TOWARDS A MORE USEFUL FIXED-POINT NUMBER CLASS

Descripción del Ejercicio

El ejercicio amplía la funcionalidad de la clase Fixed introducida en el ejercicio anterior, con el objetivo de que sea más útil y capaz de representar números distintos de 0.0. Esto se logra añadiendo constructores para inicializar objetos con valores enteros y de punto flotante, así como métodos para convertir valores de punto fijo a representaciones flotantes e enteras.

Además, se implementa la sobrecarga del operador de inserción (<<) para permitir imprimir directamente los valores representados por objetos de la clase Fixed.

bjetivo del Ejercicio

El objetivo principal es:

1. **Ampliar el entendimiento de los números de punto fijo**:
    a. Aprender cómo convertir entre representaciones de números enteros, flotantes y de punto fijo.
    b. Incorporar operaciones aritméticas básicas que respeten la precisión y las características de los números de punto fijo.
2. **Explorar conceptos avanzados de C++ 98**:
    a. **Constructores**: Implementar constructores adicionales para distintas inicializaciones.
    b. **Conversión de tipos**: Crear funciones que conviertan un número de punto fijo a enteros y flotantes.
    c. **Sobrecarga de operadores**: Implementar la sobrecarga del operador << para personalizar la salida a std::ostream.
3. **Introducir bibliotecas estándar**:
    a. Uso de la función roundf de <cmath> para manejar la conversión precisa de flotantes a enteros.

Aspectos Clave de la Implementación

1. **Constructores adicionales**:
    a. **Constructor con enteros**: Convierte un entero a su representación de punto fijo multiplicándolo por $2^{fractionalBits}$.
    b. **Constructor con flotantes**: Convierte un flotante a su representación de punto fijo multiplicándolo por $2^{fractionalBits}$ y usando roundf para redondear el resultado.
2. **Métodos de conversión**:
    a. **toFloat()**: Convierte el valor almacenado en _rawBits a un flotante dividiéndolo entre $2^{fractionalBits}$.
    b. **toInt()**: Convierte _rawBits a un entero dividiendo entre $2^{fractionalBits}$ y truncando los decimales.
3. **Sobrecarga del operador <<**:

a. Permite que un objeto de la clase Fixed se pueda imprimir directamente a través de std::cout, mostrando su representación flotante mediante toFloat().

Funcionamiento del Código de Ejemplo

Código proporcionado:

```
#include <iostream>
int main( void ) {
  Fixed a;                  // Llama al constructor por defecto
  Fixed const b(10);        // Llama al constructor con entero
  Fixed const c(42.42f);    // Llama al constructor con flotante
  Fixed const d(b);         // Llama al constructor de copia
  a = Fixed(1234.4321f);    // Llama al constructor con flotante y al operador de asignación

  std::cout << "a is " << a << std::endl;  // Usa la sobrecarga de `<<`
  std::cout << "b is " << b << std::endl;  // Usa la sobrecarga de `<<`
  std::cout << "c is " << c << std::endl;  // Usa la sobrecarga de `<<`
  std::cout << "d is " << d << std::endl;  // Usa la sobrecarga de `<<`

  std::cout << "a is " << a.toInt() << " as integer" << std::endl;  // Llama a `toInt()`
  std::cout << "b is " << b.toInt() << " as integer" << std::endl;  // Llama a `toInt()`
  std::cout << "c is " << c.toInt() << " as integer" << std::endl;  // Llama a `toInt()`
  std::cout << "d is " << d.toInt() << " as integer" << std::endl;  // Llama a `toInt()`

  return 0;
}
```

Flujo de Ejecución

1. **Creación de objetos**:
   a. a: Usa el constructor por defecto, inicializando _rawBits a 0.
   b. b: Usa el constructor con entero, convirtiendo 10 a su representación de punto fijo (_rawBits = 10 * 256 = 2560).
   c. c: Usa el constructor con flotante, convirtiendo 42.42 a punto fijo (_rawBits ≈ 42.42 * 256 = 10859).
   d. d: Usa el constructor de copia para copiar el valor de b.
2. **Asignación**:
   a. a = Fixed(1234.4321f):
      i. Crea un objeto temporal con el valor flotante 1234.4321.
      ii. Convierte este valor a punto fijo (_rawBits ≈ 1234.4321 * 256 = 316960).
      iii. Asigna este valor a a.
3. **Salida a consola**:

   a. Usa la sobrecarga de << para imprimir la representación flotante de cada objeto:
      i. a: $\frac{316960}{256} = 1234.43.$
      ii. b: $\frac{2560}{256} = 10.0.$
      iii. c: $\frac{10859}{256} = 42.4219.$
      iv. d: $\frac{2560}{256} = 10.0.$
4. **Conversión a enteros**:
   a. Llama a toInt() para convertir _rawBits a enteros:
      i. a: $\frac{316960}{256} = 1234\ (truncado).$
      ii. b: $\frac{2560}{256} = 10.$
      iii. c: $\frac{10859}{256} \approx 42\ (truncado).$
      iv. d: $\frac{2560}{256} = 10.$

Aprendizajes Clave

1. **Conversión entre tipos numéricos**:
   a. Cómo convertir enteros y flotantes a números de punto fijo, y viceversa.
2. **Uso de constructores**:
   a. Crear objetos desde distintos tipos de datos y copiar sus valores.
3. **Sobrecarga de operadores**:
   a. Implementar operadores personalizados para mejorar la legibilidad del código (como <<).
4. **Manejo de precisión**:
   a. Uso de roundf para evitar errores de redondeo al convertir flotantes a números de punto fijo.

Este ejercicio refuerza las habilidades para trabajar con conversiones, precisión numérica y sobrecarga de operadores, pilares del desarrollo en C++ 98.

Fixed.hpp - Fixed.cpp - main.cpp

Este código implementa una clase Fixed que representa un número de punto fijo utilizando una representación interna de un entero con una cantidad fija de bits fraccionales. También incluye funcionalidades para inicializar, convertir y manipular estos números, además de imprimirlos de forma legible en la consola.

¿Qué hace el código?

1. **Define una clase Fixed con capacidades avanzadas**:
   a. Representa números de punto fijo con 8 bits fraccionales.
   b. Permite inicializar objetos a partir de enteros y flotantes.

c. Convierte internamente entre representaciones de enteros, flotantes y de punto fijo.
d. Sobrecarga operadores, como el de asignación (=) y el de inserción en el flujo (<<), para facilitar la manipulación y la impresión de objetos.

2. **Incluye un flujo de ejecución en main.cpp**:
   a. Crea varios objetos de la clase Fixed.
   b. Inicializa valores con constructores específicos (por defecto, enteros y flotantes).
   c. Realiza asignaciones y copias.
   d. Convierte y muestra los valores de punto fijo como flotantes o enteros.
   e. Usa la sobrecarga de << para imprimir directamente los valores.

Detalles de cómo lo hace

1. Declaración de la clase (Fixed.hpp)

- **Miembros privados**:
  - value: Entero que almacena el valor bruto del número de punto fijo.
  - bits: Número de bits dedicados a la parte fraccional (siempre 8).
- **Constructores**:
  - Por defecto: Inicializa el valor a 0.
  - Parametrizados:
    - Aceptan enteros o flotantes y los convierten a la representación de punto fijo.
  - Copia: Crea un nuevo objeto copiando el valor de otro.
- **Métodos de conversión**:
  - toFloat: Convierte el número de punto fijo a flotante dividiendo entre 2^{bits}.
  - toInt: Convierte el número de punto fijo a un entero desplazando los bits fraccionales.
- **Sobrecarga de operadores**:
  - =: Copia el valor de un objeto Fixed a otro.
  - <<: Imprime la representación flotante de un número de punto fijo.

2. Implementación de la clase (Fixed.cpp)

- **Constructores**:
  - **Por defecto**:

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

Inicializa value a 0 y muestra un mensaje.

- **Con enteros**:

```
Fixed::Fixed(const int inInt) : value(inInt * (1 << Fixed::bits)) {
  std::cout << "Int constructor called" << std::endl;
}
```

Convierte el entero a punto fijo multiplicándolo por 2^{bits} (desplaza los bits hacia la izquierda).

- **Con flotantes**:

```
Fixed::Fixed(const float inFloat) : value(roundf(inFloat * (1 << Fixed::bits))) {
  std::cout << "Float constructor called" << std::endl;
}
```

Convierte el flotante a punto fijo multiplicándolo por 2^{bits} y redondeando con roundf.

- **Copia**:

```
Fixed::Fixed(const Fixed &to_copy) {
  std::cout << "Copy constructor called" << std::endl;
  *this = to_copy;
}
```

Copia el valor de otro objeto Fixed utilizando la sobrecarga del operador de asignación.

- **Sobrecarga de =**:

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

Copia el valor bruto (value) de otro objeto y devuelve el objeto actual.

- **Conversión a flotante**:

```
float Fixed::toFloat(void) const {
  return (float)this->value / (float)(1 << Fixed::bits);
}
```

Divide value entre 2^{bits} para obtener la representación flotante.

- **Conversión a entero**:

```
int Fixed::toInt(void) const {
  return this->value >> Fixed::bits;
}
```

Desplaza los bits hacia la derecha, eliminando la parte fraccional.

- **Sobrecarga de <<**:

```
std::ostream &operator<<(std::ostream &stream, const Fixed &nbr) {
  stream << nbr.toFloat();
  return stream;
}
```

Imprime el número de punto fijo como flotante.

3. Flujo de ejecución (main.cpp)

- **Creación y asignación de objetos**:

```
Fixed a;                    // Constructor por defecto
Fixed const b(10);          // Constructor con entero
Fixed const c(42.42f);      // Constructor con flotante
Fixed const d(b);           // Constructor de copia
a = Fixed(1234.4321f);      // Asignación con constructor flotante
```

- **Salida con <<**:

```
std::cout << "a is " << a << std::endl;  // Imprime el valor flotante de `a`
std::cout << "b is " << b << std::endl;  // Imprime el valor flotante de `b`
std::cout << "c is " << c << std::endl;  // Imprime el valor flotante de `c`
```

std::cout << "d is " << d << std::endl; // Imprime el valor flotante de `d`

- **Conversión a enteros**:

std::cout << "a is " << a.toInt() << " as integer" << std::endl;

Convierte y muestra los valores de punto fijo como enteros.

Resumen

Este código implementa una clase Fixed que:

1. **Representa números de punto fijo** utilizando una base de 8 bits fraccionales.
2. **Proporciona métodos para inicializar, copiar y convertir** números de punto fijo a flotantes e enteros.
3. **Sobrecarga operadores** para facilitar la manipulación y salida.

Es un ejercicio diseñado para reforzar conceptos clave de C++ como constructores, sobrecarga de operadores, conversión de tipos y manipulación numérica avanzada.