C++ - Module 02

Ad-hoc polymorphism, operator overloading and the Orthodox Canonical class form

Summary: This document contains the exercises of Module 02 from the C++ modules.

Version: 9.1

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, but 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 that modern C++ is quite different in many aspects. So, if you want to become a proficient C++ developer, it is 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.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

New rules

From now on, all your classes must be designed in the **Orthodox Canonical Form**, unless explicitly stated otherwise. They will then implement the four required member functions below:

- Default constructor
- Copy constructor
- Copy assignment operator
- Destructor

Split your class code into two files. The header file (.hpp/.h) contains the class definition, whereas the source file (.cpp) contains the implementation.

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: My First Class in Orthodox Canonical Form

	Exercise: 00
My First Class in Orthodox Canonical Form	
Directory: *ex*00/	
Files to Submit: `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp`	
Forbidden: `None`	

You think you know integers and floating-point numbers. How cute.

Please read this 3 pages article (1, 2, 3) to discover that you don't. Go on, read it.

Until today, every number you used in your code was basically either an integer or a floating-point number, or any of their variants (`short`, `char`, `long`, `double`, and so forth). After reading the article above, it's safe to assume that integers and floating-point numbers have opposite characteristics.

But today, things will change. You are going to discover a new and awesome number type: **fixed-point numbers**! Forever missing from the scalar types of most languages, fixed-point numbers offer a valuable balance between performance, accuracy, range and precision. That explains why fixed-point numbers are particularly applicable to computer graphics, sound processing or scientific programming, just to name a few.

As C++ lacks fixed-point numbers, you're going to add them. This article from Berkeley is a good start. If you have no idea what Berkeley University is, read this section of its Wikipedia page.

Create a class in Orthodox Canonical Form that represents a fixed-point number:

- Private members:
  - An **integer** to store the fixed-point number value.
  - A **static constant integer** to store the number of fractional bits. Its value will always be the integer literal `8`.
- Public members:
  - A default constructor that initializes the fixed-point number value to `0`.
  - A copy constructor.
  - A copy assignment operator overload.
  - A destructor.
  - A member function `int getRawBits( void ) const;` that returns the raw value of the fixed-point value.
  - A member function `void setRawBits( int const raw );` that sets the raw value of the fixed-point number.

Running this code:

```cpp
#include <iostream>

int         main( void ) {

  Fixed a;
  Fixed b( a );
  Fixed c;

  c = b;

  std::cout << a.getRawBits() << std::endl;
  std::cout << b.getRawBits() << std::endl;
  std::cout << c.getRawBits() << std::endl;

  return 0;
}
```

Should output something similar to:

```
$> ./a.out
Default constructor called
Copy constructor called
Copy assignment operator called // <-- This line may be missing depending on your implementation
getRawBits member function called
Default constructor called
Copy assignment operator called
getRawBits member function called
getRawBits member function called
0
getRawBits member function called
0
getRawBits member function called
0
Destructor called
Destructor called
Destructor called
$>
```

Chapter VI

Exercise 01: Towards a more useful fixed-point number class

	Exercise01
Towards a more useful fixed-point number class	
Directory: *ex*01/	
Files to Submit: `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp`	
Authorized: `roundf (from <cmath>)`	

The previous exercise was a good start, but our class is pretty useless. It can only represent the value `0.0`.

Add the following public constructors and public member functions to your class:

- A constructor that takes a **constant integer** as a parameter.
  It converts it to the corresponding fixed-point value. The fractional bits value should be initialized to 8, like in exercise 00.
- A constructor that takes a **constant floating-point number** as a parameter.
  It converts it to the corresponding fixed-point value. The fractional bits value should be initialized to 8, like in exercise 00.
- A member function `float toFloat( void ) const;`
  that converts the fixed-point value to a floating-point value.
- A member function `int toInt( void ) const;`
  that converts the fixed-point value to an integer value.

And add the following function to the **Fixed** class files:

- An overload of the insertion («) operator that inserts a floating-point representation of the fixed-point number into the output stream object passed as a parameter.

Running this code:

```
#include <iostream>

int    main( void ) {

    Fixed       a;
    Fixed const b( 10 );
    Fixed const c( 42.42f );
    Fixed const d( b );

    a = Fixed( 1234.4321f );

    std::cout << "a is " << a << std::endl;
    std::cout << "b is " << b << std::endl;
    std::cout << "c is " << c << std::endl;
    std::cout << "d is " << d << std::endl;

    std::cout << "a is " << a.toInt() << " as integer" << std::endl;
    std::cout << "b is " << b.toInt() << " as integer" << std::endl;
    std::cout << "c is " << c.toInt() << " as integer" << std::endl;
    std::cout << "d is " << d.toInt() << " as integer" << std::endl;

    return 0;
}
```

Should output something similar to:

```
$> ./a.out
Default constructor called
Int constructor called
Float constructor called
Copy constructor called
Copy assignment operator called
Float constructor called
Copy assignment operator called
Destructor called
a is 1234.43
b is 10
c is 42.4219
d is 10
a is 1234 as integer
b is 10 as integer
c is 42 as integer
d is 10 as integer
Destructor called
Destructor called
Destructor called
Destructor called
$>
```

Chapter VII

Exercise 02: Now we're talking

Exercise02
Now we're talking
Directory: *ex*02/
Files to Submit: `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp`
Authorized: `roundf (from <cmath>)`

Add public member functions to your class to overload the following operators:

- The 6 comparison operators: `>`, `<`, `>=`, `<=`, `==`, and `!=`.
- The 4 arithmetic operators: `+`, `-`, `*`, and `/`.
- The 4 increment/decrement (pre-increment and post-increment, pre-decrement and post-decrement) operators, which will increase or decrease the fixed-point value by the smallest representable ϵ, such that $1 + \epsilon > 1$.

Add these four public overloaded member functions to your class:

- A static member function `min` that takes two references to fixed-point numbers as parameters, and returns a reference to the smallest one.
- A static member function `min` that takes two references to **constant** fixed-point numbers as parameters, and returns a reference to the smallest one.
- A static member function `max` that takes two references to fixed-point numbers as parameters, and returns a reference to the greatest one.
- A static member function `max` that takes two references to **constant** fixed-point numbers as parameters, and returns a reference to the greatest one.

It's up to you to test every feature of your class. However, running the code below:

```
#include <iostream>

int     main( void ) {

    Fixed           a;
    Fixed const     b( Fixed( 5.05f ) * Fixed( 2 ) );

    std::cout << a << std::endl;
    std::cout << ++a << std::endl;
    std::cout << a << std::endl;
    std::cout << a++ << std::endl;
    std::cout << a << std::endl;

    std::cout << b << std::endl;

    std::cout << Fixed::max( a, b ) << std::endl;

    return 0;
}
```

Should output something like (for greater readability, the constructor/destructor messages are removed in the example below):

```
$> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
$>
```

If you ever do a division by 0, it is acceptable that the program crashes

Chapter VIII

Exercise 03: BSP

	Exercise03
BSP	
Directory: *ex03/*	
Files to Submit: `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp`	
Authorized: `roundf (from <cmath>)`	

Now that you have a functional **Fixed** class, it would be nice to use it.

Implement a function that indicates whether a point is inside a triangle or not. Very useful, isn't it?

`BSP stands for Binary Space Partitioning. You are welcome. :)`

`You can pass this module without completing exercise 03.`

Let's start by creating the **Point** class in Orthodox Canonical Form that represents a 2D point:

- Private members:
  - A Fixed const attribute `x`.
  - A Fixed const attribute `y`.
  - Anything else useful.
- Public members:
  - A default constructor that initializes `x` and `y` to `0`.
  - A constructor that takes two constant floating-point numbers as parameters. It initializes `x` and `y` with those parameters.
  - A copy constructor.
  - A copy assignment operator overload.
  - A destructor.
  - Anything else useful.

To conclude, implement the following function in the appropriate file:

```
bool bsp( Point const a, Point const b, Point const c, Point const point);
```

- a, b, c: The vertices of our beloved triangle.
- point: The point to check.
- Returns: True if the point is inside the triangle. False otherwise.
  Thus, if the point is a vertex or on an edge, it will return False.

Implement and turn in your own tests to ensure that your class behaves as expected.

Chapter IX

Submission and Peer-Evaluation

Turn in your assignment in 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.