

C++ - Module 02

d-hoc polymorphism, operator overloading and Orthodox Canonical class form

 $Summ \;\; ry:$ This document cont ins the exercises of Module 02 from C++ modules.

Version: 7

Contents

1	Introduction	2
II	General rules	3
III	New rules	5
IV	Exercise 00: My First Class in Orthodox Canonical Form	6
\mathbf{V}	Exercise 01: Towards a more useful fixed-point number class	8
VI	Exercise 02: Now we're talking	10
VII	Exercise 03: BSP	12

Chapter I

Introduction

C++ is gener l-purpose programming language created by Bj rne Stroustrup san extension of the C programming language, or "C with Cl sses" (source: Wikipedi).

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 -W ll -Wextr -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:

 Cl ssN me.hpp/Cl ssN me.h, Cl ssN me.cpp, or Cl ssN me.tpp. Then, if you have a header file containing the definition of a class "BrickWall" standing for a brick wall, its name will be BrickW 11.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.

llowed/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(), * lloc() and free(). If you use them, your grade will be 0 and that's it.

- Note that unless explicitly stated otherwise, the using n mesp ce <ns_n me> and friend keywords are forbidden. Otherwise, your grade will be -42.
- You are allowed to use the STL in Module 08 only. That means: no Containers (vector/list/map/and so forth) and no lgorithms (anything that requires to include the < lgorithm> header) until then. Otherwise, your grade will be -42.

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 08, your classes must be designed in the **Orthodox** Canonical Form, except when explicitly stated otherwise.
- ny 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). s 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 lot of classes. This can seem tedious, unless you're ble to script your favorite text editor.



You re given cert in mount of freedom to complete the exercises. However, follow the m nd tory rules nd don't be 1 zy. You would miss lot of useful inform tion! Do not hesit to re d bout theoretic 1 concepts.

Chapter III

New rules

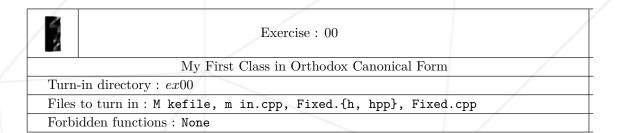
From now on, all your classes must be designed in the **Orthodox Canonical Form**, unless explicitly stated otherwise. Then, they will implement the four required member functions below:

- 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

Exercise 00: My First Class in Orthodox Canonical Form



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, ch r, long, double, and so forth). fter reading the article above, it's safe to assume that integers and floating-point numbers have opposite caracteristics.

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.

s 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:

n integer to store the fixed-point number value.

static constant integer to store the number of fractional bits. Its value will always be the integer literal 8.

• Public members:

```
default constructor that initializes the fixed-point number value to 0.
copy constructor.
copy assignment operator overload.
destructor.
member function int getR wBits( void ) const;
that returns the raw value of the fixed-point value.
member function void setR wBits( int const r w );
that sets the raw value of the fixed-point number.
```

Running this code:

Should output something similar to:

```
$> ./ .out
Def ult constructor c lled
Copy constructor c lled
Copy ssignment oper tor c lled // <-- This line m y be missing depending on your implement tion
getR wBits member function c lled
Def ult constructor c lled
Copy ssignment oper tor c lled
getR wBits member function c lled
getR wBits member function c lled
getR wBits member function c lled
0
getR wBits member function c lled
0
Destructor c lled
```

Chapter V

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

1	Exercise 01			
	Towards a more useful fixed-point number class			
Turn-in directory: $ex01$				
Files to turn in: M kefile, m in.cpp, Fixed.{h, hpp}, Fixed.cpp				
llowed functions: roundf (from <cm th="">)</cm>				

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

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

- constructor that takes a **constant integer** as a parameter. It converts it to the corresponding fixed-point value. The fractional bits value is initialized to 8 like in exercise 00.
- constructor that takes a **constant floating-point number** as a parameter. It converts it to the corresponding fixed-point value. The fractional bits value is initialized to 8 like in exercise 00.
- member function flo t toFlo t(void) const; that converts the fixed-point value to a floating-point value.
- member function int toInt(void) const; that converts the fixed-point value to an integer value.

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

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

Running this code:

Should output something similar to:

```
$> ./ .out
Def ult constructor c lled
Int constructor c lled
Flo t constructor c lled
Copy constructor c lled
Copy ssignment oper tor c lled
Flo t constructor c lled
Copy ssignment oper tor c lled
Destructor c lled
  is 1234.43
b is 10
c is 42.4219
d is 10
  is 1234 s integer
b is 10 s integer
c is 42 s integer
d is 10 s integer
Destructor c lled
Destructor c lled
Destructor c lled
Destructor c lled
```

Chapter VI

Exercise 02: Now we're talking

	Exercise 02			
/	Now we're talking			
Turn-in directory : $ex02$		/		
Files to turn in: M kefile, m in.cpp, Fixed.{h, hpp}, Fixed.cpp				
llowed functions: roundf (from <cm th="">)</cm>				

dd 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, that will increase or decrease the fixed-point value from the smallest representable—such as 1+>1.

dd these four public overloaded member functions to your class:

- static member function min that takes as parameters two references on fixed-point numbers, and returns a reference to the smallest one.
- static member function min that takes as parameters two references to constant fixed-point numbers, and returns a reference to the smallest one.
- \bullet static member function m x that takes as parameters two references on fixed-point numbers, and returns a reference to the greatest one.
- static member function m x that takes as parameters two references to **constant** fixed-point numbers, and returns a reference to the greatest one.

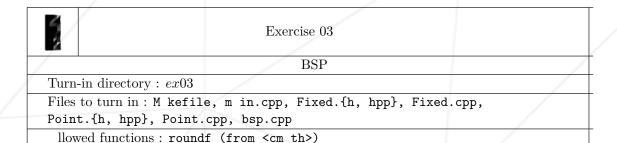
It's up to you to test every feature of your class. However, running the code below:

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

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

Chapter VII

Exercise 03: BSP



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

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



BSP st nds for Bin ry sp ce p rtitioning. You re welcome. :)



You c n p ss this module without doing exercise 03.

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

• Private members:

Fixed const attribute x. Fixed const attribute y. nything else useful.

• Public members:

default constructor that initializes \mathbf{x} and \mathbf{y} to $\mathbf{0}$.

constructor that takes as parameters two constant floating-point numbers. It initializes x and y with those parameters.

copy constructor.

copy assignment operator overload.

destructor.

nything else useful.

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

bool bsp(Point const , 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 edge, it will return False.

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