

Vigésimo quinta Sesión

Metodologías y Técnicas de Programación II

**Programación Orientada a
Objeto (POO)
C++**

Aspectos Avanzados II

Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

C++
Mi primera Clase

Repaso de Conceptos

Estándares de Programación
Punteros y Memoria

E/S
Control y Operadores

Clases y Objetos en C++

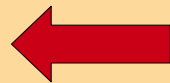
Uso y aplicación
Constructores
Constantes e "inline"

Funciones Amigas
Sobrecarga de Funciones

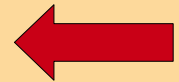


Sobrecarga

De Operadores



Creación Dinámica de Objetos



Herencia.

Tipos de Visibilidad

Herencia Múltiple

Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

Plantillas

Contenedores

Iteradores

25.1 Repaso

Sobrecarga

Hemos visto que las funciones podían ser sobrecargadas.

De igual forma los operadores también pueden sobrecargarse.

En realidad la mayoría de los operadores en C++ están sobrecargados.

Por ejemplo el operador + realiza distintas acciones cuando los operandos son enteros, o en coma flotante.

El operador * se puede usar como operador de multiplicación o como operador de indirección.

C++ permite al programador sobrecargar a su vez los operadores para sus propios usos.

```
<tipo> operator <operador> (<argumentos>);  
<tipo> operator <operador> (<argumentos>)  
{  
    <sentencias>;  
}
```

25.1 Repaso

```
class Tiempo
{
public:
    Tiempo(int h=0, int m=0)
        : hora_(h), minuto_(m) {}
    void Mostrar();
    Tiempo operator+(Tiempo h);
private:
    int hora_;
    int minuto_;
};

Tiempo Tiempo::operator+(Tiempo h)
{
    Tiempo temp;
    temp.minuto = minuto + h.minuto;
    temp.hora    = hora    + h.hora;
    if(temp.minuto >= 60) {
        temp.minuto -= 60;
        temp.hora++;
    }
    return temp;
}
```

```
void Tiempo::Mostrar()
{
    cout << hora << ":" << minuto << endl;
}

int main()
{
    Tiempo Ahora(12,24), T1(4,45);
    T1 = Ahora + T1;
    T1.Mostrar();
    (Ahora + Tiempo(4,45)).Mostrar();
    return 0;
}
```

Recibimos un parámetro por copia.
Hacemos un retorno por copia.

Si Tiempo tuviera punteros sería imprescindible el constructor copia y el operador asignación.

25.1 Repaso

Sobrecarga de operadores unitarios

```
class Tiempo
{
    ...
    Tiempo operator++();
    ...
};
Tiempo Tiempo::operator++()
{
    minuto++;
    while(minuto >= 60)
    {
        minuto -= 60;
        hora++;
    }
    return *this;
}
...
++T1;
```

No hay parámetro.

Actúa sobre “sí mismo”.

Retorno por valor.

Uso del puntero this.

25.2 Sobrecarga de Operadores

Diferentes formas de Operadores unitarios (++)

Vamos a trabajar con esta clase:

```
class Fraccional
{
private:
    int num_ ;
    int den_ ;
    void simplificar_( ) ;
public:
    Fraccional(int n=0, int d=1);
    Fraccional operator + ( Fraccional f );
    Fraccional operator - (Fraccional f);
    Fraccional operator * (Fraccional f);
    Fraccional operator / (Fraccional f);
    int numerador( );
    int denominador( );
    void numerador (int n);
    void denominador ( int d);
};
```

```
void main()
{
    Fraccional f1(1,2);
    Fraccional f2(2,1);
    Fraccional f3;

    f3 = f1 + f2 ;

    // Lo mismo que antes...
    f3 = f1.operator+( f2 );
    ....
    ....
}
```

Vamos a ver las diferencias entre estas tres declaraciones e implementaciones para seguir comprendiendo la POO

```
void operator++();
Fraccional operator++();
Fraccional& operator++();
```

25.2 Sobrecarga de Operadores

Operadores unitarios y valor de retorno(++)

¿Modifica el objeto receptor de la llamada?

```
void operator++();
```

SI

¿Devuelve algo?

NO

Problemas:

SI.

```
void Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
}
```

```
Fraccional f1(1,1),f2;  
++f1;
```

```
++(++f1);    // Nada que incrementar!!  
f2 = ++f1;   // Nada que asignar.
```

Si vamos a encadenar operaciones no podemos usar esta versión. En caso contrario está bien y de hecho lo estamos indicando a programadores usuario de nuestra clase (En Fraccional.h).

25.2 Sobrecarga de Operadores

Operadores unitarios y valor de retorno(++)

¿Modifica el objeto receptor de la llamada?

```
Fraccional operator++();
```

SI

¿Devuelve algo?

El objeto receptor incrementado.

Problemas:

SI.

```
Fraccional Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
    return *this;  
}
```

```
Fraccional f1(1,1),f2;  
++f1;  
f2 = ++f1;  
  
++(++f1);    // Error.
```

Si vamos a encadenar operaciones de incremento no podemos usar esta versión.

Veamos por qué....

25.2 Sobrecarga de Operadores

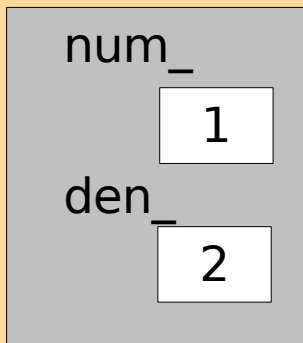
Operadores unitarios y valor de retorno(++)

```
Fraccional operator++();
```

```
Fraccional f1(1,2);  
++(++f1);
```

```
Fraccional Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
    return *this;  
}
```

◆ f1



25.2 Sobrecarga de Operadores

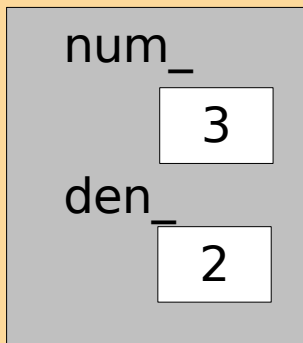
Operadores unitarios y valor de retorno(++)

```
Fraccional operator++();
```

```
Fraccional f1(1,2);  
++(++f1);
```

```
Fraccional Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
    return *this;  
}
```

◆ f1



25.2 Sobrecarga de Operadores

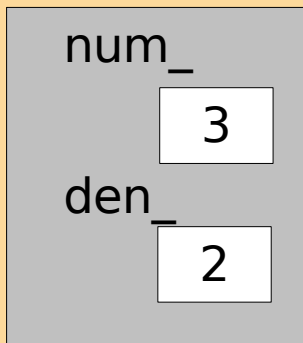
Operadores unitarios y valor de retorno(++)

```
Fraccional operator++();
```

```
Fraccional f1(1,2);  
++(++f1);
```

```
Fraccional Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica();  
    return *this;  
}
```

◆ f1



25.2 Sobrecarga de Operadores

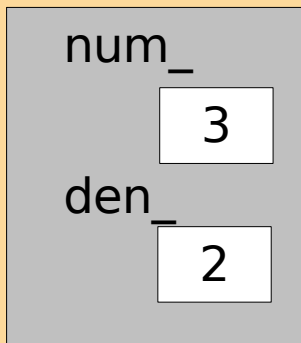
Operadores unitarios y valor de retorno(++)

```
Fraccional operator++();
```

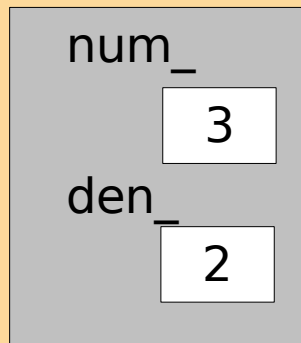
```
Fraccional f1(1,2);  
++(++f1);
```

```
Fraccional Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
    return *this;  
}
```

◆ f1



temp1



Devolución por valor.

Retornamos una copia del objeto.

25.2 Sobrecarga de Operadores

Operadores unitarios y valor de retorno(++)

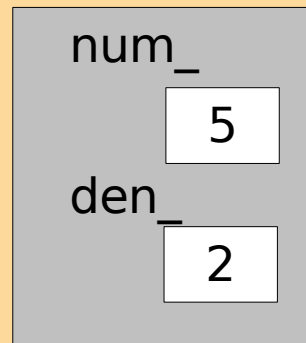
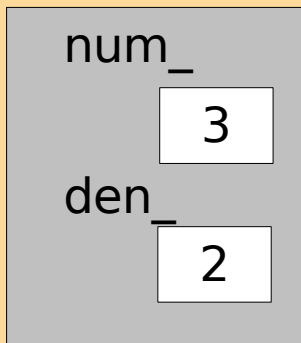
```
Fraccional operator++();
```

```
Fraccional f1(1,2);  
++(++f1);
```

```
Fraccional Fraccional::operator++( )  
{  
  num_ = num_ + den_;  
  simplifica() ;  
  return *this;  
}
```

f1

◆ temp1



El segundo “++” opera sobre la copia que devolvió la función en la llamada anterior.

25.2 Sobrecarga de Operadores

Operadores unitarios y valor de retorno(++)

```
Fraccional operator++();
```

```
Fraccional f1(1,2);  
++(++f1);
```

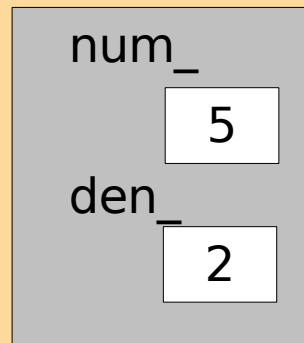
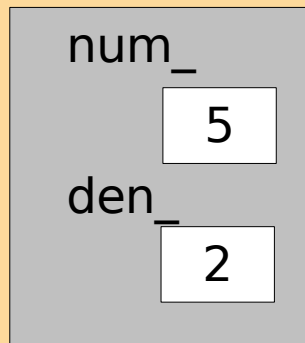
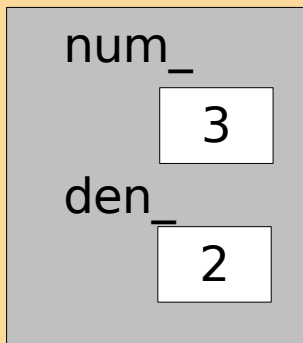
```
Fraccional Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica();  
    return *this;  
}
```

f1



temp1

temp2



Devolvemos por valor:
Otra Copia.

No funciona porque el segundo incremento actúa sobre la copia que devuelve el primero.

25.2 Sobrecarga de Operadores

Operadores unitarios y valor de retorno(++)

¿Modifica el objeto receptor de la llamada?

```
Fraccional& operator++();
```

SI

¿Devuelve algo?

El propio objeto receptor incrementado y no una copia.

Problemas:

NO.

```
Fraccional& Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
    return *this;  
}
```

```
Fraccional f1(1,1),f2;  
++f1;  
f2 = ++f1;  
++(++f1);
```

Podemos encadenar operaciones sin problemas.

Modifica el objeto receptor y lo devuelve como resultado. Como es una referencia estoy trabajando con mi dirección no con una copia.

25.2 Sobrecarga de Operadores

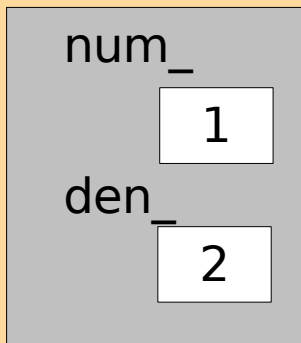
Operadores unitarios y valor de retorno(++)

```
Fraccional& operator++();
```

```
Fraccional f1(1,2),f2;  
f2 = ++f1;  
++(++f1);
```

```
Fraccional& Fraccional::operator++( )  
{  
    num_ = num_ + den_;  
    simplifica() ;  
    return *this;  
}
```

◆ f1



No hay objetos temporales.

Se trabaja siempre con f1.

25.2 Sobrecarga de Operadores

Sobrecarga: Operadores miembro – Operadores Globales

Podemos definir los operadores sobre una clase de dos formas.

```
class Complejo
{
    // Declaraciones de operadores binarios mediante funciones amigas
    // externas.
    friend ostream& operator<<(ostream& out, const Complejo& c);
    friend Complejo operator-(const Complejo& mi,const Complejo& md);
    friend Complejo operator*(const Complejo& mi,const Complejo& md);
    friend Complejo operator/(const Complejo& mi,const Complejo& md);
public:
    Complejo(int re=0, int im=0);
    // Funciones binarias miembro de la clase internas.
    Complejo& operator+= (const Complejo& c);
    Complejo operator+(const Complejo& md);
private:
    int re_;
    int im_;
}
```

25.2 Sobrecarga de Operadores

Sobrecarga: Operadores miembro – Operadores Globales

```
Complejo& Complejo::operator+= (const Complejo& c)
{
    re_ += c.re_;
    im_ += c.im_;
    return *this;    // Me devuelvo a mi mismo.
}
```

```
Complejo Complejo::operator+(const Complejo& md)
{
    return Complejo(re_ + md.re_ , im_ + md.im_);
}
```

....

```
Complejo operator-(const Complejo& mi, const Complejo& md)
{
    return Complejo(mi.re_ - md.re_ , mi.im_ - md.im_);
}
```

25.2 Sobrecarga de Operadores

Sobrecarga: Operadores miembro – Operadores Globales

Como hemos visto podemos sobrecargar los operadores sobre una clase como:

- Funciones miembro.
- Funciones globales.

Un operador de tipo función miembro requiere que **el operando de la izquierda sea un objeto de la clase.**

Un operador de tipo función global **permite los mismos tipos de conversión** para ambos operandos.

25.2 Sobrecarga de Operadores

Sobrecarga: Operadores miembro – Operadores Globales

```
#include "Complejo.h"
int main()
{
    Complejo c1(3,3);
    Complejo c2(4,5);

    Complejo c3 = c1 + c2; // Sin problemas.
    Complejo c4 = c3 + 5 ; // Conversión del operador de la derecha.
    Complejo c5 = 8 - c4; // Conversión del operador de la izquierda.

    Complejo c6 = 1 + c4; // Error. Op. derecha tiene que ser Complejo.

    return 0;
}
```

En general es más útil definir los operadores binarios como funciones amigas externas globales y los unitarios como funciones miembro.