



C++ - Module 01

Memory allocation, pointers to members,
references, switch statement

Summary:

This document contains the exercises of Module 01 from C++ modules.

Version: 10

Contents

I	Introduction	2
II	General rules	3
III	Exercise 00: BraiiiiiiinnnzzzZ	5
IV	Exercise 01: Moar brainz!	6
V	Exercise 02: HI THIS IS BRAIN	7
VI	Exercise 03: Unnecessary violence	8
VII	Exercise 04: Sed is for losers	10
VIII	Exercise 05: Harl 2.0	11
IX	Exercise 06: Harl filter	13
X	Submission and peer-evaluation	14

Chapter I

Introduction

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes" (source: [Wikipedia](#)).

The goal of these modules is to introduce you to **Object-Oriented Programming**. This will be the starting point of your C++ journey. Many languages are recommended to learn OOP. We decided to choose C++ since it's derived from your old friend C. Because this is a complex language, and in order to keep things simple, your code will comply with the C++98 standard.

We are aware modern C++ is way different in a lot of aspects. So if you want to become a proficient C++ developer, it's up to you to go further after the 42 Common Core!

Chapter II

General rules

Compiling

- Compile your code with `c++` and the flags `-Wall -Wextra -Werror`
- Your code should still compile if you add the flag `-std=c++98`

Formatting and naming conventions

- The exercise directories will be named this way: `ex00`, `ex01`, ... , `exn`
- Name your files, classes, functions, member functions and attributes as required in the guidelines.
- Write class names in **UpperCamelCase** format. Files containing class code will always be named according to the class name. For instance: `ClassName.hpp/ClassName.h`, `ClassName.cpp`, or `ClassName.hpp`. Then, if you have a header file containing the definition of a class "BrickWall" standing for a brick wall, its name will be `BrickWall.hpp`.
- Unless specified otherwise, every output messages must be ended by a new-line character and displayed to the standard output.
- *Goodbye Norminette!* No coding style is enforced in the C++ modules. You can follow your favorite one. But keep in mind that a code your peer-evaluators can't understand is a code they can't grade. Do your best to write a clean and readable code.

Allowed/Forbidden

You are not coding in C anymore. Time to C++! Therefore:

- You are allowed to use almost everything from the standard library. Thus, instead of sticking to what you already know, it would be smart to use as much as possible the C++-ish versions of the C functions you are used to.
- However, you can't use any other external library. It means C++11 (and derived forms) and Boost libraries are forbidden. The following functions are forbidden too: `*printf()`, `*alloc()` and `free()`. If you use them, your grade will be 0 and that's it.

- Note that unless explicitly stated otherwise, the `using namespace <ns_name>` and `friend` keywords are forbidden. Otherwise, your grade will be -42.
- **You are allowed to use the STL in the Module 08 and 09 only.** That means: no **Containers** (vector/list/map/and so forth) and no **Algorithms** (anything that requires to include the `<algorithm>` header) until then. Otherwise, your grade will be -42.

A few design requirements

- Memory leakage occurs in C++ too. When you allocate memory (by using the `new` keyword), you must avoid **memory leaks**.
- From Module 02 to Module 09, your classes must be designed in the **Orthodox Canonical Form, except when explicitly stated otherwise**.
- Any function implementation put in a header file (except for function templates) means 0 to the exercise.
- You should be able to use each of your headers independently from others. Thus, they must include all the dependencies they need. However, you must avoid the problem of double inclusion by adding **include guards**. Otherwise, your grade will be 0.

Read me

- You can add some additional files if you need to (i.e., to split your code). As these assignments are not verified by a program, feel free to do so as long as you turn in the mandatory files.
- Sometimes, the guidelines of an exercise look short but the examples can show requirements that are not explicitly written in the instructions.
- Read each module completely before starting! Really, do it.
- By Odin, by Thor! Use your brain!!!




You will have to implement a lot of classes. This can seem tedious, unless you're able to script your favorite text editor.



You are given a certain amount of freedom to complete the exercises. However, follow the mandatory rules and don't be lazy. You would miss a lot of useful information! Do not hesitate to read about theoretical concepts.

Chapter III

Exercise 00: BraiiiiiiinnnzzzzZ

	Exercise : 00
BraiiiiiiinnnzzzzZ	
Turn-in directory : <i>ex00/</i>	
Files to turn in : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp	
Forbidden functions : None	

First, implement a **Zombie** class. It has a string private attribute **name**.

Add a member function `void announce(void);` to the **Zombie** class. Zombies announce themselves as follows:

```
<name>: BraiiiiiiinnnzzzzZ...
```

Don't print the angle brackets (< and >). For a zombie named **Foo**, the message would be:

```
Foo: BraiiiiiiinnnzzzzZ...
```

Then, implement the two following functions:


- `Zombie* newZombie(std::string name);`
It creates a zombie, name it, and return it so you can use it outside of the function scope.
- `void randomChump(std::string name);`
It creates a zombie, name it, and the zombie announces itself.

Now, what is the actual point of the exercise? You have to determine in what case it's better to allocate the zombies on the stack or heap.

Zombies must be destroyed when you don't need them anymore. The destructor must print a message with the name of the zombie for debugging purposes.

Chapter IV

Exercise 01: Moar brainz!

	Exercise : 01
Moar brainz!	
Turn-in directory : <i>ex01/</i>	
Files to turn in : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp	
Forbidden functions : None	

Time to create a **horde of Zombies!**

Implement the following function in the appropriate file:

```
Zombie*    zombieHorde( int N, std::string name );
```

It must allocate N Zombie objects in a single allocation. Then, it has to initialize the zombies, giving each one of them the name passed as parameter. The function returns a pointer to the first zombie.

Implement your own tests to ensure your `zombieHorde()` function works as expected. Try to call `announce()` for each one of the zombies.

Don't forget to **delete** all the zombies and check for **memory leaks**.