MY FIRST CLASS IN ORTHODOX CANONICAL FORM

Este ejercicio en C++ tiene como objetivo el aprendizaje y la implementación de **clases en la forma canónica ortodoxa** (Orthodox Canonical Form) y el concepto de **números de punto fijo (fixed-point numbers)**. Aquí están los puntos clave y el objetivo:

Objetivo del ejercicio

El ejercicio se centra en la creación de una clase que maneje números de **punto fijo**, un tipo de dato que es útil cuando se necesita un balance entre precisión, rendimiento y rango (usado, por ejemplo, en gráficos por computadora, procesamiento de sonido o programación científica). Los números de punto fijo son una alternativa a los números de punto flotante, ya que pueden ser más eficientes en algunos contextos al no requerir operaciones complejas de punto flotante.

Materias que se desarrollan

1. **Estructura de una clase en C++**:
    a. Implementación de una clase en **Orthodox Canonical Form**.
    b. Uso de constructores (por defecto y copia), destructores y sobrecarga del operador de asignación.
2. **Manejo de miembros privados y públicos**:
    a. Uso de variables privadas (como el valor de número de punto fijo).
    b. Implementación de funciones públicas para acceder y modificar los valores.
3. **Operadores de sobrecarga**:
    a. Se pide la sobrecarga del operador de asignación (operator=), que debe ser correctamente implementado para manejar la asignación de objetos correctamente.
4. **Funciones miembro**:
    a. Implementación de funciones como getRawBits() y setRawBits(), que permiten manipular y obtener el valor "crudo" del número fijo.
5. **Uso de constantes estáticas**:
    a. Se define un valor constante para los bits fraccionarios (en este caso, 8), que es una parte integral de los números de punto fijo.

Estructura de la clase Fixed

La clase Fixed debe tener:

- Un **miembro privado** para almacenar el valor entero del número de punto fijo.
- Una **constante estática** que especifica el número de bits fraccionarios (en este caso, 8).
- **Constructores**:
    - Constructor por defecto: Inicializa el número a 0.
    - Constructor de copia: Crea una nueva instancia a partir de otra existente.
    - Operador de asignación: Permite copiar el valor de un objeto a otro ya existente.
- **Destructor**: Libera los recursos, aunque en este caso no se hace nada especial, ya que no se manejan recursos dinámicos.

- **Funciones miembro**:
  - getRawBits(): Devuelve el valor crudo del número fijo.
  - setRawBits(): Establece el valor crudo del número fijo.

Objetivo de aprendizaje

- Familiarizarse con la **gestión de memoria** y la correcta implementación de las **reglas de los constructores**, **destructores** y **operadores de asignación**.
- Aprender a implementar **tipos de datos personalizados** (en este caso, números de punto fijo).
- Desarrollar habilidades en la implementación de **clases en C++**, siguiendo una estructura que garantiza la correcta manipulación de objetos (mediante la forma canónica ortodoxa).

Resultado esperado

Al ejecutar el código de ejemplo que se proporciona en el enunciado, el programa debe imprimir el valor crudo de los objetos Fixed creados con el constructor por defecto, el constructor de copia y el operador de asignación. Además, se deben mostrar las llamadas a las funciones correspondientes (getRawBits, setRawBits, etc.), lo que demuestra que los métodos de la clase están funcionando correctamente.

Este ejercicio ayuda a consolidar conceptos fundamentales de programación orientada a objetos en C++, especialmente en la creación y manejo de clases personalizadas.

Fixed.hpp

Explicación del código

El código implementa una definición para una clase llamada Fixed, que sirve para manejar números de **punto fijo** (fixed-point numbers) en C++. A continuación, te explico **de forma clara y sencilla** qué hace cada parte del código y cómo funciona:

Encabezado (#ifndef FIXED_HPP ... #endif)

1. **#ifndef FIXED_HPP / #define FIXED_HPP:**
   a. Este mecanismo es una **guarda de inclusión**. Evita que el archivo Fixed.hpp sea incluido más de una vez en el mismo proyecto, lo que podría causar errores de redefinición.
2. **#include <iostream>:**

a. Se incluye esta biblioteca para usar std::cout y std::endl en caso de necesitar imprimir mensajes en pantalla (como en las implementaciones de los métodos en el archivo .cpp).

Miembros privados

private:

```
int                 _rawBits;
static const int    _fractionalBits = 8;
```

1. **_rawBits:**
   a. Es una **variable privada** de tipo int que almacena el valor "crudo" del número en punto fijo.
   b. La idea de un número de punto fijo es que el número real se represente con una parte entera y una parte fraccionaria usando un valor entero.
2. **_fractionalBits:**
   a. Es una **constante estática privada** de tipo int que almacena la cantidad de bits dedicados a la parte fraccionaria del número. En este caso, siempre es 8.
   b. Al ser estática, su valor es compartido por todas las instancias de la clase.

Miembros públicos

Estos son los métodos y constructores que permiten interactuar con la clase Fixed.

Constructores y Destructor

public:

```
Fixed();
Fixed(const Fixed &fix1);
~Fixed();
```

1. **Fixed() (Constructor por defecto):**
   a. Inicializa una nueva instancia de la clase. Aunque no vemos su implementación en este archivo, típicamente establece _rawBits en 0.
2. **Fixed(const Fixed &fix1) (Constructor de copia):**

   a. Crea una nueva instancia de la clase copiando los datos de otro objeto de tipo Fixed. Esto asegura que se pueda crear una copia exacta de un objeto existente.
3. **~Fixed() (Destructor):**
   a. Libera recursos cuando el objeto deja de existir. En este caso, probablemente no realiza ninguna acción, ya que no hay recursos dinámicos asociados.

Operador de asignación

Fixed &operator = (const Fixed &fix2);

- Este método sobrecarga el operador = para que sea posible asignar un objeto de tipo Fixed a otro.
- Copia el valor de _rawBits del objeto fix2 al objeto que recibe la asignación.
- Devuelve una referencia al objeto actual para permitir encadenar asignaciones (por ejemplo: a = b = c).

Funciones miembro

int getRawBits(void) const;
void setRawBits(int const raw);

1. **getRawBits() const:**
   a. Devuelve el valor "crudo" almacenado en _rawBits.
   b. Es una función constante (const), lo que significa que no modifica ningún atributo del objeto.
2. **setRawBits(int const raw):**
   a. Permite establecer manualmente el valor "crudo" de _rawBits.
   b. Se pasa un número entero (raw) como argumento y este se asigna directamente a _rawBits.

¿Qué hace esta clase en términos simples?

La clase Fixed:

1. **Almacena números de punto fijo**:
   a. Usa _rawBits para representar un número con una parte entera y fraccionaria.
   b. La cantidad de bits dedicados a la parte fraccionaria es fija y está definida por _fractionalBits.

2. **Proporciona herramientas para trabajar con estos números**:
    a. getRawBits() y setRawBits() permiten acceder y modificar el valor crudo almacenado.
    b. Soporta creación de objetos por defecto (Fixed()), copia (Fixed(const Fixed&)), y asignación (operator=).
3. **Sigue la forma canónica ortodoxa**:
    a. Implementa un constructor por defecto, un constructor de copia, un destructor, y un operador de asignación, cumpliendo con las buenas prácticas en la gestión de objetos en C++.

Resumen:

La clase define la estructura y comportamiento básico de un número de punto fijo. Aunque el archivo Fixed.hpp solo contiene la **declaración** de la clase, es el primer paso para implementar un sistema de manejo de números de punto fijo en C++. Su objetivo es que los objetos de tipo Fixed puedan ser inicializados, copiados, asignados, y utilizados sin problemas.

¿Qué son los números de punto fijo?

Los números de **punto fijo** (fixed-point numbers) son una representación numérica que divide un número en dos partes:

1. **Parte entera**: Representa el valor principal del número.
2. **Parte fraccionaria**: Representa los decimales, pero está almacenada como parte del número entero.

En lugar de usar flotantes (como float o double), los números de punto fijo usan **valores enteros para almacenar el número completo**, y un número fijo de bits para representar la parte fraccionaria. Esto los hace:

- Más rápidos de procesar (en comparación con números de punto flotante, especialmente en hardware limitado).
- Más deterministas en términos de precisión y rango, ya que la cantidad de bits para cada parte es fija.

Un número de punto fijo se puede imaginar como un número entero desplazado por una cierta cantidad de bits hacia la derecha para dividirlo en su parte entera y fraccionaria.

¿Cómo funcionan los números de punto fijo?

Supongamos que usamos **8 bits para la parte fraccionaria** (como en este caso, donde $2^8 = 256$), y tenemos un valor almacenado como entero en _rawBits:

- Si _rawBits = 256, esto equivale a:

$$Valor\ real = \frac{rawBits}{256} = \frac{256}{256} = 1.0$$

- Si _rawBits = 512, entonces:

$$Valor\ real = \frac{rawBits}{256} = \frac{512}{256} = 2.0$$

- Si _rawBits = 384, entonces:

$$Valor\ real = \frac{rawBits}{256} = \frac{384}{256} = 1.5$$

Al mover el punto decimal de manera fija, podemos representar números fraccionarios usando enteros.

¿Cómo se usan los números de punto fijo en el código?

En el código proporcionado:

1. Almacenamiento en _rawBits

- El atributo _rawBits almacena el valor del número en su forma cruda como un entero.
- Este valor se interpreta como un número de punto fijo dividiendo por $2^{fractionalBits}$ (en este caso, $2^8 = 256$).

2. Uso de _fractionalBits

- _fractionalBits fija cuántos bits representan la parte fraccionaria. En este caso, siempre son 8 bits.
- Esto significa que cada número almacenado en _rawBits será interpretado como:

$$Valor\ real = \frac{rawBits}{256}$$

3. Funciones getRawBits y setRawBits

- **getRawBits()**:
  - Devuelve el valor entero crudo almacenado en _rawBits, sin hacer ninguna conversión.
  - Esto es útil para inspeccionar el número en su forma interna.
- **setRawBits(int const raw)**:
  - Permite asignar un nuevo valor entero crudo a _rawBits.
  - Por ejemplo, si asignamos 384, el número representado será $384/256=1.5$.

4. Constructor y operador de asignación

- Cuando se crea un nuevo objeto de la clase Fixed o se asigna uno a otro, se copian los valores de _rawBits.
- Esto asegura que el número de punto fijo representado en el nuevo objeto sea igual al del original.

Ejemplo Práctico

Supongamos que tienes este código:

```
Fixed a;
a.setRawBits(384);  // Configuramos _rawBits con 384
std::cout << a.getRawBits() << std::endl;  // Imprime "384"
```

- Aquí, el valor real que representa a es: $384/256=1.5$

Ahora supongamos que creas un segundo objeto Fixed:

```
Fixed b(a);  // Constructor de copia
std::cout << b.getRawBits() << std::endl;  // También imprimirá "384"
```

- El objeto b es una copia exacta de a, por lo que también representa 1.5

Ventajas de los números de punto fijo en este contexto

1. **Eficiencia**:
   a. Usan enteros en lugar de flotantes, lo que los hace más rápidos en dispositivos con hardware limitado.
2. **Precisión controlada**:
   a. El número de bits para la parte fraccionaria es fijo, lo que elimina errores inesperados de redondeo o precisión.

3. **Determinismo**:
    a. El comportamiento es predecible y consistente, lo cual es útil para aplicaciones como gráficos por computadora o simulaciones.

Este ejercicio enseña cómo implementar números de punto fijo y los conceptos fundamentales que los respaldan, como el manejo de bits, constructores y funciones miembro.

Fixed.cpp

Explicación del código

Este código implementa los métodos de la clase Fixed que se definieron previamente en Fixed.hpp.

1. Constructor por defecto

```
Fixed::Fixed() : _rawBits(0) {
        std::cout << "Default constructor called" << std::endl;
}
```

- Este **constructor por defecto** inicializa un nuevo objeto de la clase Fixed.
- **Inicialización**:
- Asigna el valor 0 a _rawBits, estableciendo que el número representado es *0.00*
- **Salida**:
    - Imprime "Default constructor called" en consola para indicar que se está creando un objeto usando el constructor por defecto.

2. Constructor de copia

```
Fixed::Fixed(const Fixed &fix1) {
        std::cout << "Copy constructor called" << std::endl;
        this->setRawBits(fix1.getRawBits());
}
```

- Este **constructor de copia** crea un nuevo objeto Fixed copiando los valores de un objeto existente (fix1).
- **Qué hace**:
    - Llama a getRawBits() en el objeto original fix1 para obtener su valor crudo.
    - Usa setRawBits() para copiar ese valor en el nuevo objeto.
- **Salida**:

- Imprime "Copy constructor called" en consola para indicar que se está creando un objeto copiando otro.

3. Destructor

```
Fixed::~Fixed() {
	std::cout << "Destructor called" << std::endl;
}
```

- El **destructor** es llamado automáticamente cuando un objeto Fixed deja de existir (por ejemplo, cuando sale de su ámbito).
- **Qué hace**:
  - Libera los recursos asociados al objeto. En este caso, como no hay recursos dinámicos (como punteros), no realiza acciones adicionales.
- **Salida**:
  - Imprime "Destructor called" en consola para indicar que el objeto está siendo destruido.

4. Operador de asignación (operator=)

```
Fixed &Fixed::operator=(const Fixed &fix2) {
	std::cout << "Copy assignation operator called" << std::endl;
	this->setRawBits(fix2.getRawBits());
	return *this;
}
```

- Este método sobrecarga el operador de asignación (=) para copiar los valores de un objeto Fixed (fix2) a otro ya existente.
- **Qué hace**:
  - Llama a getRawBits() en el objeto fix2 para obtener su valor crudo.
  - Usa setRawBits() para asignar ese valor al objeto actual (*this).
  - Devuelve una referencia al objeto actual para permitir encadenar asignaciones como a = b = c.
- **Salida**:
  - Imprime "Copy assignation operator called" para indicar que se está realizando una asignación entre objetos.

5. Método getRawBits

```
int Fixed::getRawBits(void) const {
        std::cout << "getRawBits member function called" << std::endl;
        return _rawBits;
}
```

- Este método **devuelve el valor crudo** almacenado en _rawBits.
- **Qué hace**:
  - Imprime "getRawBits member function called" para mostrar que se está llamando a este método.
  - Devuelve el valor de _rawBits sin realizar ninguna conversión.
- **Nota**:
  - La palabra clave const garantiza que este método no modifica el objeto.

6. Método setRawBits

```
void Fixed::setRawBits(int const raw) {
        this->_rawBits = raw;
}
```

- Este método **asigna un nuevo valor crudo** al atributo _rawBits.
- **Qué hace**:
  - Toma el valor raw pasado como argumento y lo almacena directamente en _rawBits.
- **Nota**:
  - No realiza validaciones ni conversiones; simplemente almacena el valor.

¿Cómo funciona este código en conjunto?

Ejemplo de uso:

Atendiendo al programas principal main.c:

```
#include "Fixed.hpp"

int main() {
        Fixed a;        // Llama al constructor por defecto
        Fixed b(a);       // Llama al constructor de copia
        Fixed c;        // Llama al constructor por defecto
```

```
    c = b;          // Llama al operador de asignación

    std::cout << a.getRawBits() << std::endl;  // Llama a getRawBits()
    std::cout << b.getRawBits() << std::endl;  // Llama a getRawBits()
    std::cout << c.getRawBits() << std::endl;  // Llama a getRawBits()

    return 0;       // Llama a los destructores
}
```

Paso a paso del flujo:

1. **Creación del objeto a**:
   a. Llama al constructor por defecto. Inicializa _rawBits a 0 y muestra: Default constructor called

2. **Creación del objeto b (copia de a)**:
   a. Llama al constructor de copia. Copia _rawBits de a a b y muestra: Copy constructor called
   getRawBits member function called

3. **Creación del objeto c**:
   a. Llama al constructor por defecto. Inicializa _rawBits a 0 y muestra: Default constructor called

4. **Asignación de b a c**:
   a. Llama al operador de asignación. Copia _rawBits de b a c y muestra: Copy assignation operator called
   getRawBits member function called

5. **Llamadas a getRawBits()**:
   a. Imprime los valores crudos (0 en todos los casos) y muestra: getRawBits member function called
   0
   getRawBits member function called
   0
   getRawBits member function called
   0

6. **Destrucción de los objetos**:

a. Llama al destructor para c, b, y luego a, mostrando: Destructor called
   Destructor called
   Destructor called

Resumen

Este código implementa el comportamiento básico de la clase Fixed:

- **Creación de objetos** con constructores (por defecto y copia).
- **Asignación de valores** con el operador =.
- **Gestión de recursos** con el destructor.
- **Acceso y modificación** del valor crudo _rawBits mediante getRawBits() y setRawBits(). El código sigue la **forma canónica ortodoxa** y muestra mensajes para visualizar el flujo de ejecución.