**Capítulo I**

Introducción

C++ es un lenguaje de programación de propósito general creado por Bjarne Stroustrup como una extensión del lenguaje de programación C, o «C con Clases» (fuente: Wikipedia).

El objetivo de estos módulos es introducirte en la Programación Orientada a Objetos.

Este será el punto de partida de tu viaje en C++. Se recomiendan muchos lenguajes

para aprender POO. Decidimos elegir C++ ya que deriva de tu viejo amigo C.

Debido a que este es un lenguaje complejo, y con el fin de mantener las cosas simples, su código se

cumplirá con el estándar C++98.

Somos conscientes de que el C++ moderno es muy diferente en muchos aspectos. Así que si quieres

convertirte en un desarrollador C++ competente, depende de ti ir más allá después de los 42 Common

¡Core!

Descubrirás nuevos conceptos paso a paso. Los ejercicios aumentarán progresivamente

en complejidad.

**Capítulo II**

Reglas generales

Compilación

• Compila tu código con C++ y los indicadores -Wall -Wextra -Werror

• Tu código debería compilar incluso si añades el indicador -std=c++98

Formato y convenciones de nomenclatura

• Los directorios de ejercicios se nombrarán de la siguiente manera: ex00, ex01, ..., exn

• Nombra tus archivos, clases, funciones, funciones miembro y atributos según lo requerido en las directrices.

• Escribe los nombres de clase en formato Mayúsculas y minúsculas. Los archivos que contienen código de clase siempre se nombrarán según el nombre de la clase. Por ejemplo: ClassName.hpp/ClassName.h, ClassName.cpp o ClassName.tpp. Si tienes un archivo de encabezado que contiene la definición de la clase "BrickWall" que representa un muro de ladrillos, su nombre será BrickWall.hpp.

• A menos que se especifique lo contrario, todos los mensajes de salida deben terminar con un carácter de nueva línea y mostrarse en la salida estándar.

• ¡Adiós Norminette! No se impone ningún estilo de codificación en los módulos de C++. Puedes seguir tu estilo favorito. Pero recuerda que el código que tus compañeros evaluadores no pueden entender es código que no pueden calificar. Haz lo posible por escribir un código limpio y legible.

Permitido/Prohibido

Ya no estás programando en C. ¡Hora de C++! Por lo tanto:

• Puedes usar casi todo de la biblioteca estándar. Por lo tanto, en lugar de limitarte a lo que ya sabes, sería inteligente usar, en la medida de lo posible, las versiones similares a C++ de las funciones de C que usas.

• Sin embargo, no puedes usar ninguna otra biblioteca externa. Esto significa que C++11 (y sus derivados) y las bibliotecas Boost están prohibidas. Las siguientes funciones también están prohibidas: \*printf(), \*alloc() y free(). Si las usas, tu calificación será 0 y listo.

Tenga en cuenta que, a menos que se indique explícitamente lo contrario, está prohibido usar las palabras clave <ns\_name> y friend en el espacio de nombres. De lo contrario, su calificación será de -42.

Solo se permite usar la STL en los Módulos 08 y 09. Esto significa:

No se permiten contenedores (vector, lista, mapa, etc.) ni algoritmos (cualquier cosa que requiera incluir la cabecera <algorithm>) hasta entonces. De lo contrario, su calificación será de -42.

Algunos requisitos de diseño

Las fugas de memoria también ocurren en C++. Al asignar memoria (usando la palabra clave new), debe evitarlas.

Del Módulo 02 al Módulo 09, sus clases deben diseñarse en la forma canónica ortodoxa, a menos que se indique explícitamente lo contrario.

Cualquier implementación de función incluida en un archivo de cabecera (excepto las plantillas de función)

significa un 0 en el ejercicio.

Debería poder usar cada una de sus cabeceras independientemente de las demás. Por lo tanto, deben incluir todas las dependencias necesarias. Sin embargo, debes evitar el problema de la doble inclusión añadiendo protecciones de inclusión. De lo contrario, tu calificación será 0.

Léeme

• Puedes añadir archivos adicionales si lo necesitas (por ejemplo, para dividir tu código). Como estas tareas no son verificadas por un programa, puedes hacerlo siempre que entregues los archivos obligatorios.

• A veces, las pautas de un ejercicio parecen cortas, pero los ejemplos pueden mostrar requisitos que no están explícitamente escritos en las instrucciones.

• ¡Lee cada módulo completo antes de empezar! ¡En serio, hazlo!

• ¡Por Odín, por Thor! ¡Usa tu cerebro!

En cuanto al Makefile para proyectos de C++, se aplican las mismas reglas que en C (consulta el capítulo de Normas sobre el Makefile).

Tendrás que implementar muchas clases. Esto puede parecer tedioso, a menos que puedas programar tu editor de texto favorito.

Tienes cierta libertad para completar los ejercicios. Sin embargo, sigue las reglas obligatorias y no seas perezoso. ¡Te perderás mucha información útil! No dudes en leer sobre conceptos teóricos.

**Capítulo III**

Nuevas reglas

A partir de ahora, todas sus clases deben diseñarse en la Forma Canónica Ortodoxa, a menos que se indique explícitamente lo contrario. Implementarán las cuatro funciones miembro requeridas a continuación:

• Constructor predeterminado

• Constructor de copia

• Operador de asignación de copia

• Destructor

Divida el código de su clase en dos archivos. El archivo de encabezado (.hpp/.h) contiene la definición de la clase, mientras que el archivo fuente (.cpp) contiene la implementación.

**Capítulo IV**

Ejercicio 00: Mi primera clase en forma canónica ortodoxa

Ejercicio: 00

Mi primera clase en forma canónica ortodoxa

Directorio de entrega: ex00/

Archivos a entregar: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp

Funciones prohibidas: Ninguna

Crees que conoces los números enteros y de punto flotante. ¡Qué bonito!

Lee este artículo de 3 páginas (1, 2, 3) para descubrir que no. ¡Vamos, léelo!

Hasta hoy, todos los números que usabas en tu código eran básicamente enteros o de punto flotante, o cualquiera de sus variantes (short, char, long, double, etc.).

Después de leer el artículo anterior, podemos asumir que los números enteros y de punto flotante tienen características opuestas.

Pero hoy, las cosas cambiarán. Descubrirás un nuevo e increíble tipo de número: ¡números de punto fijo! Los números de punto fijo, ausentes en los tipos escalares de la mayoría de los lenguajes, ofrecen un valioso equilibrio entre rendimiento, exactitud, alcance y precisión. Esto explica por qué son particularmente aplicables a la gráfica computacional, el procesamiento de sonido o la programación científica, por nombrar solo algunos.

Como C++ carece de números de punto fijo, los añadirás. Este artículo de Berkeley es un buen comienzo. Si no tienes idea de qué es la Universidad de Berkeley, lee esta sección de su página de Wikipedia.

Cree una clase en la forma canónica ortodoxa que represente un número de punto fijo:

• Miembros privados:

◦ Un entero para almacenar el valor del número de punto fijo.

◦ Un entero constante estático para almacenar el número de bits fraccionarios. Su valor siempre será el literal entero 8.

• Miembros públicos:

◦ Un constructor predeterminado que inicializa el valor del número de punto fijo a 0.

◦ Un constructor de copia.

◦ Una sobrecarga del operador de asignación de copia.

◦ Un destructor.

◦ Una función miembro int getRawBits( void ) const;

que devuelve el valor bruto del valor de punto fijo.

◦ Una función miembro void setRawBits( int const raw );

que establece el valor bruto del número de punto fijo.

Ejecutando este 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;

}

Tu output debería ser similar a:

$> ./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

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

$>

**Capítulo V**

Ejercicio 01: Hacia una clase de número de punto fijo más útil

Ejercicio 01

Hacia una clase de número de punto fijo más útil

Directorio de entrega: ex01/

Archivos a entregar: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp

Funciones permitidas: roundf (de <cmath>)

El ejercicio anterior fue un buen comienzo, pero nuestra clase es bastante inútil. Solo puede representar el valor 0.0.

Agregue los siguientes constructores y funciones miembro públicas a su clase:

• Un constructor que toma un entero constante como parámetro.

Lo convierte al valor de punto fijo correspondiente. El valor de los bits fraccionarios debe inicializarse en 8, como en el ejercicio 00.

• Un constructor que toma un número de punto flotante constante como parámetro.

Lo convierte al valor de punto fijo correspondiente. El valor de los bits fraccionarios debe inicializarse en 8, como en el ejercicio 00.

• Una función miembro float toFloat( void ) const;

que convierte el valor de punto fijo en un valor de punto flotante.

• Una función miembro int toInt( void ) const;

que convierte el valor de punto fijo en un valor entero.

Y agregue la siguiente función a los archivos de la clase Fixed:

• Una sobrecarga del operador de inserción («) que inserta una representación en punto flotante del número de punto fijo en el objeto de flujo de salida pasado como parámetro.

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

$>

**Capítulo VI**

Ejercicio 02: Ahora sí que hablamos

Ejercicio 02

Ahora sí que hablamos

Directorio de entrega: ex02/

Archivos a entregar: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp

Funciones permitidas: roundf (de <cmath>)

Añade funciones miembro públicas a tu clase para sobrecargar los siguientes operadores:

• Los 6 operadores de comparación: >, <, >=, <=, == y !=.

• Los 4 operadores aritméticos: +, -, \* y /.

• Los cuatro operadores de incremento/decremento (preincremento y postincremento, predecremento y postdecremento), que incrementan o decrementan el valor de punto fijo en el valor ϵ más pequeño representable, de modo que 1 + ϵ > 1.

Agregue estas cuatro funciones miembro públicas sobrecargadas a su clase:

• Una función miembro estática min que toma dos referencias a números de punto fijo como parámetros y devuelve la referencia al más pequeño.

• Una función miembro estática min que toma dos referencias a números de punto fijo constantes como parámetros y devuelve la referencia al más pequeño.

• Una función miembro estática max que toma dos referencias a números de punto fijo constantes como parámetros y devuelve la referencia al más grande.

• Una función miembro estática max que toma dos referencias a números de punto fijo constantes como parámetros y devuelve la referencia al más grande.

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

**Capítulo VII**

Ejercicio 03: BSP

Ejercicio 03

BSP

Directorio de entrega: ex03/

Archivos a entregar: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp

Funciones permitidas: roundf (de <cmath>)

Ahora que tienes una clase Fixed funcional, sería bueno usarla.

Implementa una función que indique si un punto está dentro de un triángulo o no. Muy útil, ¿verdad?

BSP significa Partición del Espacio Binario (Binary Space Partitioning). De nada. :)

Puedes aprobar este módulo sin completar el ejercicio 03.

Comencemos creando la clase Punto en la Forma Canónica Ortodoxa, que representa un punto 2D:

• Miembros privados:

◦ Un atributo fijo constante x.

◦ Un atributo fijo constante y.

◦ Cualquier otra cosa útil.

• Miembros públicos:

◦ Un constructor predeterminado que inicializa x e y a 0.

◦ Un constructor que toma dos números constantes de punto flotante como parámetros.

Inicializa x e y con esos parámetros.

◦ Un constructor de copia.

◦ Una sobrecarga del operador de asignación de copia.

◦ Un destructor.

◦ Cualquier otra cosa útil.

Para concluir, implemente la siguiente función en el archivo correspondiente:

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

• a, b, c: Los vértices de nuestro triángulo.

• point: El punto a comprobar. • Devuelve: Verdadero si el punto está dentro del triángulo. Falso en caso contrario.

Por lo tanto, si el punto es un vértice o está en una arista, devolverá Falso.

Implementa y entrega tus propias pruebas para asegurar que tu clase se comporte como se espera.