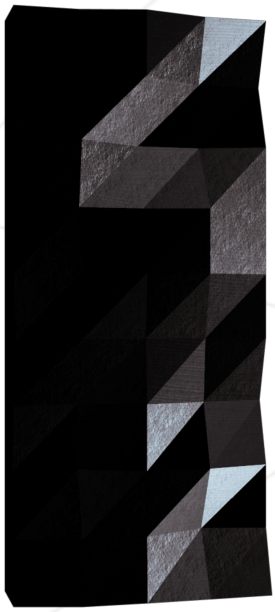

C++ - Module 02

Ad-hoc polymorphism, operators overload and orthodox canonical classes

Summary: This document contains the subject for Module 02 of the C++ modules.

Contents

Chapter I

General rules

- Any function implemented in a header (except in the case of templates), and any unprotected header, means 0 to the exercise.
- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names.
- Remember: You are coding in `C++` now, not in `C` anymore. Therefore:
  - The following functions are FORBIDDEN, and their use will be punished by a `0`, no questions asked: `*alloc`, `*printf` and `free`.
  - You are allowed to use basically everything in the standard library. HOWEVER, it would be smart to try and use the C++-ish versions of the functions you are used to in C, instead of just keeping to what you know, this is a new language after all. And NO, you are not allowed to use the STL until you actually are supposed to (that is, until module 08). That means no vectors/lists/maps/etc... or anything that requires an include <algorithm> until then.
- Actually, the use of any explicitly forbidden function or mechanic will be punished by a 0, no questions asked.
- Also note that unless otherwise stated, the `C++` keywords `"using namespace"` and `"friend"` are forbidden. Their use will be punished by a `-42`, no questions asked.
- Files associated with a class will always be `ClassName.hpp` and `ClassName.cpp`, unless specified otherwise.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description. If something seems ambiguous, you don't understand `C++` enough.
- Since you are allowed to use the `C++` tools you learned about since the beginning, you are not allowed to use any external library. And before you ask, that also means

no `C++11` and derivates, nor `Boost` or anything your awesomely skilled friend told you `C++` can't exist without.

- You may be required to turn in an important number of classes. This can seem tedious, unless you're able to script your favorite text editor.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `clang++`.
- Your code has to be compiled with the following flags : `-Wall -Wextra -Werror`.
- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on, obviously.
- In case you're wondering, no coding style is enforced during in `C++`. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code she or he can't grade.
- Important stuff now : You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. However, be mindful of the constraints of each exercise, and DO NOT be lazy, you would miss a LOT of what they have to offer !
- It's not a problem to have some extraneous files in what you turn in, you may choose to separate your code in more files than what's asked of you. Feel free, as long as the result is not graded by a program.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter II

Bonus rules

- From now on, each class you write MUST be in orthodox canonical form: At least one default constructor, a copy contructor, an assignation operator overload and a destructor. We won't ask again.

Chapter III

Exercise 00: My First Orthodox Class

	Exercise : 00
Exercise 00: My First Orthodox Class	
Turn-in directory : *ex*00/	
Files to turn in : `Makefile, main.cpp, Fixed.hpp and Fixed.cpp`	
Forbidden functions : `None`	

You know integers and you also know floating point numbers. How cute.

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

Until today, any numbers you used in your programs were basically integers or floating point numbers, or any of their variants (`short`, `char`, `long`, `double`, etc). From your previous reading, it's safe to assume that integers and floating point numbers have opposite caracteristics.

But today, this will change. You are going to discover a new and awesome number type: fixed point numbers! Always missing from most languages scalar types, fixed point numbers offer a valuable balance between performance, accuracy, range and precision that explains why these numbers are widely used in graphics, sound or scientific programming to name a few.

As `C++` lacks fixed point numbers, you're going to add them yourself today. I'd recommend this article from `Berkeley` as a start. If it's good for them, it's good for you. If you have no idea what `Berkeley` is, read this section of their wikipedia page.

Write an orthodox class to represent fixed point numbers:

- Private members:
  - An integer to store the fixed point value.
  - A `static constant` integer to store the number of fractional bits. This constant will always be the litteral `8`.
- Public members:
  - A default constructor that initializes the fixed point value to `0`.
  - A destructor.
  - A copy constructor.
  - An assignation operator overload.
  - 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 value.

The code:

```
#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 like:

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

Exercise 01: Towards a more useful fixed point class

	Exercise 01
Exercise 01: Towards a more useful fixed point class	
Turn-in directory : *ex*01/	
Files to turn in : `Makefile, main.cpp, Fixed.hpp and Fixed.cpp`	
Allowed functions : `roundf` `(from <cmath>)`	

Ok, `ex00` was a good start, but our class is still pretty useless, being only able to represent the fixed point 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 and that converts it to the correspondant fixed(8) point value. The fractional bits value is initialized like in `ex00`.
- A constructor that takes a constant floating point as a parameter and that converts it to the correspondant fixed(8) point value. The fractional bits value is initialized like in `ex00`.
- 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.

You will also add the following function overload to your header (declaration) and source (definition) files:

- An overload to the `«` operator that inserts a floating point representation of the fixed point value into the parameter output stream.

The code:
Should output something like:

```cpp
#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;
}
```

```
$> ./a.out
Default constructor called
Int constructor called
Float constructor called
Copy constructor called
Assignation operator called
Float constructor called
Assignation 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 V

Exercise 02: Now we're talking

	Exercise 02
Exercise 02: Now we're talking	
Turn-in directory : *ex*02/	
Files to turn in : `Makefile, main.cpp, Fixed.hpp and Fixed.cpp`	
Allowed functions : `roundf (from <cmath>)`	

We're getting closer. Add the following public member operator overloads to your class:

- Six comparison operators: `>`, `<`, `>=`, `<=`, `==` and `!=`.
- Four arithmetic operators: `+`, `-`, `*`, and `/`.
- The pre-increment, post-increment, pre-decrement and post-decrement operators, that will increment or decrement the fixed point value from the smallest representable ϵ such as $1 + \epsilon > 1$.

Add the following public static member functions overloads to your class:

- The static member function `min` that takes references on two fixed point values and returns a reference to the smallest value, and an overload that takes references on two constant fixed point values and returns a reference to the smallest constant value.
- The static member function `max` that takes references on two fixed point values and returns a reference to the biggest value, and an overload that takes references on two constant fixed point values and returns a reference to the biggest constant value.

It's up to you to test every feature of your class, but the short code:

```
#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 (I deleted the ctors/dtor logs):

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

Chapter VI

Exercise 03: BSP

	Exercise 03
Exercise 03: BSP	
Turn-in directory : *ex*03/	
Files to turn in : `Makefile, main.cpp, Fixed.hpp, Fixed.cpp, Point.hpp, Point.cpp, and bsp.cpp`	
Allowed functions : `roundf (from <cmath>)`	

`This exercise is not required to validate this module.`

Now that you have a fully working fixed point class, it could be cool to use it for something useful. You are going to write a function which indicates whether a point is inside of a triangle or not. Very useful, isn't it?

`BSP stands for Binary space partitioning. You are welcome :)`

Let's start by writing the orthodox class `Point` to represent a 2D point:

- Private members:
  - A `Fixed const x`
  - A `Fixed const y`
  - Anything else you judge useful.
- Public members:
  - A default constructor that initializes `x` and `y` to `0`.
  - A destructor.
  - A copy constructor.
  - A constructor that takes two constant floating points as parameters and that initializes `x` and `y` with those values.
  - An assignation operator overload.
  - Anything else you judge useful.

Now, you should write the function `bsp`:

- The first three parameters are the vertices of our beloved triangle.
- The fourth is the point we are evaluating.
- The return value is `True` if the point is inside the triangle, otherwise, the return value should be `False`. This means that if the point is a vertex or a point that is on the edge, the return value should be `False`.
- Therefore, the prototype of the function is:
  `bool bsp( Point const a, Point const b, Point const c, Point const point);`.

Don't forget to submit your main with some tests to prove that your class works as intended.