

# Sobrecarga de Operadores



Ma. Guadalupe Roque Díaz de León

# Sobrecarga de operadores

---

- En C ++, permite que los operadores se apliquen para clases definidas por el usuario.
- C ++ tiene la capacidad de proporcionar a los operadores un significado especial para un tipo de datos, llamada **sobrecarga del operador**.
- Ejemplo: la **sobrecarga del operador '+' en la clase String** para concatenar dos strings simplemente usando +.
- Otras clases donde los operadores aritméticos podrían sobrecargarse son Número complejo, Número fraccionario, Entero grande, etc.

# ¿Cuál es la diferencia entre las funciones **operator** y las funciones normales?

---

Las funciones **operator** son las iguales a las funciones normales.

Las 2 **diferencias** son:

1. El nombre de una **función operator** es siempre la palabra reservada **operator** seguido del **símbolo** del operador.
2. Las funciones **operator** se invocan cuando se utiliza el operador correspondiente.

# Función operator

- Sintaxis

tipo **operator** símbolo (parámetros) {

... ..

}

```
Complex operator + (Complex const &obj) {  
    Complex res;  
    res.real = real + obj.real;  
    res.imag = imag + obj.imag;  
    return res;  
}
```

```
Complex a(1,2), b(3,4), c;  
c = a + b;
```

```
class Complex {
private:
    int real, imag;
public:
    Complex(int _real = 0, int _imag =0) {
        real = _real;
        imag = _imag;
    }

    // This is automatically called when '+' is used with  between two Complex objects
    Complex operator + (Complex const &obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }

    void print() {
        cout << real << " + i" << imag << endl;
    }
};
```

```
int main(){
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
} // https://www.programiz.com/cpp-programming/operator-overloading
```


# ¿Podemos sobrecargar a todos los operadores?

---

Casi todos los operadores se pueden sobrecargar, excepto:

- :: (scope resolution)
- . (member selection)
- .\* (member selection through pointer to function)
- ?: (ternary operator)
- 

## Puntos importantes sobre la sobrecarga del operador:

1) Para que la sobrecarga del operador funcione , al menos uno de los operandos debe ser un objeto de la clase definida por el usuario.

2) **Operador de asignación** : **el compilador crea automáticamente un operador de asignación** = predeterminado con cada clase. El operador de asignación predeterminado asigna todos los miembros del lado derecho al lado izquierdo.

# Funciones Libres



Son aquellas funciones que se generan para un fin específico y no pertenecen a ninguna clase en particular.

*Ejemplo: función que encuentre el mayor de dos números enteros.*

```
int mayor(int a, int b){  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

```
int a, b, max;  
cin >> a >> b;  
max = mayor(a,b);
```

# Función Miembro-Método

---

- Es aquella función que pertenece a una clase en particular.

*Ejemplo: una función llamada leerArchivo() que pertenece a la Clase Series y lee todas las series desde un archivo.*

```
void Series::leerArchivo(){  
    // Instrucciones  
}
```

```
Series listaSeries;  
  
listaSeries.leerArchivo();
```



# Funciones Amigas



- Una **función amiga** 🧑🧑 es aquella función libre 🦋 a la cual se le permite acceder 😬 los atributos privados.
- Para que una **función libre** pueda ser **función amiga**, se debe **poner el encabezado de la función dentro del prototipo de la clase**, anteponiendo la palabra friend.

*friend Complejo resta(Complejo c1, Complejo c2);*

# Ejemplos de Complejos

---

- Los número Complejos son aquellos que tienen una parte real y una parte imaginaria.

```
private:  
    int real, imag;
```

# Ejemplos de Complejos

---

- Los suma de dos números complejos

$$(8 + 5i) + (5 - 2i) = (13 + 3i)$$

- Los resta de dos números complejos da como resultado un número complejo.

$$(8 + 5i) - (5 - 2i) = (3 + 7i)$$

# Complex-función friend operator

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

class Complex {
private:
    int real, imag;
public:
    Complex(int r = 0, int i =0) {real = r;    imag = i;}
    void print() { cout << real << " + i" << imag << endl; }

    // The global operator function is made friend of this class so
    // that it can access private members
    friend Complex operator + (Complex const &, Complex const &);
};

Complex operator + (Complex const &c1, Complex const &c2)
{
    return Complex(c1.real + c2.real, c1.imag + c2.imag);
}

int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
    return 0;
}
```

# class Complejo-función miembro

---

```
class Complejo{  
    private:  
        int real, imag;  
    public:  
        Complejo();  
        Complejo(int _real, int _imag);  
  
        int getReal();  
        void setReal(int _real);  
  
        int getImag();  
        void setImag(int _imag);  
  
        Complejo suma(Complejo c2);  
}
```

# Función Miembro vs Libre

// Función Miembro

```
Complejo Complejo::suma(Complejo c2){  
    return Complejo(real + c2.real, imag + c2.imag);  
}
```

// Función libre

```
Complejo suma(Complejo c1, Complejo c2){  
    return Complejo (c1.getReal( ) + c2.getReal( ),  
        c1.getImag( ) + c2.getImag( ) );  
}
```

Complejo a(8,5), b(5,2), c;  
c = a.suma(b);

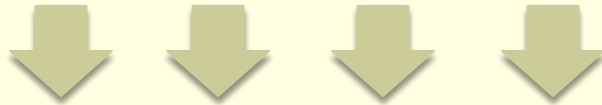
Complejo a(8,5), b(5,2), c;  
c = suma(a,b);

# Class Complejo con función Amiga 🎉

```
class Complejo{  
    private:  
        int real, imag;  
    public:  
        Complejo( );  
        Complejo(int _real, int _imag);  
  
        void setReal(int _real);  
        void setImag(int _imag);  
  
        int getReal( );  
        int getImag( );  
  
        Complejo suma(Complejo c2);  
        // función Amiga 🎉  
        friend Complejo resta(Complejo c1, Complejo c2)  
}
```

# De fx libre a fx amiga de class Complejo

```
Complejo resta(Complejo c1, Complejo c2){  
    return Complejo (c1.getReal() - c2.getReal(),  
                     c1.getImag() - c2.getImag());  
}
```



```
Complejo resta(Complejo c1, Complejo c2){  
    return Complejo (c1.real - c2.real,  
                     c1.imag - c2.imag);  
}
```

Complejo a(8,5), b(5,2), c;  
c = resta(a,b);



# Ejemplos de sobrecarga:

---

## 1. Con métodos de la clase

[https://replit.com/@roque\\_itesm\\_mx/SobrecargaConFuncionesMiembro](https://replit.com/@roque_itesm_mx/SobrecargaConFuncionesMiembro)

## 2. Con funciones amigas :

a. [https://replit.com/@roque\\_itesm\\_mx/OverloadingOperator-using-friend-functions](https://replit.com/@roque_itesm_mx/OverloadingOperator-using-friend-functions)

## 3. Con funciones normales-libres :

a. [https://replit.com/@roque\\_itesm\\_mx/SobreCarcaOperadoresFuncionesNormales](https://replit.com/@roque_itesm_mx/SobreCarcaOperadoresFuncionesNormales)

# Actividad-

---

Añade a la clase Fracción una función amiga llamada suma - que sume dos fracciones y retorne la suma de las dos fracciones -  
Añade la llamada desde el main-

# Actividad-

## Solución

```
Fracciones suma(Fracciones f1, Fracciones f2){
    int iNum, iDen;
    if (f1.iDen == f2.iDen)
    {
        iNum = f1.iNum + f2.iNum;
        iDen = f1.iDen ;
    }
    else
    {
        iNum = (f1.iNum * f1.iDen + f1.iDen * f2.iNum);
        iDen = (f1.iDen * f2.iDen);
    }

    Fracciones nuevo(iNum, iDen);
    return nuevo;
}
```