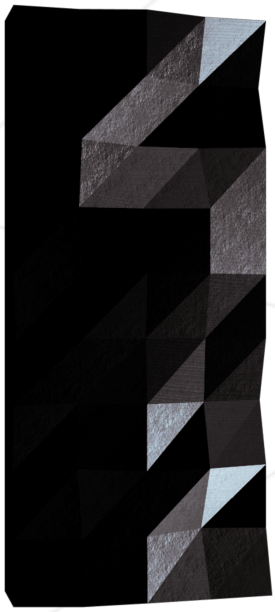

C++ - Module 03

Inheritance

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

Contents

I	**General rules**	**2**
II	**Exercise 00: Aaaaand... OPEN!**	**4**
III	**Exercise 01: Serena, my love!**	**6**
IV	**Exercise 02: Repetitive work**	**8**
V	**Exercise 03: Now it's easier!**	**9**
VI	**Exercise 04: Ultimate assault shoebox**	**10**

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: Aaaaand... OPEN!

	Exercise : 00
Aaaaand... OPEN!	
Turn-in directory : *ex*00/	
Files to turn in : `FragTrap.cpp FragTrap.hpp main.cpp`	
Forbidden functions : `None`	

Here you have to make a class that models the FR4G-TP assault robot/shoebox.

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

- Hit points (100)
- Max hit points (100)
- Energy points (100)
- Max energy points (100)
- Level (1)
- Name (Parameter of constructor)
- Melee attack damage (30)
- Ranged attack damage (20)
- Armor damage reduction (5)

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

- rangedAttack(std::string const & target)
- meleeAttack(std::string const & target)
- takeDamage(unsigned int amount)

- beRepaired(unsigned int amount)

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

```
FR4G-TP <name> attacks <target> at range, causing <damage> points of damage!
```

The constructor and destructor must also display something, so people can see they have been called properly. Bonus points if these messages are funny references (If you don't know what a FR4G-TP is, Google it at least, and use a few well-chosen quotes ...)

A few constraints:

- The number of hit points can never exceed the number of max hit points. Same for the energy points. If, for example, you repair too much HP, then you set them to the max HP number. In the same way, they can't fall below 0.
- When you take damage, you have to take your armor damage reduction into account.

Finish by adding a `vaulthunter_dot_exe(std::string const & target)` function, that will effect a semi-random attack on the target. Make it so each time it is called, it chooses a (preferably) funny attack chosen at random from a pool of at least 5 possible attacks. Whatever you want to use to accomplish this is fine, but as usual, the more elegant your method, the better. This function costs 25 energy points to run. If you don't have enough energy points, it will do nothing else than print something indicating it's out of energy.

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 : $ex01/$	
Files to turn in : `Same as previous exercise + ScavTrap.cpp ScavTrap.hpp`	
Forbidden functions : `None`	

Because we can't ever have enough Claptraps, now you will make another one that serves a slightly different purpose: Manning the door of your soon-to-be evil lair, and challenging people who want to come in.

The class will be named `ScavTrap`, and will have these attributes:

- Hit points (100)
- Max hit points (100)
- Energy points (50)
- Max energy points (50)
- Level (1)
- Name (Parameter of constructor)
- Melee attack damage (20)
- Ranged attack damage (15)
- Armor damage reduction (3)

Add the same functions as in the `FragTrap`, but the constructor, destructor, and attacks have to use different outputs. After all, a Claptrap has to have some measure of individuality.

The one exception will be that the ScavTrap doesn't have a `vaulthunter_dot_exe` function. Instead, it has a `challengeNewcomer` function, which makes the `ScavTrap` choose a challenge at random from a set of various (and hopefully funny) challenges you will have to invent, and print it on the standard output.

Extend your `main` function to test both classes.

Chapter IV

Exercise 02: Repetitive work

	Exercise : 02
Repetitive work	
Turn-in directory : $ex02/$	
Files to turn in : `Same as previous exercise + ClapTrap.cpp ClapTrap.hpp`	
Forbidden functions : `None`	

Making Claptraps is probably starting to get on your nerves, isn't it?

Well, before you can work less, you have to work more.

Now you will make a `ClapTrap` class, that both `FragTrap` and `ScavTrap` will inherit from.

You will put all the common functions in the `ClapTrap` class, but the specific functions must remain where they are. In other words, you must make sure that the `FragTrap` and `ScavTrap` classes contain only what isn't shared between the both of them, and put everything they both share in the parent class.

The `ClapTrap` class will have its own 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.

Chapter V

Exercise 03: Now it's easier!

	Exercise : 03
Now it's easier!	
Turn-in directory : $ex03/$	
Files to turn in : `Same as previous exercise + NinjaTrap.cpp NinjaTrap.hpp`	
Forbidden functions : `None`	

Using everything you have done before, make a `NinjaTrap`, with the following attributes:

- Hit points (60)
- Max hit points (60)
- Energy points (120)
- Max energy points (120)
- Level (1)
- Name (Parameter of constructor)
- Melee attack damage (60)
- Ranged attack damage (5)
- Armor damage reduction (0)

Its special attack will be the `ninjaShoebox` function. There will be multiple functions with the same signature, each taking a reference to a different Claptrap as parameter (including the `NinjaTrap`), and having a different action. Too bad there isn't a way to make it take ANY Claptrap but still react in specific ways... Oh well, you'll see tomorrow. Even I don't know what it does exactly, make it do something funny.

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

Do you notice how easy it is to make a new Claptrap now that you have a parent class for it?

Chapter VI

Exercise 04: Ultimate assault shoebox

	Exercise : 04
Ultimate assault shoebox	
Turn-in directory : $ex04/$	
Files to turn in : `Same as previous exercise + SuperTrap.cpp SuperTrap.hpp`	
Forbidden functions : `None`	

Now, you will combine the best of both worlds by making a Claptrap that's half Fragtrap, half Ninjatrap.

It will be named `SuperTrap`, and it will inherit from both the `FragTrap` AND the `NinjaTrap`.

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

- Hit points (Fragtrap)
- Max hit points (Fragtrap)
- Energy points (Ninjatrap)
- Max energy points (Ninjatrap)
- Level (1)
- Name (Parameter of constructor)
- Melee attack damage (Ninjatrap)
- Ranged attack damage (Fragtrap)
- Armor damage reduction (Fragtrap)
- rangedAttack (Fragtrap)

- meleeAttack (Ninjatrap)

It will have the special attacks of both.

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

Of course, the Claptrap part of the Supertrap will have to be created once, and only once... Yes, there's a trick. Look it up.