Emilio Gomez Breschi

19300100

4D1

Sonia Erika Ibáñez de la
Torre Desarrollo de Software
25/Noviembre/2021

Programación Orientada a Objetos

Sobrecarga del Operador !=

¿Qué tipo de operador es?

Es un operador de tipo relacional, también denominados operadores binarios lógicos y de comparación. También es un operador de igualdad y sirve para verificar la desigualdad entre valores aritméticos o punteros. Este operador puede comparar ciertos tipos de punteros, mientras que el resto de los operadores relacionales no pueden utilizarse con ellos.

(Rodríguez, 2019)

Requerimientos de uso:

- Que ambos operandos sean del mismo tipo
- Que devuelvan un bool
- Que se cumplan las propiedades conmutativa, reflexiva, simétrica y transitiva.

(Rodríguez, 2019)

Sintaxis

```
class nombreDeLaClase {
Tipo de dato:
bool operator != (/* parámetros */) {
        // Código, asegurar de retornan true o false
    }
};
```

(Rodríguez, 2019)

Sobrecarga del Operador >

¿Qué tipo de operador es?

Es un operador de tipo relacional, también denominados operadores binarios lógicos y de comparación, también es un operador binario. (Rodríguez, 2017)

Aunque el programador tiene cierta libertad en el modo de sobrecargar los operadores, conviene mantener una cierta coherencia en su comportamiento, de forma que sus propiedades formales se mantengan también en la versión sobrecargada. En el caso de los operadores relacionales estas propiedades podrían ser sintetizarlas en los puntos siguientes:

- Que ambos operandos sean del mismo tipo
- Que devuelvan un bool
- Que se cumplan las propiedades conmutativa, reflexiva, simétrica y transitiva.

(Rodríguez, 2017)

Sintaxis

```
class nombreDeLaClase {
Tipo de dato:
    bool operator> (/* parámetros */) {
            // Código, asegurar de retornan true o false
    }
};
```

(Rodríguez, 2017)

Ejemplo completo de estos dos últimos operadores

```cpp
#include <cmath>
using namespace std;

template <typename T>
class linea {
public:
    struct punto
    {
        T x, y;
    };
    punto punto1, punto2;
    float longitud;
    linea();
    linea(T, T, T, T);
    bool operator >(const linea&);
    bool operator !=(const linea&);

};

template <typename T> linea<T>::linea(T x1, T y1, T x2, T y2) {
    this->punto1.x = x1;
    this->punto1.y = y1;
    this->punto2.x = x2;
    this->punto2.y = y2;
    this->longitud = sqrt(pow((punto2.x - punto1.x), 2) + pow((punto2.y - punto1.y), 2));
}
template <typename T> linea<T>::linea() {
    this->punto1.x = 0;
    this->punto1.y = 0;
    this->punto2.x = 0;
    this->punto2.y = 0;
    this->longitud = 0.0;
```

```
    this->longitud = 0.0;
}
template <typename T> bool linea<T>::operator>(const linea& p) {
    if (this->longitud > p.longitud)
    {
        return true;
    }
    else
        return false;
}
template <typename T> bool linea<T>::operator!=(const linea& p) {
    if (this->punto1.x != p.punto1.x && this->punto2.y != p.punto2.y)
    {
        return true;
    }
    else
        return false;
}
int main()
{
    linea<int> x(6, 7,9,10);
    linea<int> y(5, 5,8,9);
    cout << "Longitud linea x " << x.longitud << endl;
    cout << "Longitud linea y " << y.longitud << endl;
    if (x > y) {
        cout << "Linea x es mayor que la linea y" << endl;
    }
    else
        cout << "Linea x es menor que la linea y" << endl;
    if (x != y) {
        cout << "Las lineas si son diferentes" << endl;
    }
    else
        cout << "Las lineas no son diferentes" << endl;
}
```

Sobrecarga del operador - -

¿Qué tipo de operador es?

El operador decremento -- actúan siempre sobre un solo operando, normalmente una variable. Por tanto, son operadores monarios, y sirven para decrementar en una unidad el valor de dicho operando.

(Pes, s.f.)

Sintaxis

-- unary-expression

Ejemplo Completo

```
#include <iostream>
using namespace std;
class Entero {
 public: int x;
friend Entero& operator--(Entero&);
};
Entero& operator-- (Entero& e) {
```

```
 e.x = e.x / 2;
 return e;
}
void main () {
 Entero e1, e2, e3;
 e1.x = 5;
e3 = --e2 = --e1;
 cout << " e1 = " << e1.x << "; e2 = " << e2.x
}
```
(jc, 2019)

Ventajas de utilizar sobrecarga de operadores

El objetivo último de la sobrecarga de operadores es simplificar al máximo el código a escribir, a cambio de complicar algo la definición de las clases. Una clase que disponga de operadores sobrecargados es una clase más compleja de definir, pero más sencilla e intuitiva de utilizar.

Las ventajas de la sobrecarga de operadores terminan cuando se utiliza de modo que añade complejidad o confusión a los programas. Por ejemplo, aunque esté permitido por el lenguaje, no se deberá nunca utilizar el operador (-) para multiplicar matrices o el (+) para imprimir vectores.

(Belmonte, s.f.)

Bibliografía

Rodríguez, G. (2017, septiembre 23). Sobrecarga de Operadores Relacionales. Retraído noviembre 10, 2021, de https://www.zator.com/Cpp/E4_9_18b1.htm.

Rodríguez, G. (2019, septiembre 1). Curso de C++. Operadores Relacionales. Retraído noviembre 10, 2021, de https://www.zator.com/Cpp/E4_9_12.htm.

Belmonte, J. C. (s.f.). *Tic*++. (J. F. Valdivia, Productor) Recuperado el 25 de Noviembre de 2021, de https://ccia.ugr.es/~jfv/ed1/c++/cdrom3/TIC-CD/web/tema15/teoria_2.htm#:~:text=El%20objetivo%20%C3%BAltimo%20de%20la,sencilla%20e%20intuitiva%20de%20utilizar.

jc, M. (13 de Julio de 2019). *EcuRed*, 11. (J. jc, Editor) Recuperado el 25 de Noviembre de 2021, de https://www.ecured.cu/Sobrecarga_de_operadores_unarios_(Programaci%C3%B3n)

Pes, C. (s.f.). *carlospes.com*. Recuperado el 25 de Noviembre de 2021, de http://www.carlospes.com/curso_de_lenguaje_c/01_08_06_operadores_incremento_y_decremento.php