

Programación Orientada o Objetos

Sobre Carga de Operadores

*Univesidad de Los Andes - Facultad de Ingeniería
Escuela de Sistemas
Programación Digital II*

*Profesor: Gilberto Diaz
gilberto@ula.ve*

Las operaciones tradicionales que se encuentran presentes en sentencias sencillas como la de asignación

```
NombreClase obj1, obj2;  
obj2 = obj1
```

No se pueden realizar cuando se tienen objetos con atributos distintos a los tipos de datos comunes.

Es muy importante mantener una notación consisa para las operaciones más comunes:

- Suma
- Resta
- Multiplicación
- División, etc

C++ soporta una serie de operadores para los tipos de datos que vienen incorporados en el lenguaje.

Sin embargo, para Clases definidas por el usuario, donde se realizan operaciones como:

- Aritmética compleja
- Álgebra matricial
- Señales lógicas
- Cadenas de caracteres

C++ no puede darle soporte directo

Como ejemplo veamos la clase Matriz ([Matriz.h](#))

```
#ifndef MATRIZ_H
#define MATRIZ_H
class Matriz {
    private:
        int filas, columnas;
        float **elementos;
    public:
        //Constructores
        //Métodos de acceso .....
        //Métodos de modificación .....
        //Métodos miscelaneos
        void sumarMatrices(Matriz);
        //Destructor
};
```

Como ejemplo veamos la clase Matriz ([Matriz.cpp](#))

```
#include "Matriz.h"
```

-
-
-

```
void Matriz :: sumarMatrices(Matriz m){
```

```
    int i,j;
```

```
    if ( filas == m.filas && columnas == m.columnas ){
```

```
        for(i=0; i<m.filas; i++)
```

```
            for(j=0; j<m.columnas; j++)
```

```
                elementos[i][j] += m.elementos[i][j];
```

```
    }
```

```
}
```

Como ejemplo veamos la clase Matriz ([Principal.cpp](#))

```
#include "Matriz.h"
int main(){
    Matriz m1(2, 2), m2(2, 2), m3;

    m1.Inicializar(2.0);
    m2.Inicializar(2.0);

    m2.sumarMatrices(m1);
}
```

La idea es mantener, como dijimos al principio una notación sencilla, es decir, utilizar una sentencia como esta:

Matriz $m1(2, 2)$, $m2(2, 2)$, $m3$;

▪

▪

$m3 = m1 + m2$;

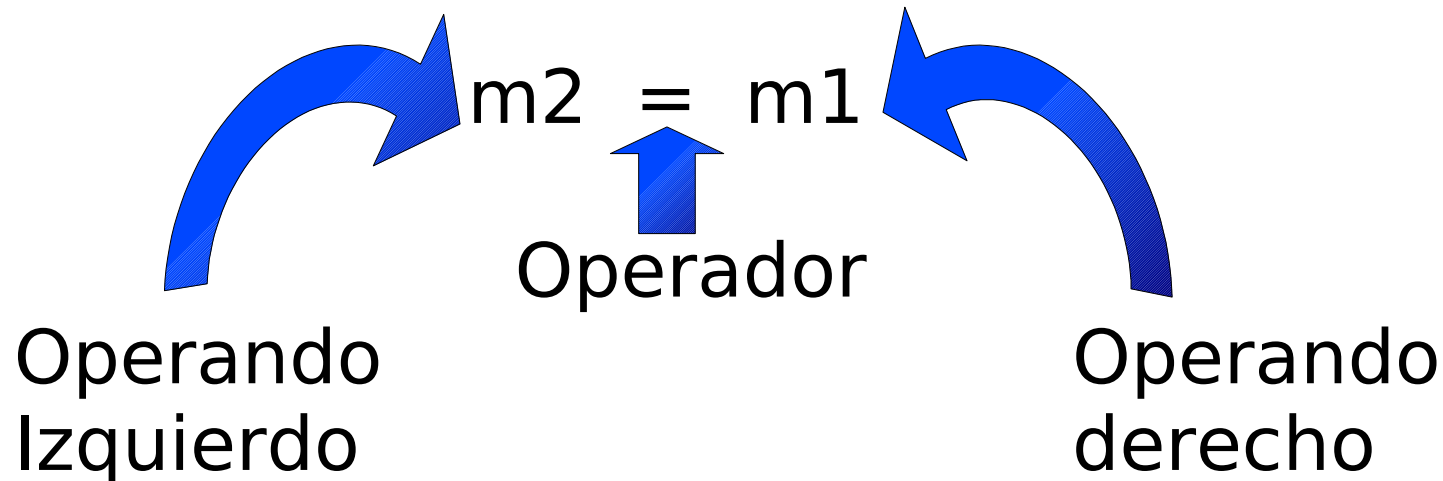
Para ello, C++ proporciona una manera de cambiar el comportamiento original de los operadores que tiene incorporado el lenguaje. A esto se le llama **sobrecarga de operadores**.

En nuestro ejemplo debemos sobrecargar dos operadores: '=' y '+' para poder construir sentencias como


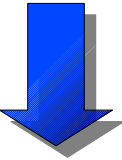
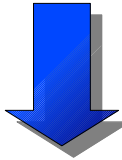
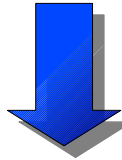
$$m3 = m1 + m2$$

En el operador de asignación (=) están presentes dos operandos.

En el caso de la clase Matriz, la operación tiene dos objetos de esta clase:



C++ utiliza métodos para cambiar el comportamiento original de un operador. Estos métodos tienen el siguiente formato:

tipo de dato del resultado	palabra reservada	operador a sobrecargar	argumentos del método
			
TipoDato	operator	operador	(parámetros){
sentencias;			
return TipoDato;			
}			

En nuestro ejemplo de la operación de asignación para la clase Matriz, este método es de la siguiente manera:

tipo de dato del resultado	palabra reservada	operador a sobrecargar	argumentos del método
Matriz	operator	=	(const Matriz& m){
	<code>this->filas</code>		<code>= m.filas;</code>
		
	<code>return *this;</code>		
	}		

El argumento representa el operando derecho.

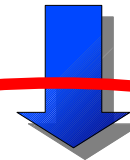
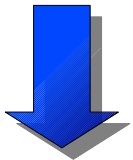
m2 = m1;

tipo de dato del resultado

palabra reservada

operador a sobrecargar

argumentos del método



```
Matriz operator = (const Matriz& m){  
    this->filas = m.filas;  
    ....  
    return *this;  
}
```

El apuntador a objeto “this” representa el operando izquierdo.

```
m2 = m1;
```

tipo de dato del resultado palabra reservada operador a sobrecargar argumentos del método



```
Matriz operator = (const Matriz& m){  
  this->filas = m.filas;  
  ....  
  return *this;  
}
```

El objeto que invoca el método es el operando izquierdo (**this**)

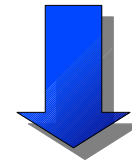
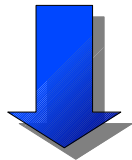
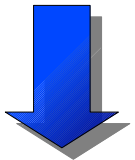
```
m2 = m1;
```

tipo de
dato del
resultado

palabra
reservada

operador
a sobrecargar

argumentos
del método



```
Matriz operator = (const Matriz& m){  
  this->filas = m.filas;  
  ....  
  return *this;  
}
```

El resultado es una Matriz

`m2 = m1;`

tipo de dato del resultado

palabra reservada

operador a sobrecargar

argumentos del método

`Matriz operator = (const Matriz& m){`
`this->filas = m.filas;`
`....`
`return *this;`
`}`

En este caso particular se retorna el objeto **this* pues él mismo es el que recibe el resultado.

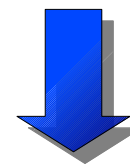
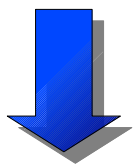
```
m2 = m1;
```

tipo de dato del resultado

palabra reservada

operador a sobrecargar

argumentos del método



```
Matriz operator = (const Matriz& m){  
    this->filas = m.filas;  
    ....  
    return *this;  
}
```

La implementación es como sigue: (Matriz.h)

```
#ifndef MATRIZ_H
#define MATRIZ_H
class Matriz {
    private:
        int filas, columnas;
        float **elementos;
    public:
        //Constructores
        //Métodos de acceso .....
        //Métodos de modificación .....
        //Métodos miscelaneos
        void sumarMatrices(Matriz);
        Matriz operator = (const Matriz&);
        //Destructor
};
```

La implementación es como sigue: ([Matriz.cpp](#))

```
Matriz Matriz :: operator = (const Matriz& mat){  
    int i,j;  
    if(this->filas != 0 || this->columnas != 0)  
        delete [] this->elementos;  
  
    this->filas = mat.filas;    this->columnas = mat.columnas;  
    this->elementos = new float * [this->filas*this->columnas];  
  
    for(i=0; i<this->filas; i++){  
        this->elementos[i] = new float[columnas];  
        for(j=0; j<this->columnas; j++){  
            this->elementos[i][j] = mat.elementos[i][j];  
        }  
    }  
    return *this;  
}
```

Ahora tenemos la posibilidad de utilizar una sentencia sencilla y natural. ([Principal.cpp](#))

```
#include "Matriz.h"  
int main(){  
    Matriz m1(2, 2), m2(2, 2), m3;  
  
    m1.Inicializar(2.0);  
    m2.Inicializar(2.0);  
  
    m3 = m1;  
}
```

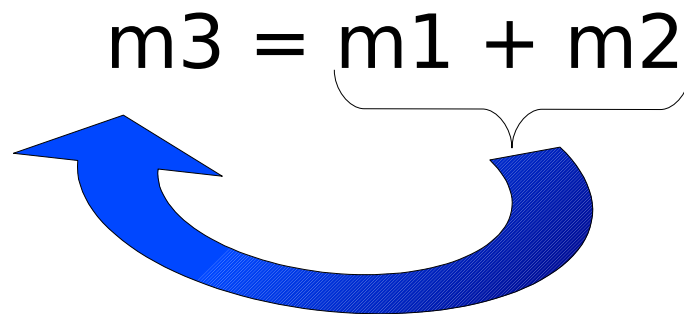
Implementación del operador suma (+) (Matriz.h)

```
#ifndef MATRIZ_H
#define MATRIZ_H
class Matriz {
    private:
        int filas, columnas;
        float **elementos;
    public:
        .....
        //Métodos miscelaneos
        void sumarMatrices(Matriz);
        Matriz operator = (const Matriz&);
        Matriz operator + (const Matriz&);
        //Destructor
};
```

La implementación es como sigue: (Matriz.cpp)

```
Matriz Matriz :: operator + (const Matriz& m){
    int i,j;
    if(this->filas==m.filas && this->columnas==m.columnas ){
        Matriz res(m);
        for(i=0; i<m.filas; i++)
            for(j=0; j<m.columnas; j++)
                res.elementos[i][j] += this->elementos[i][j];
        return res;
    }
    else{
        Matriz res;
        return res;
    }
}
```

En el caso del operador de suma el resultado no se almacena en ninguno de los operandos involucrados. Por esta razón, se debe crear un nuevo objeto tipo Matriz el cual se almacenará en un tercer objeto del mismo tipo.

$$m3 = m1 + m2$$


Por eso se crea un nuevo objeto que se retorna:
(Matriz.cpp)

```
Matriz Matriz :: operator + (const Matriz& m){
```

```
int i,j;
```

```
if(this->filas==m.filas && this->columnas==m.columnas ){
```

```
    Matriz res(m);
```

```
    for(i=0; i<m.filas; i++)
```

```
        for(j=0; j<m.columnas; j++)
```

```
            res.elementos[i][j] += this->elementos[i][j];
```

```
    return res;
```

```
}
```

```
else{
```

```
    Matriz res;
```

```
    return res;
```

```
}
```

```
}
```

Ahora tenemos la posibilidad de utilizar una sentencia sencilla y natural. ([Principal.cpp](#))

```
#include "Matriz.h"  
int main(){  
    Matriz m1(2, 2), m2(2, 2), m3;  
  
    m1.Inicializar(2.0);  
    m2.Inicializar(2.0);  
  
    m3 = m1 + m2;  
}
```

Tarea:

Fecha de entrega: Domingo 1ro de Mayo hasta las 8:00 pm

Sobrecargar los siguientes operadores:

Resta: -

Multiplicación: *

Inversa: !