C++ - Module 06

C++ casts

Summary: This document contains the exercises for Module 06 of the C++ modules.

Version: 8.1

Contents

I	Introduction	2
II	General rules	3
III	Additional Rule	6
IV	AI Instructions	7
V	Exercise 00: Conversion of scalar types	9
VI	Exercise 01: Serialization	12
VII	Exercise 02: Identify real type	13
VIII	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, often referred to as "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, but we chose C++ since it is derived from your old friend, C. As C++ is a complex language, your code will adhere to the C++98 standard to keep things simple.

We acknowledge that modern C++ differs significantly in many aspects. If you want to become a proficient C++ developer, it will be up to you to explore further beyond 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.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

Additional Rule

The following rule applies to the entire module and is mandatory.

For each exercise, type conversion must be handled using a specific type of casting. Your choice will be reviewed during the defense.

Chapter IV

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 V

Exercise 00: Conversion of scalar types

	Exercise00
Conversion of scalar types	
Directory: *ex00/*	
Files to Submit: `Makefile, *.cpp, *.{h, hpp}`	
Authorized: `Any function to convert from a string to an int, a float, or a double. This will help, but won't do the whole job.`	

Write a class ScalarConverter that will contain only one `static` method "convert" that will take as a parameter a string representation of a C++ literal in its most common form and output its value in the following series of scalar types:

- char
- int
- float
- double

As this class doesn't need to store anything at all, it must not be instantiable by users.

Except for char parameters, only the decimal notation will be used.

Examples of char literals: `'c'`, `'a'`, ...

To make things simple, please note that non-displayable characters shouldn't be used as inputs. If a conversion to char is not displayable, print an informative message.

Examples of int literals: `0`, `-42`, `42`...

Examples of float literals: `0.0f`, `-4.2f`, `4.2f`...

You have to handle these pseudo-literals as well (you know, for science): `-inff`, `+inff`, and `nanf`.

Examples of double literals: `0.0`, `-4.2`, `4.2`...
You have to handle these pseudo-literals as well (you know, for fun): **`-inf`**, **`+inf`**, and **`nan`**.

Write a program to test that your class works as expected.

You have to first detect the type of the literal passed as a parameter, convert it from string to its actual type, then convert it **explicitly** to the three other data types. Lastly, display the results as shown below.

If a conversion does not make any sense or overflows, display a message to inform the user that the type conversion is impossible. Include any header you need in order to handle numeric limits and special values.

```
./convert 0
char: Non displayable
int: 0
float: 0.0f
double: 0.0
./convert nan
char: impossible
int: impossible
float: nanf
double: nan
./convert 42.0f
char: '*'
int: 42
float: 42.0f
double: 42.0
```

Chapter VI

Exercise 01: Serialization

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

Implement a class Serializer, which will not be initializable by the user in any way, with the following `static` methods:

`uintptr_t serialize(Data* ptr);`
It takes a pointer and converts it to the unsigned integer type `uintptr_t`.

`Data* deserialize(uintptr_t raw);`
It takes an unsigned integer parameter and converts it to a pointer to `Data`.

Write a program to test that your class works as expected.

You must create a non-empty (meaning it has data members) `Data` structure.

Use `serialize()` on the address of the Data object and pass its return value to `deserialize()`. Then, ensure the return value of `deserialize()` compares equal to the original pointer.

Do not forget to turn in the files of your `Data` structure.

Chapter VII

Exercise 02: Identify real type

	Exercise: 02
Identify real type	
Directory: *ex*02/	
Files to Submit: `Makefile, *.cpp, *.{h, hpp}`	
Forbidden: `std::typeinfo`	

Implement a **Base** class that has a public virtual destructor only. Create three empty classes **A**, **B**, and **C**, that publicly inherit from Base.

`These four classes don't have to be designed in the Orthodox Canonical Form.`

Implement the following functions:

`Base * generate(void);`
It randomly instantiates A, B, or C and returns the instance as a Base pointer. Feel free to use anything you like for the random choice implementation.

`void identify(Base* p);`
It prints the actual type of the object pointed to by p: "A", "B", or "C".

`void identify(Base& p);`
It prints the actual type of the object referenced by p: "A", "B", or "C". Using a pointer inside this function is forbidden.

Including the `typeinfo` header is forbidden.

Write a program to test that everything works as expected.

Chapter VIII

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 folders and 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.