C++ - Module 00

Namespaces, classes, member functions, stdio streams, initialization lists, static, const, and some other basic stuff

Summary:
This document contains the exercises of Module 00 from C++ modules.

Version: 11.1

Contents

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 for learning OOP. We decided to choose C++ since it's derived from your old friend C. Since C++ is a complex language, and in order to keep things simple, your code will follow 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!

You will discover new concepts step-by-step. The exercises will progressively increase in complexity.

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.tpp`. 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 message must end with a newline character and be 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 code your peer evaluators can't understand is code they can't grade. Do your best to write 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 the C++-ish versions of the C functions you are used to as much as possible.
- 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 only in Modules 08 and 09.** That means: no **Containers** (vector/list/map, and so forth) and no **Algorithms** (anything that requires including 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!!!

Regarding the Makefile for C++ projects, the same rules as in C apply (see the Norm chapter about the Makefile).

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

AI Instructions

Context

This project is designed to help you discover the fundamental building blocks of your 42 training.

To properly anchor key knowledge and skills, it's essential to adopt a thoughtful approach to using AI tools and support.

True foundational learning requires genuine intellectual effort — through challenge, repetition, and peer-learning exchanges.

For a more complete overview of our stance on AI — as a learning tool, as part of the 42 training, and as an expectation in the job market — please refer to the dedicated FAQ on the intranet.

Main message

- Build strong foundations without shortcuts.
- Really develop tech & power skills.
- Experience real peer-learning, start learning how to learn and solve new problems.
- The learning journey is more important than the result.
- Learn about the risks associated with AI, and develop effective control practices and countermeasures to avoid common pitfalls.

Learner rules:

- You should apply reasoning to your assigned tasks, especially before turning to AI.

- You should not ask for direct answers to the AI.
- You should learn about 42 global approach on AI.

Phase outcomes:

Within this foundational phase, you will get the following outcomes:

- Get proper tech and coding foundations.
- Know why and how AI can be dangerous during this phase.

Comments and example:

- Yes, we know AI exists — and yes, it can solve your projects. But you're here to learn, not to prove that AI has learned. Don't waste your time (or ours) just to demonstrate that AI can solve the given problem.
- Learning at 42 isn't about knowing the answer — it's about developing the ability to find one. AI gives you the answer directly, but that prevents you from building your own reasoning. And reasoning takes time, effort, and involves failure. The path to success is not supposed to be easy.
- Keep in mind that during exams, AI is not available — no internet, no smartphones, etc. You'll quickly realise if you've relied too heavily on AI in your learning process.
- Peer learning exposes you to different ideas and approaches, improving your interpersonal skills and your ability to think divergently. That's far more valuable than just chatting with a bot. So don't be shy — talk, ask questions, and learn together!
- Yes, AI will be part of the curriculum — both as a learning tool and as a topic in itself. You'll even have the chance to build your own AI software. In order to learn more about our crescendo approach you'll go through in the documentation available on the intranet.

✓ Good practice:

> I'm stuck on a new concept. I ask someone nearby how they approached it. We talk for 10 minutes — and suddenly it clicks. I get it.

✗ Bad practice:

> I secretly use AI, copy some code that looks right. During peer evaluation, I can't explain anything. I fail. During the exam — no AI — I'm stuck again. I fail.

Chapter IV

Exercise 00: Megaphone

	Exercise: 00
Megaphone	
Directory: *ex*00/	
Files to Submit: `Makefile, megaphone.cpp`	
Forbidden: `None`	

Just to make sure that everybody is awake, write a program that produces the following output:

```
$>./megaphone "shhhhh... I think the students are asleep..."
SHHHHH... I THINK THE STUDENTS ARE ASLEEP...
$>./megaphone Damnit " ! " "Sorry students, I thought this thing was off."
DAMNIT ! SORRY STUDENTS, I THOUGHT THIS THING WAS OFF.
$>./megaphone
* LOUD AND UNBEARABLE FEEDBACK NOISE *
$>
```

`Solve the exercises in a C++ manner.`

Chapter V

Exercise 01: My Awesome PhoneBook

	Exercise: 01
My Awesome PhoneBook	
Directory: *ex*01/	
Files to Submit: `Makefile, *.cpp, *.{h, hpp}`	
Forbidden: `None`	

Welcome to the 80s and their unbelievable technology! Write a program that behaves like a crappy awesome phonebook software.

You have to implement two classes:

- **PhoneBook**
  - It has an array of contacts.
  - It can store a maximum of **8 contacts**. If the user tries to add a 9th contact, replace the oldest one by the new one.
  - Please note that dynamic allocation is forbidden.
- **Contact**
  - Stands for a phonebook contact.

In your code, the phonebook must be instantiated as an instance of the **PhoneBook** class. Same thing for the contacts. Each one of them must be instantiated as an instance of the **Contact** class. You're free to design the classes as you like but keep in mind that anything that will always be used inside a class is private, and that anything that can be used outside a class is public.

`Don't forget to watch the intranet videos.`

At program start-up, the phonebook is empty and the user is prompted to enter one of three commands. The program only accepts ADD, SEARCH and EXIT.

- **ADD**: save a new contact
  - If the user enters this command, they are prompted to input the information of the new contact one field at a time. Once all the fields have been completed, add the contact to the phonebook.
  - The contact fields are: first name, last name, nickname, phone number, and darkest secret. A saved contact can't have empty fields.
- **SEARCH**: display a specific contact
  - Display the saved contacts as a list of **4 columns**: index, first name, last name and nickname.
  - Each column must be **10 characters** wide. A pipe character ('|') separates them. The text must be right-aligned. If the text is longer than the column, it must be truncated and the last displayable character must be replaced by a dot ('.').
  - Then, prompt the user again for the index of the entry to display. If the index is out of range or wrong, define a relevant behavior. Otherwise, display the contact information, one field per line.
- **EXIT**
  - The program quits and the contacts are lost forever!
- **Any other input is ignored.**

Once a command has been correctly executed, the program waits for another one. It stops when the user inputs EXIT.

Give a relevant name to your executable.

http://www.cplusplus.com/reference/string/string/ and of course http://www.cplusplus.com/reference/iomanip/

Chapter VI

Exercise 02: The Job Of Your Dreams

	Exercise: 02
The Job Of Your Dreams	
Directory: *ex*02/	
Files to Submit: `Makefile, Account.cpp, Account.hpp, tests.cpp`	
Forbidden: `None`	

`Account.hpp, tests.cpp, and the log file are available for download on the module's intranet page.`

Today is your first day at *GlobalBanksters United*. After successfully passing the recruitment tests (thanks to a few *Microsoft Office* tricks a friend showed you), you joined the dev team. You also know the recruiter was amazed by how quickly you installed *Adobe Reader*. That little extra made all the difference and helped you defeat all your opponents (aka the other applicants): you made it!

Anyway, your manager just gave you some work to do. Your first task is to recreate a lost file. Something went wrong and a source file was deleted by mistake. Unfortunately, your colleagues don't know what `Git` is and use USB keys to share code. At this point, it would make sense to leave this place right now. However, you decide to stay. Challenge accepted!

Your fellow developers give you a bunch of files. Compiling `tests.cpp` reveals that the missing file is `Account.cpp`. Lucky you, the header file `Account.hpp` was saved. There is also a log file. Maybe you could use it in order to understand how the **Account** class was implemented.

You start to recreate the `Account.cpp` file. In only a few minutes, you code a few lines of pure awesome C++. After a couple of failed compilations, your program passes the tests. Its output matches perfectly the one saved in the log file (**except for the timestamps** which will obviously differ since the tests saved in the log file were run before you were hired).

Damn, you're impressive!

`The order in which the destructors are called may differ depending on your compiler/operating system. So your destructors may be called in a reverse order.`

`Completing exercise 02 is not mandatory to pass this module.`

Chapter VII

Submission and peer-evaluation

Submit your assignment to your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behavior change, a few lines of code to write or rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific timeframe is defined as part of the evaluation.
You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may vary from one evaluation to another for the same project.