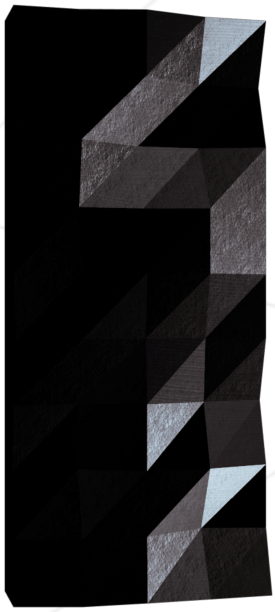

C++ - Módulo 02

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.

Version:

Contents

Chapter I

Reglas generales

Compilando

- Compila tu código con `c++` y los flags `-Wall -Wextra -Werror`
- Tu código deberá compilar si además agregas el flag `-std=c++98`

Formato y convención de nomenclatura

- Los directorios de los ejercicios serán nombrados de la siguiente manera: `ex00, ex01, ... , exn`
- Nombra tus ficheros, clases, funciones, funciones miembros y atributos como lo requieran las reglas.
- Escribe los nombres de las clases con el formato **UpperCamelCase**. Los archivos que contengan código de clase deben ser nombrados de acuerdo con el nombre de la clase. Por ejemplo:
  `ClassName.hpp/ClassName.h`, `ClassName.cpp`, o `ClassName.tpp`. Entonces, si tu tienes un archivo header que contiene la definición de la clase "BrickWall" que representa una pared de ladrillos, su nombre será `BrickWall.hpp`.
- Si no se especifica lo contrario, todos los mensajes de salida deben terminar con un salto de linea y mostrarse en la salida estándar.
- *¡Adiós Norminette!* No se aplica ningún estilo de codificación en los módulos de C++. Puedes seguir a tu favorito. Pero ten en cuenta que un código que tus evaluadores no puede entender es un código que no pueden calificar. Haz tu mejor esfuerzo para escribir un código limpio y legible.

Permitido/Prohibido

No estas codificando en C. ¡llegó la hora de C++! Por lo tanto:

- Se te permite utilizar casi todo de la biblioteca estándar. De este modo, en lugar de seguir aferrado a lo que ya sabes sería inteligente usar, tanto como sea posible, las versiones de C++ de las funciones de C a las que estás acostumbrado.

- Sin embargo, no puedes usar ninguna otra biblioteca externa. Esto significa que ni C++11 ni ninguno de sus derivados, así como las librerías `Boost` están permitidos. Las siguientes funciones también están prohibidas: `*printf()`, `*alloc()` y `free()`. Si los usas, tu calificación sera un 0 definitivo.
- Ten en cuenta que, a menos que se indique explícitamente lo contrario, el `using namespace <ns_name>` y la palabra reservada `friend` están prohibidas. De lo contrario, tu calificación será -42.
- **Podrás usar la librerías STL unicamente en el modulo 08.** Esto significa: No podrás utilizar **Contenedores** (vector/list/map/etcétera) ni **Algorítmos** (o cualquier cosa que requiera incluir el header `<algorithm>`) hasta ese momento. De lo contrario, tu calificación será un -42.

Algunos requisitos de diseños

- Aunque no lo creas se producen fugas de memoria en C++. Cuando reservas memoria (utilizando la palabra reservada `new`), debes evitar **memory leaks**.
- Desde el Modulo 02 al Modulo 08, Tus clases deben estar diseñadas en la **forma canónica ortodoxa, excepto cuando expresamente se indique lo contrario**.
- Cualquier implementación de una función en un archivo header (excepto para funciones de tipo template) significaría un 0 para el ejercicio.
- Deberías poder usar cada uno de tus headers independientemente de los demás. Por lo tanto, deben incluir todas las dependencias que necesitan. No obstante, debes evitar el problema de la doble inclusión agregando **include guards**. De lo contrario, tu calificación sera un 0.

Léeme

- Puedes agregar algunos archivos adicionales si necesitas (p.ej. para organizar tu código). Como estos ejercicios no serán verificados por un programa, siéntete libre de hacerlo siempre y cuando entregues los archivos obligatorios.
- a veces, los requerimientos de un ejercicio parecen breves, pero los ejemplos pueden mostrar requisitos que no están escritos explícitamente en las instrucciones.
- ¡Lee cada modulo completamente antes de empezarlos! De verdad, hazlo.
- ¡Por Odin!, ¡Por Thor! ¡¡¡Usa tu cerebro!!!

Tendrás que implementar muchas clases. Esto puede ser tedioso, a no ser que domines los scripts en tu editor de texto favorito.

Tienes cierta libertad para completar los ejercicios. Sin embargo, Sigues las reglas obligatorias y no seas un sangana/o. ¡Te perderías mucha infomación útil! No dudes en leer sobre conceptos teóricos.

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.