Grado en Ingeniería Información

PROGRAMACIÓN II - Sesión 5

Tema 2.

Sobrecarga

Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

UNIVERSIDAD NEBRIJA

Puntero *this*

ÍNDICE

Puntero *this*

this es un ***puntero implícito*** o una ***referencia*** al propio **objeto** instanciado a partir de la **clase**.

Está **disponible** en la **implementación** de cualquier **método** u **operación** de la **clase**.

this: es la **dirección de memoria** de dicho objeto

****this:*** es el **propio objeto**

this permite saber el **objeto al que pertenece una operación** o **diferenciar variables con el mismo nombre pero distinto ámbito**.

this evita la ambigüedad entre el **parámetro de un método de una clase** y un **atributo de la misma**.

```
class Temperatura {
private:
     int minima;
     int maxima;
     int actual;
    void imprimir ();
public:
    void LaTemperatura (int minima, int maxima, int actual);
};
```

```
void Temperatura::LaTemperatura (int minima, int maxima, int actual)
{
    this->minima = minima;
    this->maxima = maxima;
    this->actual = actual;
    imprimir();
}
void Temperatura::imprimir ()
{
    cout << endl << endl;
    cout << "\t" << minima << "  " << actual << "  " << maxima;
}
```

```
int main()
{
    Temperatura temperatura1, temperatura2;

    temperatura1.LaTemperatura (15, 35, 25);
    temperatura2.LaTemperatura2 (20, 40, 30);
    cout << endl << endl;
    return 0;
}
```

UNIVERSIDAD NEBRIJA

Composición

ÍNDICE

La **composición** permite crear **miembros** (***atributos*** o ***métodos/funciones***) en **una clase cuyo tipo sea otra clase previamente creada**.

La **composición** permite crear **miembros** (***atributos*** o ***métodos/funciones***) en **una clase cuyo tipo sea otra clase previamente creada**.

```
class Cfecha {
    private:
        int dia;
        int mes;
        int anyo;
    public:
        Cfecha ():dia(1), mes(1), anyo(1900) {}
        Cfecha (Cfecha const &f);
        void mostrarFecha ();
};
class Persona  {
    private:
        string nombre;
        Cfecha fechaNac;        // Composición: objeto miembro
    public:
        Persona (string const &n, Cfecha const &fnac);
        void imprime ();
};
```

```
Persona::Persona
        (string const &n, Cfecha const & fnac):fechaNac (fnac){
    nombre= n;
}
void Persona::imprime () {
    cout << "Me llamo " << nombre  << " y naci en " ;
   // Se llama un método de la clase Cfecha
    fechaNac.mostrarFecha();
   cout << endl;
}
```

```
Cfecha:: Cfecha (Cfecha const &f) {
    dia = f.dia;
    mes = f.mes;
    anyo  = f.anyo;
}

void Cfecha::mostrarFecha () {
    cout << dia << "/" << mes << "/" << anyo << endl;
}
```

UNIVERSIDAD NEBRIJA

T2. Sobrecarga

2.1. Concepto de sobrecarga

2.2. Características de la sobrecarga

2.3. Sobrecarga de operadores binarios

2.4. Sobrecarga de operadores unarios

ÍNDICE

UNIVERSIDAD NEBRIJA

T2. Sobrecarga

2.1. Concepto de sobrecarga

2.2. Características de la sobrecarga

2.3. Sobrecarga de operadores binarios

2.4. Sobrecarga de operadores unarios

ÍNDICE

Concepto de sobrecarga

Sobrecarga es **redefinir el comportamiento** de un **operador** o **función** para que realice **operaciones diferentes** en función del tipo de **operandos** o **parámetros** con los que trabaja.

En C++:

- La mayoría de los **operadores** están **sobrecargados**, aunque **conservan el sentido** y **comportamiento originales** cuando se usan con los ***tipos de datos básicos***.
- La **sobrecarga** de los **operadores** permite que los **operandos** sean ***tipos definidos por el programador*** y que los **operadores** tengan **otro comportamiento,** más adecuado para ellos.

Sintaxis:

Consiste en la definición de una ***función***, donde el **nombre de la función** es la **palabra reservada *operator*** seguida del ***operador*** que se sobrecarga:

```
<tipoDato> operator <operador>(<parámetros>)
    {
        <sentencias>;        // Implementación
    }
```

La **sobrecarga** de un **operador** **no** puede **cambiar el número de operandos** o la **asociatividad** y **precedencia** del mismo.

Los **operadores sobrecargados**, pueden implementarse como:

- **Funciones miembro de una clase**
- **Funciones generales de la aplicación**, en cuyo caso se suelen declarar como **funciones *friend* de la clase** **que especifica el tipo de los operandos para los que se sobrecarga dicho operador**.

Si se **sobrecargan operadores unarios**, el **método donde se implementa** su comportamiento suele ser una **función miembro de la clase para la que se ha implementado la sobrecarga**.

Si **el operador sobrecargado actúa sobre diferentes tipos de clases** lo más frecuente es que **se implemente** como una **función general de la aplicación** y se establezcan **relaciones de tipo *friend* con todas las clases implicadas**.

Ejemplo:

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

class Rectangulo
{   private :
        int ladox;
        int ladoy;
    public :
        Rectangulo (void);
        Rectangulo (int x, int y);

        void mostrarLados (void);
        int getladoX()const;
        int getladoY()const;
        int calcularArea (void);
        int calcularPerimetro (void);

        // Operador BINARIO CON UN PARÁMETRO
    //    Rectangulo operator +(const Rectangulo &r);

        Rectangulo suma (const Rectangulo &r);
};


// Operador BINARIO
// Rectangulo operator +(const Rectangulo &r1, const Rectangulo &r2);
```

Ejemplo:

```
Rectangulo::Rectangulo (void)  {
    ladox=0;
    ladoy=0;
}

Rectangulo::Rectangulo (int x, int y)  {
        ladox = x;
        ladoy = y;
}

int Rectangulo::getladoX()const {
    return ladox;
}
int Rectangulo::getladoY()const {
    return ladoy;
}

int Rectangulo::calcularArea (void)  {
    return ladox * ladoy;
}

int Rectangulo::calcularPerimetro (void)  {
    return 2*(ladox + ladoy);
}
```

Ejemplo:

```
// OPERADOR BINARIO CON UN PARÁMETRO
Rectangulo Rectangulo::operator +(const Rectangulo &r)  {
    return Rectangulo (ladox + r.getladoX(), ladoy + r.getladoY());
}

Rectangulo Rectangulo::suma (const Rectangulo &r)  {
    return Rectangulo (ladox + r.getladoX(), ladoy + r.getladoY());
}

// OPERADOR BINARIO
Rectangulo operator +(const Rectangulo &r1, const Rectangulo &r2)  {
    return Rectangulo(r1.getladoX() + r2.getladoX(), r1.getladoY() + r2.getladoY());
}
void Rectangulo::mostrarLados (void)  {
        cout << "\n\tEl lado horizontal vale: " << ladox;
        cout << "\n\tEl lado vertical vale: " << ladoy << endl;
}
```

Ejemplo:

```
// Programa Principal
int main(void)
{
    Rectangulo R1(5, 7), R2(20, 2), Rsuma1, Rsuma2;

    cout << "\n\n\tDatos del rectangulo R1 " << endl;
    cout << "\t======================== " << endl;
    R1.mostrarLados ();

    cout << "\n\n\tDatos del rectangulo R2 " << endl;
    cout << "\t======================== " << endl;
    R2.mostrarLados ();

    // Forma con un método que suma rectángulos
    // El objeto Rsuma tendrá ladox = 25 y ladoy = 9
    Rsuma2 = R1.suma(R2);

    cout << "\n\n\tDatos del rectangulo Rsuma " << endl;
    cout << "\t========================== " << endl;
    Rsuma2.mostrarLados ();

    // Forma con el operador + sobrecargado
    // El objeto Rsuma tendrá ladox = 25 y ladoy = 9
    Rsuma1 = R1 + R2;

    cout << "\n\n\tDatos del rectangulo Rsuma " << endl;
    cout << "\t========================== " << endl;
    Rsuma1.mostrarLados ();

    return 0;
}
```

Operadores que **NO** se pueden **sobrecargar** :

.	Operador selector directo de miembro.
::	Operador de acceso a ámbito.
.*	Operador de indirección de puntero-a-miembro.
? :	Operador condicional abreviado.
# y # #	Directivas del preprocesador.

Los siguientes **operadores** sólo se pueden sobrecargar cuando **se definen como miembros de una clase**:

= **[ ]** **->** **new** **delete**

Los **parámetros** en la **sobrecarga de operadores** deben de ser **tipos enumerados** o **estructurados**:

enum

struct

union

class

UNIVERSIDAD NEBRIJA

T2. Sobrecarga

2.1. Concepto de sobrecarga

2.2. Características de la sobrecarga

2.3. Sobrecarga de operadores binarios

2.4 Sobrecarga de operadores unarios

ÍNDICE

Los **operadores binarios** son aquellos que realizan operaciones entre **dos operandos**.

La implementación de la sobrecarga recibirá **dos parámetros**. El **primero** será **el operando izquierdo** al operador y el **segundo el operando derecho** al operador.

Caso 1: El **operador** está definido como una **función general**

Recibirá dos operandos como **parámetros.**

<tipDat> ***operator<operBin>*** *(<tipDat> <identif1>, <tipDat> <identif2>);*

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

class Rectangulo
{    private :
         int ladox;
         int ladoy;
     public :
         Rectangulo (void);
         Rectangulo (int x, int y);

         void mostrarLados (void);
         int getladoX()const;
         int getladoY()const;
         int calcularArea (void);
         int calcularPerimetro (void);

         Rectangulo suma (const Rectangulo &r);
};

// Operador BINARIO definido como FUNCIÓN GENERAL
Rectangulo operator +(const Rectangulo &r1, const Rectangulo &r2);
```

```
// Programa Principal
int main(void)
{
    Rectangulo R1(5, 7), R2(20, 2), Rsuma1, Rsuma2;

    cout << "\n\n\tDatos del rectangulo R1 " << endl;
    cout << "\t========================= " << endl;
    R1.mostrarLados ();

    cout << "\n\n\tDatos del rectangulo R2 " << endl;
    cout << "\t========================= " << endl;
    R2.mostrarLados ();

    // El objeto Rsuma tendrá ladox = 25 y ladoy = 9
    Rsuma1 = R1 + R2;

    cout << "\n\n\tDatos del rectangulo Rsuma " << endl;
    cout << "\t============================ " << endl;
    Rsuma1.mostrarLados ();
```

```
// OPERADOR BINARIO COMO FUNCIÓN GENERAL
Rectangulo operator +(const Rectangulo &r1, const Rectangulo &r2)  {

    Rectangulo R(r1.getladoX() + r2.getladoX(), r1.getladoY() + r2.getladoY());

    return R;
}
```

Caso 2: El **operador** está definido como una **función o método miembro de la clase.**

Recibirá sólo un operando como **parámetro** y será el que ocupa el **lado derecho de la operación**, mientras que el propio objeto de la **clase** donde se está definiendo el operador será el que ocupa el **lado izquierdo de la misma**.

<tipoDato> ***operator*** ***<operBinario>*** *(<tipoDato> <identificador>);*

Normalmente, ***<tipoDato>*** es de la clase para la que estamos sobrecargando el operador.

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

class Rectangulo
{   private :
        int ladox;
        int ladoy;
    public :
        Rectangulo (void);
        Rectangulo (int x, int y);

        void mostrarLados (void);
        int getladoX()const;
        int getladoY()const;
        int calcularArea (void);
        int calcularPerimetro (void);

        // Operador BINARIO DE LA CLASE - UN PARÁMETRO
        Rectangulo operator +(const Rectangulo &r);

        Rectangulo suma (const Rectangulo &r);
};
```

```
// Programa Principal
int main(void)
{
    Rectangulo R1(5, 7), R2(20, 2), Rsuma1, Rsuma2;

    cout << "\n\n\tDatos del rectangulo R1 " << endl;
    cout << "\t======================== " << endl;
    R1.mostrarLados ();

    cout << "\n\n\tDatos del rectangulo R2 " << endl;
    cout << "\t======================== " << endl;
    R2.mostrarLados ();

    // El objeto Rsuma tendrá ladox = 25 y ladoy = 9
    // R1 es el parámetro implícito y R2 el parámetro
    Rsuma1 = R1 + R2;

    cout << "\n\n\tDatos del rectangulo Rsuma " << endl;
    cout << "\t========================== " << endl;
    Rsuma1.mostrarLados ();

...

// OPERADOR BINARIO DE LA CLASE - CON UN PARÁMETRO
Rectangulo Rectangulo::operator +(const Rectangulo &r)  {

    return Rectangulo (ladox + r.getladoX(), ladoy + r.getladoY());
}
```

UNIVERSIDAD NEBRIJA

T2. Sobrecarga

2.1. Concepto de sobrecarga

2.2. Características de la sobrecarga

2.3. Sobrecarga de operadores binarios

2.4. Sobrecarga de operadores unarios

ÍNDICE

Los **operadores unarios** son aquellos que **sólo requieren un operando**.

Por ejemplo, el **operador de incremento** (**++**) o el **operador lógico de negación** (**!**).

Caso 1: El **operador** está definido como una **función general.**

Recibirá el operando requerido como **parámetro**.

Sintaxis de la forma prefija:

<tipoDato> ***operator <operUnario>*** *(<tipDat> <identif>);*

Sintaxis de la forma postfija:

<tipoDato> ***operator <operUnario>*** *(<tipDat> <identif>,* ***int****);*

Caso 2: El **operador** está definido como un **método de la clase,** entonces **no recibirá parámetros.**

El **operando** será **el propio objeto de la clase** donde se está definiendo el operador.

Normalmente el ***<tipoDato>*** es la **clase** para la que estamos sobrecargando el operador.

Sintaxis de la forma <u>prefija</u>:

<tipoDato> ***operator <operUnario> ();***

Sintaxis de la forma <u>postfija</u>:

<tipoDato> ***operator <operUnario> (int);***

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

class Rectangulo
{   private :
        int ladox;
        int ladoy;
    public :
        Rectangulo (void);
        Rectangulo (int x, int y);

        void mostrarLados (void);
        int getladoX()const;
        int getladoY()const;
        void setLadox(int newLadox);
        void setLadoy(int newLadoy);
        int calcularArea (void);
        int calcularPerimetro (void);

        // Forma postfija del operador: r++
        Rectangulo operator ++(int);

        // Forma prefija del operador: ++r
        Rectangulo operator ++();

        // Operador BINARIO CON UN PARÁMETRO
        Rectangulo operator +(const Rectangulo &r);

        Rectangulo suma (const Rectangulo &r);

};

// Forma prefija del operador: r--
Rectangulo &operator --(Rectangulo &r);
```

```
// Forma postfija del operador: r++
Rectangulo Rectangulo::operator ++(int)  {
    return Rectangulo(ladox++, ladoy++);
}

// Forma prefija del operador: ++r
Rectangulo Rectangulo::operator ++()  {
    ++ladox;
    ++ladoy;
    return *this;
}

 void Rectangulo::setLadox(int newLadox)
 {
     ladox = newLadox;
 }

 void Rectangulo::setLadoy(int newLadoy)
 {
     ladoy = newLadoy;
 }
```

```
// OPERADOR UNARIO COMO FUNCIÓN GENERAL
// Forma prefija del operador: r--
Rectangulo &operator --(Rectangulo &r)  {
    int lx{r.getladoX()}, ly{r.getladoY()};

    lx--; ly--;
    r.setLadox(lx);
    r.setLadoy(ly);

    return r;
}
```

```
// Programa Principal
int main(void)
{
    Rectangulo R1(5, 7), R2(20, 2), R3(10, 15);

    cout << "\n\n\tDatos del rectangulo R1 " << endl;
    cout << "\t======================== " << endl;
    R1++;
    R1.mostrarLados ();

    cout << "\n\n\tDatos del rectangulo R2 " << endl;
    cout << "\t======================== " << endl;
    ++R2;
    R2.mostrarLados ();

    cout << "\n\n\tDatos del rectangulo R3 " << endl;
    cout << "\t======================== " << endl;
    --R3;
    R3.mostrarLados();

...
```

Ejercicio:

1. Realizar la sobrecarga de otras operaciones binarias y unarias sobre la clase Rectángulo: resta, decremento, >, <, ==.

2. Definir la clase Complejo con dos atributos reales, parte real y parte imaginaria. Declarar e implementar los constructores y métodos/funciones necesarios para acceder a sus variables. Finalmente, definir e implementar operadores sobrecargados para realizar operaciones binarias y unarias.