not graded by a program.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

# Chapter II

# Objectives

In this project, you will implement the various container types of the C++ standard template library.
For each container, turn in the appropriately named class files.
The namespace will always be `ft` and your containers will be tested using ft::<container>.
You need to respect the structure of the reference container. If it's missing a part of the orthodox canonical form, do not implement it.
As a reminder, we are coding in C++98, so any new feature of the containers `MUST NOT` be implemented, but every old feature (even deprecated) is expected.

# Chapter III

# Mandatory part

- Implement the following containers and turn in the necessary files `<container>.hpp`
- You must also provide a `main.cpp` which tests everything for your evaluations. (you need to go further than the example!)
- You must produce one binary with only your containers and one with the same testing with STL containers.
- Compare outputs and timing (you can be up to 20 times slower).
- member functions, Non-member and overloads are expected.
- respect the naming, take care to details.
- You must use std::allocator.
- You must justify your inner data structure for each container (using a simple array for a map is not ok).
- If the container has an iterator system, you must implement it.
- iterators_traits, reverse_iterator, enable_if, is_integral, equal/lexicographical compare, std::pair, std::make_pair, must be reimplemented.
- You can use https://www.cplusplus.com/ and https://cppreference.com/ as references.
- You cannot implement more public functions than the ones offered in the standard containers. Everything else must be private or protected. Each public function/-variable must be justified.
- For non-member overloads, the keyword `friend` is allowed. Each use of `friend` must be justified and will be checked during evaluation.

You must turn in the following containers and their associated functions:

- Vector
- Map

- Stack

For your vector implementation, it is not mandatory to code the `vector<bool>` specialization.
Your stack will use your vector class as default underlaying container, it must still be compatible with others containers like the STL one.
STL containers are forbidden.
You are allowed to use the STD library.

# Chapter IV

# Bonus part

If you finished the mandatory part, you can try and turn in the bonuses.

As a bonus one last container:

- Set - But this time a Black - Red tree is mandatory.