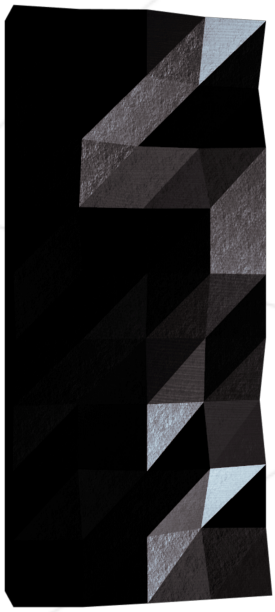

C++ - Module 01

Memory allocation, References, Pointers to members, File streams

Summary: This document contains the subject for Module 01 of the C++ modules.

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: Heap of quadrupeds

	Exercise : 00
Heap of quadrupeds	
Turn-in directory : $ex00/$	
Files to turn in : `Pony.cpp Pony.hpp main.cpp`	
Forbidden functions : `None`	

An easy one, to start with.

Make a `Pony` class, containing whatever you think adequately describes a pony. Then, create two functions, `ponyOnTheHeap` and `ponyOnTheStack`, in which you will allocate a `Pony`, and make it do some stuff.

Of course, in the first one, the `Pony` must be allocated on the heap, on in the second one it must be allocated on the stack.

You will provide a `main` with enough code to prove what you made works as intended.

In both cases, the `Pony` objects must not exist after the function hands off control. (Your main will also have to demonstrate this during the correction!)

Chapter III

Exercise 01: Plumbing problem

	Exercise : 01
Plumbing problem	
Turn-in directory : *ex*01/	
Files to turn in : `ex01.cpp`	
Forbidden functions : `None`	

Again, a simple exercise.

You must turn in the following function, after correcting the memory leak contained in it.

Of course, you must play with the memory allocation/deallocation here. Simply removing the variable, or otherwise fiddling around the problem without actually sorting it out, will be considered a wrong answer...

```
void        memoryLeak()
{
        std::string*        panther = new std::string("String panther");

        std::cout << *panther << std::endl;
}
```

Chapter IV

Exercise 02: Plucking some brains

	Exercise : 02
Plucking some brains	
Turn-in directory : $ex02/$	
Files to turn in : `Zombie.cpp Zombie.hpp ZombieEvent.cpp ZombieEvent.hpp main.cpp`	
Forbidden functions : `None`	

First, make a `Zombie` class. Make it contain a type, and a name (at least), and add an `announce()` member function, that will output something along the lines of:

```
<name (type)> Braiiiiiiinnnssss...
```

Whatever you want, really, as long as you output the name and type of the `Zombie`.

After this, you will create a `ZombieEvent` class. It will have a `setZombieType` function, that will store a type in the object, and a function `Zombie* newZombie(std::string name)` that will create a `Zombie` with the chosen type, name it, and return it.

You will also make a `randomChump` function, that will create a `Zombie` with a random name, and make it `announce` itself. Whatever "random" method you choose, truly random names or a random choice from a pool of names, is fine.

You must turn in a full program, `main` included, with enough to prove that what you made works as required. For example, make your newly created Zombies announce themselves.

Now the actual point of the exercise: Your `Zombies` must be destroyed at appropriate times (so, when they are not needed anymore). They must also be allocated in the appropriate fashion: Some times it's appropriate to have them on the stack, at other times the heap is a better choice. You will have to justify what you did to get a positive grade.

Chapter V

Exercise 03: Moar brainz !

	Exercise : 03
Moar brainz !	
Turn-in directory : $ex03/$	
Files to turn in : `Zombie.cpp Zombie.hpp ZombieHorde.cpp ZombieHorde.hpp main.cpp`	
Forbidden functions : `None`	

Re-using the `Zombie` class you made in the previous exercise, make a `ZombieHorde` class.

This class will have a constructor that takes an integer N. At creation, it must allocate N `Zombie` objects, with random names (Same notion of "random" as before), and store them. It will then have an `announce()` function that calls `announce()` on each of the contained `Zombie` objects.

You must allocate all the `Zombie` objects in a single allocation, and release them when the `ZombieHorde` it destroyed.

As usual, provide a `main` with tests and justify your choices.

Chapter VI

Exercise 04: HI THIS IS BRAIN

	Exercise : 04
HI THIS IS BRAIN	
Turn-in directory : *ex*04/	
Files to turn in : `ex04.cpp`	
Forbidden functions : `None`	

Make a program in which you will create a string containing `"HI THIS IS BRAIN"`, a pointer to it, and a reference to it.

You will then display it using the pointer, and finally display it using the reference.

That's all, no tricks.

Chapter VII

Exercise 05: HI BRAIN THIS IS HUMAN

	Exercise : 05
HI BRAIN THIS IS HUMAN	
Turn-in directory : $ex05/$	
Files to turn in : `Brain.cpp Brain.hpp Human.cpp Human.hpp main.cpp`	
Forbidden functions : `None`	

Create a `Brain` class, with whatever you think befits a brain. It will have an `identify()` function, that returns a string containing the brain's address in memory, in hexadecimal format, prefixed by 0x (For example, `"0x194F87EA"`).

Then, make a `Human` class, that has a constant `Brain` attribute, with the same lifetime. It has an `identify()` function, that just calls the `identify()` function of its `Brain` and returns its result.

Now, make it so this code compiles and displays two identical adresses:

```
int main()
{
        Human        bob;

        std::cout << bob.identify() << std::endl;
        std::cout << bob.getBrain().identify() << std::endl;
}
```

This code must be turned in as your `main`, and whatever you add to the `Human` or `Brain` classes in order to make it work must be justified (with another argument than "Er, yeah, well, I fiddled with it until it worked.").

Chapter VIII

Exercise 06: Unnecessary violence

	Exercise : 06
Unnecessary violence	
Turn-in directory : $ex06/$	
Files to turn in : `Weapon.cpp Weapon.hpp HumanA.cpp HumanA.hpp HumanB.cpp HumanB.hpp main.cpp`	
Forbidden functions : `None`	

Make a `Weapon` class, that has a `type` string, and a `getType` that returns a const reference to this string. Also has a `setType`.

Now, create two classes, `HumanA` and `HumanB`, that both have a `Weapon`, a name, and an `attack()` function that displays something like:

```
NAME attacks with his WEAPON_TYPE
```

Make it so the following code produces attacks with "crude spiked club" THEN "some other type of club", in both test cases:

```
int main()
{
	{
		Weapon		club = Weapon("crude spiked club");

		HumanA bob("Bob", club);
		bob.attack();
		club.setType("some other type of club");
		bob.attack();
	}
	{
		Weapon		club = Weapon("crude spiked club");

		HumanB jim("Jim");
		jim.setWeapon(club);
		jim.attack();
		club.setType("some other type of club");
		jim.attack();
	}
}
```

In which case is it appropriate to store the `Weapon` as a pointer? As a reference? Why? Is it the best choice in light of what's asked? These are the questions you should ask yourself before turning in this exercise.

Chapter IX

Exercise 07: Sed is for losers

	Exercise : 07
Sed is for losers	
Turn-in directory : $ex07/$	
Files to turn in : `Makefile, and whatever else you need`	
Forbidden functions : `None`	

Make a program called `replace` that takes a filename and two strings, let's call them s1 and s2, that are NOT empty.

It will open the file, and write its contents to FILENAME.replace, after replacing every occurence of s1 with s2.

Of course, you will handle errors as best you can, and not use the C file manipulation functions, because that would be cheating, and cheating's bad, m'kay?

You will turn in some test files to show your program works.

Chapter X

Exercise 08: I ain't heard of no fancy switches

	Exercise : 08
I ain't heard of no fancy switches	
Turn-in directory : *ex*08/	
Files to turn in : `Human.hpp Human.cpp main.cpp`	
Forbidden functions : `None`	

`This exercise does not offer any points but is still useful. You can do it, or not.`

Use the following `Human` class:

```
class Human
{
        private:
        void        meleeAttack(std::string const & target);
        void        rangedAttack(std::string const & target);
        void        intimidatingShout(std::string const & target);

        public:
        void        action(std::string const & action_name, std::string const & target);
};
```

Implement all these functions, the first three will simply output something to the standard output so you can see they've been called, and the last one will have to call the appropriate action on the appropriate target. You must use an array of pointers to members to select which function to call: using multiple `if` statements, or `switch` statements, is forbidden.

Chapter XI

Exercise 09: Over logging

	Exercise : 09
Over logging	
Turn-in directory : *ex*09/	
Files to turn in : `Logger.cpp Logger.hpp main.cpp`	
Forbidden functions : `None`	

`This exercise does not offer any points but is still useful. You can do it, or not.`

Make a `Logger` class that must, well, do some logging.

It will have two private functions, `logToConsole` and `logToFile`, that both take a string and will respectively write it to the standard output and append it to a file, which name will be stored in the `Logger` at creation time.

You will also make a private function called `makeLogEntry` that will take a simple message as a string, and return a new string containing the message formatted to look like a legitimate log entry. At the very minimum, add the current date before the message, so we see when it's been logged.

Finally, create a `log(std::string const & dest, std::string const & message)`, that will make a log entry with the message, and pass it to `logToFile` or `logToConsole`, depending on the `dest` parameter. As in the previous exercise, you have to use pointers to members to select which function to call.

Chapter XII

Exercise 10: Cat o' nine tails

	Exercise : 10
Cat o' nine tails	
Turn-in directory : $ex10/$	
Files to turn in : `main.cpp + Whatever you need`	
Forbidden functions : `None`	

`This exercise does not offer any points but is still useful. You can do it, or not.`

Make a `cato9tails` program that does the same thing as the system's `cat` command without options. It can take read from files and/or the standard input. Be thorough in your testing, this is not as simple as it seems.