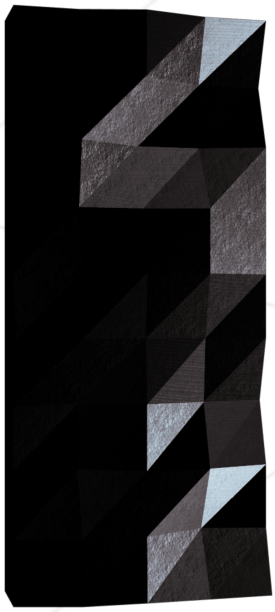

C++ Piscine - Module 07

c++ Templates

Summary: This document contains the subject for the module 07 of 42's C++ piscine.

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.
- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- 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 basically 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. And NO, you are not allowed to use the STL until you actually are supposed to (that is, until module 08). That means no vectors/lists/maps/etc... or anything that requires an include <algorithm> until then.
- 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.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- 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` or anything your awesomely skilled friend told you `C++` can't exist without.

- You may be required to turn in an important number of classes. This can seem tedious, unless you're able to script your favorite text editor.
- Read each exercise 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`.
- 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 now : You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. However, be mindful of the constraints of each exercise, and 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

Exercise 00: A few functions

	Exercise 00
Exercise 00: A few functions	
Turn-in directory : $ex00/$	
Files to turn in : `whatever.cpp and a Makefile`	
Allowed functions : `None`	

Write the following function templates:

- `swap` : Swaps the values of two arguments. Does not return anything.
- `min` : Compares the two arguments and returns the smallest one. If the two arguments are equal, then it returns the second one.
- `max` : Compares the two arguments and returns the biggest one. If the two arguments are equal, then it returns the second one.

These functions can be called with any type of argument, with the condition that the two arguments have the same type and supports all comparison operators. Provide enough code to compile an executable that prooves that everything works as intended.

The following code :

```
int     main( void ) {

    int a = 2;
    int b = 3;

    ::swap( a, b );
    std::cout << "a = " << a << ", b = " << b << std::endl;
    std::cout << "min( a, b ) = " << ::min( a, b ) << std::endl;
    std::cout << "max( a, b ) = " << ::max( a, b ) << std::endl;

    std::string c = "chaine1";
    std::string d = "chaine2";

    ::swap(c, d);
    std::cout << "c = " << c << ", d = " << d << std::endl;
    std::cout << "min( c, d ) = " << ::min( c, d ) << std::endl;
    std::cout << "max( c, d ) = " << ::max( c, d ) << std::endl;

    return 0;
}
```

Should output the following if you did well:

```
a = 3, b = 2
min(a, b) = 2
max(a, b) = 3
c = chaine2, d = chaine1
min(c, d) = chaine1
max(c, d) = chaine2
```

Chapter III

Exercise 01: Iter

	Exercise 01
Exercise 01: Iter	
Turn-in directory : $ex01/$	
Files to turn in : `iter.cpp and a Makefile`	
Allowed functions : `None`	

Write a function template `iter` that take 3 parameters and returns nothing. The first parameter is the address of an array, the second one is the length of the array and the third one is a function that is called on each element of the array.

Wrap your work in an executable that prooves that your function template `iter` works with any type of array and/or with an instantiated function template as a third parameter.

Chapter IV

Exercise 02: Array

	Exercise 02
Exercise 02: Array	
Turn-in directory : *ex*02/	
Files to turn in : `Array.hpp, or Array.tpp, or Array.h, or anything that makes sens, plus anything necessary to build an executable, and a Makefile`	
Allowed functions : `None`	

Write a class template `Array` that contains elements of type `T` and that allows the following behaviors:

- Construction with no parameter: creates an empty array.
- Construction with an `unsigned int n` as a parameter: creates an array of `n` elements, initialized by default. (Tip: try to compile `int * a = new int();`, then display `*a`.)
- Construction by copy and assignment operator. In both cases, modifying one of the two arrays after copy/assignment won't affect anything in the other array.
- You MUST use the operator `new[]` for your allocation. You must not do preventive allocation. Your code must never access non allocated memory.
- Elements are accessible through the `operator[]`.
- When accessing an element with the `operator[]`, if this element is out of the limits, a `std::exception` is thrown.
- A member function `size` that returns the number of elements in the array. This member function takes no parameter and does not modify the current instance in any way.

Wrap your work into an executable that prooves that your class template works as intended.