UNIVERSIDAD DE LOS LLANOS

08 SOBRECARGA DE OPERADORES

Néstor Suat-Rojas. Ing. Msc (c)
nestor.suat@unillanos.edu.co

Escuela de Ingeniería
Facultad de Ciencias Básicas e Ingeniería

SOBRECARGA (1)

C++ le permite especificar más de una definición para un nombre de **función** o un **operador** en el mismo ámbito, lo que se denomina **sobrecarga de función** y **sobrecarga de operador**.

SOBRECARGA (2)

Una sobrecargada es una declaración que se declara con el mismo nombre que una declaración declarada previamente en el mismo ámbito, excepto que ambas declaraciones tienen argumentos diferentes y, obviamente, una definición (implementación) diferente.

SOBRECARGA (3)

Cuando llama a una **función** u **operador** sobrecargado, el compilador determina la definición más apropiada para usar, comparando los tipos de argumentos que ha usado para llamar a la función u operador con los tipos de parámetros especificados en las definiciones. El proceso de seleccionar la función u operador sobrecargado más apropiado se denomina **resolución de sobrecarga.**

UNIVERSIDAD DE LOS LLANOS

Sobrecarga de operadores

- La sobrecarga de operadores es la definición de dos o más operaciones que usan el mismo operador.
- C++ usa un conjunto de símbolos llamados operadores para manipular tipos de datos fundamentales como enteros y puntos flotantes.
- La mayoría de estos símbolos están sobrecargados para manejar varios tipos de datos.

Sobrecarga de operadores

- Por ejemplo, el símbolo para el operador de suma se puede utilizar para sumar dos valores de tipos int, long, double y long double.

14 + 20	14.21 + 20.45

UNIVERSIDAD DE LOS LLANOS

Sobrecarga de operadores

- Operadores que son sobrecargables

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Sobrecarga de operadores

- Operadores que **NO** son sobrecargables

::	.*	.	?:

Operadores unarios

Los operadores unarios operan en un solo operando (`++`, `--`, `-`, `+`, `!`).

Los operadores unarios operan en el objeto para el que fueron llamados: `!obj`, `-obj`, `++obj`, `obj++` u `obj--`.

```
<Type> operator <oper>()
<Type> operator <oper>(const <Type>& obj)
```

UNIVERSIDAD DE LOS LLANOS

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

class Fraccion{
    int numerador, denominador;
public:
    Fraccion(int n, int d){
        numerador = n;
        denominador = d;
    }

    Fraccion operator+(){
        numerador = +numerador;
        return Fraccion(numerador, denominador);
    }
    Fraccion operator-(){
        numerador = -numerador;
        return Fraccion(numerador, denominador);
    }
    string toString(){
        return to_string(numerador)+ "/"+to_string(denominador);
    }
};
```

```cpp
int main() {
    Fraccion f1( n: 3, d: 4);
    Fraccion f2( n: 2, d: 3);
    -f1;
    cout<<f1.toString()<<endl;

    +f2;
    cout<<f2.toString();
    return 0;
}
```

output:

-3/4

2/3

Operadores unarios

UNIVERSIDAD DE LOS LLANOS

Operadores binarios

Los operadores binarios toman dos argumentos como operador de suma (+), operador de resta (-) y operador de división (/).

```
<Type> operator <oper>(const <Type>& obj)
```

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

class Fraccion{
    int numerador, denominador;
public:
    Fraccion(){
        numerador = 0;
        denominador = 1;
    }
    Fraccion(int n, int d){
        numerador = n;
        denominador = d;
    }
    Fraccion operator+(const Fraccion& obj){
        Fraccion f;
        f.numerador = numerador*obj.denominador + obj.numerador*denominador;
        f.denominador = denominador * obj.denominador;
        return f;
    }
    string toString(){
        return to_string(numerador)+ "/"+to_string(denominador);
    }
};
```

```cpp
int main() {
    Fraccion f1( n: 3, d: 4);
    Fraccion f2( n: 2, d: 3);
    Fraccion f3;

    f3 = f1 + f2;
    cout<<f3.toString();
    return 0;
}
```

output:

17/12

Operadores binarios

UNIVERSIDAD DE LOS LLANOS

Operadores relacionales

Hay varios operadores relacionales como (<,>, <=, >=, ==, etc.) que se pueden usar para comparar tipos de datos. Se puede sobrecargar cualquiera de estos operadores para comparar los objetos de una clase.

Operadores relacionales

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

class Fraccion{
    int numerador, denominador;
public:
    Fraccion(int n, int d){
        numerador = n;
        denominador = d;
    }
    bool operator<(const Fraccion& obj){
        return (numerador * obj.denominador < obj.numerador * denominador);
    }
    string toString(){
        return to_string(numerador)+ "/"+to_string(denominador);
    }
};
```

```cpp
int main() {
    Fraccion f1( n: 3, d: 4);
    Fraccion f2( n: 2, d: 3);

    if(f1 < f2){
        cout<<"F1 es menor que F2";
    }else{
        cout<<"F2 es menor que F1";
    }

    return 0;
}
```

output:

F2 es menor que F1

UNIVERSIDAD DE LOS LLANOS

Operadores cin/cout

Los operadores de inserción >> y extracción << de flujos también se pueden sobrecargar para realizar entradas y salidas para tipos definidos por el usuario, como un objeto.

Aquí, es importante hacer que la función de sobrecarga del operador sea una función amiga de la clase porque se llamaría sin crear un objeto.

UNIVERSIDAD DE LOS LLANOS

Operadores cin/cout

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

class Fraccion{
    int numerador, denominador;
public:
    Fraccion(){
        numerador = 0;
        denominador = 1;
    }
    Fraccion(int n, int d){
        numerador = n;
        denominador = d;
    }
    friend ostream &operator<<(ostream &output, const Fraccion &f){
        output << f.numerador << "/" << f.denominador;
        return output;
    }
    friend istream &operator>>(istream &input, Fraccion &f){
        cout << "Ingrese el valor del numerador: ";
        input >> f.numerador;
        cout << "Ingrese el valor del denominador: ";
        input >> f.denominador;
        return input;
    }
};
```

```cpp
int main() {
    Fraccion f1( n: 3, d: 4), f2;

    cin >> f2;
    cout << f1 << endl;
    cout << f2 << endl;

    return 0;
}
```

output:

```
Ingrese el valor del numerador: 2
Ingrese el valor del denominador: 3
3/4
2/3
```

UNIVERSIDAD DE LOS LLANOS

Operadores ++/--

Los operadores de incremento (++) y decremento (--) son dos importantes operadores unarios disponibles en C++.

El siguiente ejemplo explica cómo se puede sobrecargar el operador de incremento (++) para el uso de prefijo y postfijo. De manera similar, puede sobrecargar el operador (--).

UNIVERSIDAD DE LOS LLANOS

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

class Fraccion{
    int numerador, denominador;
public:
    Fraccion(int n, int d){
        numerador = n;
        denominador = d;
    }
    // prefijo ++
    Fraccion operator++(){
        numerador = numerador + denominador;
        return Fraccion(numerador, denominador);
    }
    // postfijo
    Fraccion operator++(int){
        numerador = numerador + denominador;
        return Fraccion(numerador, denominador);
    }
    string toString(){
        return to_string(numerador) + "/" + to_string(denominador);
    }
};
```

```cpp
int main() {
    Fraccion f1( n: 3, d: 4);

    ++f1;
    cout<<f1.toString()<<endl;

    f1++;
    cout<<f1.toString()<<endl;

    return 0;
}
```

output:

7/4

11/4

UNIVERSIDAD DE LOS LLANOS

Operadores de asignación

Puede sobrecargar el operador de asignación (=) al igual que otros operadores (+=, -=, *=, /=) y puede usarse para crear un objeto como el constructor de copia.

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

class Fraccion{
    int numerador, denominador;
public:
    Fraccion(int n, int d){
        numerador = n;
        denominador = d;
    }
    void operator = (const Fraccion &f){
        numerador = f.numerador;
        denominador = f.denominador;
    }

    void operator += (const Fraccion &f){
        numerador = numerador * f.denominador + f.numerador * denominador;
        denominador = f.denominador * denominador;
    }

    string toString(){
        return to_string(numerador) + "/" + to_string(denominador);
    }
};
```

```cpp
int main() {
    Fraccion f1( n: 3, d: 4), f2( n: 2, d: 3);
    cout<<"Primera fracción: "<< f1.toString()<<endl;
    cout<<"Segunda fracción: "<< f2.toString()<<endl;

    f1 = f2;
    cout<<"Primera fracción: "<<f1.toString()<<endl;

    f1 += f2;
    cout<<"Primera fracción: "<<f1.toString()<<endl;

    return 0;
}
```

output:

```
Primera fracción: 3/4
Segunda fracción: 2/3
Primera fracción: 2/3
Primera fracción: 12/9
```

UNIVERSIDAD DE LOS LLANOS

Taller

Realizar un programa de una calculadora de fracciones, debe cumplir con las siguientes restricciones:

- La calculadora debe contener las operaciones aritméticas básicas +, -, *, /, entre una fracción y otra. Ejemplo 3/4 + 2/3.
- Sobrecargar la función cin/cout, validar que no puede existir fracciones con denominador = 0 (división por cero).
- Agregar opciones para comparar dos fracciones: ==, !=, <, <=, >, >=.
- Agregar opciones para autoincremento y autodecremento.

Bibliografía

- Tutorialspoint https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm
- Behrouz A. Forouzan, Richard Gilberg - C++ Programming An Object-Oriented Approach. 2019.

UNIVERSIDAD DE LOS LLANOS

Gracias...

UNIVERSIDAD DE LOS LLANOS®