



# C++ - Module 03

## Inheritance

*Summary: This document contains the subject for Module 03 of the C++ modules.*

# Contents

<b>I</b>	<b>General rules</b>	<b>2</b>
<b>II</b>	<b>Exercise 00: Aaaaand... OPEN!</b>	<b>4</b>
<b>III</b>	<b>Exercise 01: Serena, my love!</b>	<b>5</b>
<b>IV</b>	<b>Exercise 02: Repetitive work</b>	<b>6</b>
<b>V</b>	<b>Exercise 03: Now it's weird!</b>	<b>7</b>

# 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 derivatives, 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 contain 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 explicitly 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: Aaaaand... OPEN!

	Exercise : 00
Aaaaand... OPEN!	
Turn-in directory : <i>ex00/</i>	
Files to turn in : Makefile ClapTrap.cpp ClapTrap.hpp main.cpp	
Forbidden functions : None	

Here you have to make a class! How original!

The class will be called `ClapTrap`, and will have the following private attributes, initialized to the specified values:

- Name (Parameter of constructor)
- Hitpoints (10)
- Energy points (10)
- Attack damage (0)

You will also give it a few public functions to make it more life-like:

- `void attack(std::string const & target)`
- `void takeDamage(unsigned int amount)`
- `void beRepaired(unsigned int amount)`

In all of these functions, you have to display something to describe what happens. For example, the `attack` function may display something along the lines of:


`ClapTrap <name> attack <target>, causing <damage> points of damage!`

The constructor and destructor must also display something, so people can see they have been called.

You will provide a `main` function, with enough tests to demonstrate that your code is functional.

# Chapter III

## Exercise 01: Serena, my love!

	Exercise : 01
Serena, my love!	
Turn-in directory : <i>ex01/</i>	
Files to turn in : Same as previous exercise + <code>ScavTrap.cpp</code> <code>ScavTrap.hpp</code>	
Forbidden functions : None	

Because we can't ever have enough Claptraps, now you will make another one.

The class will be named **ScavTrap**, and will inherit from **ClapTrap**, the constructor, destructor, and attack have to use different outputs. After all, a Claptrap has to have some measure of individuality.

The **ScavTrap** class will have its construction and destruction messages. Also, proper construction/destruction chaining must be present (When you build a **ScavTrap**, you must start by building a **ClapTrap**... Destruction is in reverse order), and the tests have to show it.

ScavTrap will use the attributes of Claptrap (You need to change Claptrap accordingly)! And must initialize them to:


- Name (Parameter of constructor)
- Hitpoints (100)
- Energy points (50)
- attack damage (20)

ScavTrap will also have a new specific function: `void guardGate();` this function will display a message on the standard outputs to announce that ScavTrap have entered in Gate keeper mode.

Extend your `main` function to test everything.

# Chapter IV

## Exercise 02: Repetitive work

	Exercise : 02
Repetitive work	
Turn-in directory : <i>ex02/</i>	
Files to turn in : Same as previous exercise + <code>FragTrap.cpp</code> <code>FragTrap.hpp</code>	
Forbidden functions : None	

Making Claptraps is probably starting to get on your nerves.

Now, you will make a `FragTrap` class that inherits from `ClapTrap`.

- Name (Parameter of constructor)
- Hitpoints (100)
- Energy points (100)
- attack damage (30)


The `FragTrap` class will have its construction and destruction messages. Also, proper construction/destruction chaining must be present (When you build a `FragTrap`, you must start by building a `ClapTrap`... Destruction is in reverse order), and the tests have to show it.

The specific function for `FragTrap` will be `void highFivesGuys(void)` and will display a positive high fives request on the standard output.

Extend your `main` function to test everything.

# Chapter V

## Exercise 03: Now it's weird!

	Exercise : 03
Now it's weird!	
Turn-in directory : <i>ex03/</i>	
Files to turn in : Same as previous exercise + <code>DiamondTrap.cpp</code> <code>DiamondTrap.hpp</code>	
Forbidden functions : None	

Now, you will create a monster by making a Claptrap that's half Fragtrap, half ScavTrap.

It will be named `DiamondTrap`, and it will inherit from both the `FragTrap` AND the `ScavTrap`.

Very risky and horrible act the `DiamondTrap` class will declare a private attribute called `name` like the one inside `ClapTrap`.

Its attributes and functions will be chosen from either of its parent classes:

- `Name` (Parameter of constructor)
- `Claptrap::Name` (Parameter of constructor + `"_clap_name"`)
- `Hitpoints` (`Fragtrap`)
- `Energy points` (`Scavtrap`)
- `Attack damage` (`Fragtrap`)
- `attack` (`Scavtrap`)

It will have the special functions of both.

As usual, your `main` will be extended to test the new class.

Of course, the `Claptrap` part of the `Diamondtrap` will be created once, and only once... Yes, there's a trick.



DiamondTrap will have a new function `void whoAmI();` that will display its name and its clapTrap name.



Compilation flags like `-Wshadow` and `-Wno-shadow`!