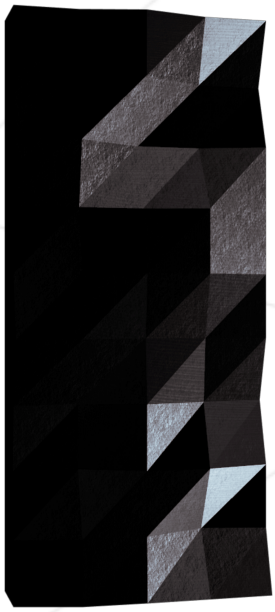

C++ - Módulo 02

Polimorfismo ad-hoc, sobrecargas y clases canónico-ortodoxas en C++

Summary: Este documento contiene el enunciado del módulo 02 de la piscina de C++ de 42.

Contents

Chapter I

Instrucciones generales

Para los módulos de C++ utilizarás y aprenderás exclusivamente C++98. Tu objetivo es aprender las nociones de la programación orientada a objetos. Sabemos que las últimas versiones de C++ son muy diferentes en muchos aspectos, si quieres volverte un experto en C++ deberás aprender C++ moderno más adelante. Este es el principio de tu largo viaje por C++, es cosa tuya ir más allá después del common core.

- Cualquier función implementada en un header (excepto en el caso de templates), y cualquier header desprotegido, significa un 0 en el ejercicio.
- Todos los output deben ir al standard output, y deben terminar con un salto de línea, salvo que se indique lo contrario.
- Se deben seguir los nombres de archivos impuestos al pie de la letra, así como las clases, funciones y métodos.
- Recuerda: estás programando en `C++`, no en `C`. Por lo tanto:
    - Las siguientes funciones están PROHIBIDAS, y su uso será sancionado con un `0`, sin preguntas: `*alloc`, `*printf` y `free`.
    - Tienes permitido utilizar básicamente todo de la librería estándar. SIN EMBARGO, sería inteligente utilizar la versión de C++ de las funciones a las que estás acostumbrado en C, en lugar de simplemente seguir utilizando lo que sabes... En realidad, estás aprendiendo un lenguaje nuevo. Y NO, no tienes permitido utilizar STL hasta que debas hacerlo (es decir, el módulo 08). Esto significa que nada de vectors/ lists/maps/etc. O nada que requiera un "include <algorithm>" hasta entonces.
- De hecho, el uso de cualquier función o mecánica explícitamente prohibida será recompensada con un `0`, sin preguntas.
- Ten en cuenta que, salvo especificado de otro modo, las palabras de `C++ "using namespace"` y `"friend"` están terminantemente prohibidas. Su uso será sancionado con `-42`, sin preguntas.
- Los archivos asociados con una clase se llamarán siempre `ClassName.hpp` y `ClassName.cpp`, salvo especificado de otro modo.
- Entrega en directorios denominados `ex00/`, `ex01/` ... `exn/`.

- Debes leer los ejemplos con cuidado. Pueden contener requisitos no tan obvios en la descripción del ejercicio.
- Dado que tienes permitido utilizar herramientas de `C++` que llevas aprendiendo desde el principio, no tienes permitido utilizar librerías externas. Y antes de que te lo preguntes, esto significa que ningún derivado de `C++11`, ni `Boost` o similares se permite.
- Puede que se requiera entregar un importante número de clases. Esto puede parecer tedioso, salvo que sepas instalar scripts en tu editor de texto favorito.
- Lee cuidadosamente los ejercicios POR COMPLETO antes de empezarlos. En serio, hazlo.
- El compilador a usar es `clang++`.
- Tu código debe compilar con las flags: `-Wall -Werror -Wextra`.
- Cada uno de tus `include` debe poder incluirse independientemente del resto. Los `include` deben contener obviamente los `include` de los que dependan.
- Por si te lo preguntas, no se requiere ningún estilo de código durante `C++`. Puedes utilizar una guía de estilos que te guste, sin limitaciones. Recuerda que si tu evaluador no es capaz de leer tu código, tampoco lo será de evaluarte.
- Algo importante: NO te evaluará un programa, salvo que el subject lo indique explícitamente. Por lo tanto, tienes cierta libertad en cómo hagas los ejercicios. Sin embargo, sé inteligente con los principios de cada ejercicio, y NO seas perezoso, te perderás MUCHO de lo que estos proyectos te pueden ofrecer.
- No es un problema real si tienes archivos adicionales a los que se te solicita, puedes elegir separar el código en más archivos de los que se te piden. Siéntete libre, siempre y cuando el resultado no lo evalúe un programa.
- Aunque el subject de un ejercicio sea corto, merece la pena gastar algo de tiempo para estar absolutamente seguro de que entiendes lo que se espera que entiendas, y que lo has hecho de la mejor forma posible.
- Por Odin, por Thor. Utiliza tu cerebro.

Chapter II

Más reglas

- De ahora en adelante, cada clase que escribas **tiene** que estar en la forma canónico-ortodoxa: al menos un constructor por defecto, un constructor de copia, la sobrecarga de un operador de asignación y un destructor. No lo vamos a volver a pedir.

Chapter III

Ejercicio 00: Mi primera clase canónica

	Exercise : 00
Ejercicio 00: Mi primera clase canónica	
Turn-in directory : *ex*00/	
Files to turn in : `Makefile, main.cpp, Fixed.hpp and Fixed.cpp`	
Forbidden functions : `None`	

Conoces los enteros y los puntos flotantes, qué mono.

Lee los siguientes tres artículos para descubrir que en realidad no los entiendes, adelante:

- http://www.cprogramming.com/tutorial/floating_point/understanding_floating_point.html
- http://www.cprogramming.com/tutorial/floating_point/understanding_floating_point_representation.html
- http://www.cprogramming.com/tutorial/floating_point/understanding_floating_point_printing.html

Hasta hoy, todos los números utilizados en tus programas eran básicamente enteros o números de punto flotante, o cualquiera de sus variantes (`short`, `char`, `long`, `double`, etc). De la lectura de esos artículos, es indiscutible que los enteros y los números de punto flotante tienen características opuestas.

Pero hoy, esto va a cambiar... Vas a descubrir un nuevo e increíble tipo de número: los números de punto fijo. Siempre falta en los lenguajes un tipo escalar. Los números de punto fijo ofrecen un buen equilibrio entre rendimiento, precisión, rango y precisión que explica por qué estos números se utilizan ampliamente en gráficos, sonido o programación científica por nombrar unos pocos.

Como C++ no tiene números de punto fijo, vas a hacerlos por ti mismo. Como lectura puedes tomar este artículp de Berkeley como punto de inicio. Si es bueno para ellos, lo es para ti también. Si no tienes ni idea de qué es Berkeley, puedes leer este otro, un fragmento de la página de Wikipedia.

Escribe una clase canónica para representar los números de punto fijo:

- Miembros privados:
  - Un entero para almacenar el valor de punto entero.
  - Un entero `static constant` para almacenar el número de bits fraccionales. Esta constante será siempre el literal `8`.
- Miembros públicos:
  - Un constructor por defecto que inicialice el valor de punto fijo a `0`.
  - Un destructor.
  - Un constructor de copia.
  - Una sobrecarga del operador de asignación.
  - Una función miembro `int getRawBits( void ) const;` que devuelva el valor bruto del valor de punto fijo.
  - Una función miembro `void setRawBits ( int const raw );` que establece el valor bruto del valor de punto fijo.

El código:

```
#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;
}
```

Debe imprimir algo como:

```
$> ./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

Ejercicio 01: Avanzando hacia una clase de punto fijo más útil

	Exercise 01
Ejercicio 01: Avanzando hacia una clase de punto fijo más útil	
Turn-in directory : $ex01/$	
Files to turn in : `Makefile, main.cpp, Fixed.hpp and Fixed.cpp`	
Allowed functions : `roundf (from <cmath>)`	

Okay, `ex00` ha sido un buen punto de partida, pero nuestra clase todavía es bastante inútil; simplemente podemos representar el valor de punto fijo `0.0`. Añade los siguientes constructores públicos y las siguientes funciones miembro públicas a tu clase:

- Un constructor que acepte un entero constante como parámetro y que lo convierta al valor de punto fijo(8) correspondiente. El valor de los bits fraccionales se inicializa como en el `ex00`.
- Un constructor que acepte un punto flotante constante como parámetro y que lo convierta al valor de punto fijo(8) correspondiente. El valor de los bits fraccionales se inicializa como en el `ex00`.
- Una función miembro `float toFloat( void ) const;` que convierta el valor de punto fijo a un valor de punto flotante.
- Una función miembro `int toInt( void ) const;` que convierta el valor de punto fijo a un valor entero.

Necesitarás añadir también la siguiente sobrecarga de función a tus archivos de header (declaración) y archivo fuente (definición):

- Una sobrecarga del operador « que inserte una representación de punto flotante del valor de punto fijo en el parámetro del stream de salida.

El código:
Debe mostrar algo como:

```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

Ejercicio 02: Ahora estamos hablando

	Exercise 02
Ejercicio 02: Ahora estamos hablando	
Turn-in directory : $ex02/$	
Files to turn in : `Makefile, main.cpp, Fixed.hpp and Fixed.cpp`	
Allowed functions : `roundf (from <cmath>)`	

Nos acercamos poco a poco. Añade las siguientes sobrecargas de operadores como miembros públicos a tu clase:

- Seis operaciones de comparación: `>`, `<`, `>=`, `<=`, `==` y `!=`.
- Cuatro operadores aritméticos: `+`, `-`, `*` y `/`.
- Six comparison operators: `>`, `<`, `>=`, `<=`, `==` and `!=`.
- Four arithmetic operators: `+`, `-`, `*`, and `/`.
- Los operadores de preincremento, postincremento, predecremento y postdecremiento, que aumentarán o disminuirán el valor de punto fijo utilizando el menor número representable ϵ como $1 + \epsilon > 1$.

Añade las siguientes sobrecargas de operadores como miembros públicos y estáticos a tu clase:

- La función miembro estática `min` que acepte referencias a dos valores de punto fijo y devuelva una referencia al valor más pequeño, y una sobrecarga que acepte referencias de dos valores de punto fijo constantes y devuelva una referencia al valor constante más pequeño.
- La función miembro estática `max` que acepte referencias a dos valores de punto fijo y devuelva una referencia al valor más grande, y una sobrecarga que acepte referencias

de dos valores de punto fijo constantes y devuelva una referencia al valor constante más grande.

Es decisión tuya probar cada característica de tu clase, pero aquí tienes algo de código como ejemplo:

```
#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;
}
```

Deberá mostrar algo como esto, pero incluyendo el registro de constructores y destructores:

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

Chapter VI

Ejercicio 03: BSP

	Exercise 03
Ejercicio 03: BSP	
Turn-in directory : $ex03/$	
Files to turn in : `Makefile, main.cpp, Fixed.hpp, Fixed.cpp, Point.hpp, Point.cpp, and bsp.cpp`	
Allowed functions : `roundf (de <cmath>)`	

`Este ejercicio no se requiere para validar este módulo.`

Ahora que has definido por completo tu clase de punto fijo, estaría bien encontrar algún uso práctico. Vas a escribir una función que indica si un punto está dentro o fuera de un triángulo. Bastante útil, ¿no crees?

`BSP significa Binary Space Partitioning, de nada :)`

Vamos a empezar escribiendo una clase ortodoxa `Point` para representar un punto bidimensional:

- Miembros privados:
  - Un `Fixed const x`
  - Un `Fixed const y`
  - Cualquier cosa que consideres útil.

`Fixed es un simple nombre para la clase que definimos en los anteriores ejercicios. Siéntete libre de llamarla como consideres.`

- Miembros públicos:
  - Un constructor por defecto que inicialice `x` e `y` a `0`.
  - Un destructor.
  - Un constructor de copia.
  - Un constructor que acepte dos parámetros de punto flotante constantes y que inicie `x` e `y` con esos valores.
  - Una sobrecarga del operador de asignación.
  - Cualquier cosa que consideres útil.

Ahora deberás escribir la función `bsp`:

- Los tres primeros parámetros son los vértices de nuestro querido triángulo.
- El cuarto es el punto a evaluar.
- El valor de retorno es `true` si el punto está dentro del triángulo, y en caso contrario el valor es `false`. Esto quiere decir que si el punto está en un vértice o el punto está en un borde, el valor de retorno deberá ser `false`.
- Por lo tanto, el prototipo de la función es el siguiente: `bool bsp( Point const a, Point const b, Point const c, Point const point);`.

No olvides entregar un main con algunas pruebas útiles para demostrar que el trabajo entregado funciona como debe.