

Métodos operadores

1. Sobrecarga de operadores
2. Métodos operadores unarios
Operador de incremento (prefijo)
3. Devolución de objetos en los métodos.
Retorno por referencia

1

Métodos operadores

Sobrecarga de operadores

Los **operadores** aceptan uno o varios operandos de tipo específico, produciendo y/o modificando un valor de acuerdo con ciertas reglas.

C++ permite redefinir la mayor parte de los operadores, es decir, permite que los operandos no sean tipos básicos, sino tipos definidos por el programador y seguir otro comportamiento.

Los operadores conservan el sentido y comportamiento originales cuando se usan con los tipos básicos.



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

2

Operadores sobrecargables

El lenguaje C++ permite redefinir la funcionalidad de los siguientes operadores:

```
+ - * / % ^ =  
+= -= *= /= %=  
== != <= >= < >  
&& || ++ -- -> *  
new new[] delete delete[]
```

Hay más ...

Operadores NO sobrecargables

```
. Operador punto  
:: Operador de acceso a ámbito  
# y ## Directivas de preprocesado
```

Hay más ...

Métodos operadores binarios

operador + : para sumar fracciones,
operador - : para restar fracciones,
operador * : para multiplicar fracciones,
operador / : para dividir fracciones

Paso de parámetro **por copia**

Retorno **por copia**

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

Prototipo

```
Fraccional Fraccional :: operator + (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador + denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

Métodos operadores binarios

```
void main()
{
    Fraccional f1(1,2), f2(2,1), f3;
    f3 = f1 + f2 ;
    ...
    ...
}
```



```
...
f3 = f1.operator + ( f2 ) ;
...
```

```
class Fraccional
{
private:
    int numerador ;
    int denominador ;
    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 ac_numerador( );
    int ac_denominador( );
    void mu_numerador (int n);
    void mu_denominador ( int d);
};
```

Métodos operadores unarios

Existen métodos que modifican el objeto receptor del mensaje.

operador ++
operador +=
operador *=

operador --
operador -=
operador /=

Vamos a jugar con el operador incremento prefijo (**operador ++**)

1 Modifica el objeto receptor pero no devuelve nada.

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

Prototipo

```
void Fraccional :: operator ++ ( )
{
    numerador = numerador + denominador ;
    simplifica() ;
}
```

Comportamiento

1

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

```
void Fraccional :: operator ++ ( )  
{  
    numerador = numerador + denominador ;  
    simplifica() ;  
}
```

Modifica el objeto receptor pero no devuelve nada.

Modifica adecuadamente el objeto receptor

Se puede utilizar para incrementar un Fraccional

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

ERROR
Nada que incrementar

Se impide encadenar operaciones

```
++(++f1) ;  
f2 = ++f1 ;  
...
```

ERROR
Nada que asignar

2 Método operador que devuelve un nuevo objeto

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

```
Fraccional Fraccional :: operator ++ ( )  
{  
    numerador = numerador + denominador ;  
    simplifica() ;  
    return *this ;  
}
```

El objeto devuelto es igual al objeto receptor incrementado

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

Seguimos teniendo un problema, **el método modifica el objeto y devuelve una copia del objeto receptor.**

Modifica adecuadamente el objeto receptor

Se puede utilizar para incrementar un Fraccional y asignar el resultado a otro Fraccional

```
++(++f1) ;
```

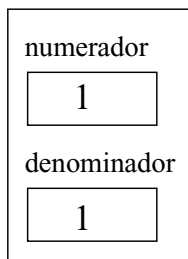
ERROR

Comportamiento

2 Método operador que devuelve un nuevo objeto

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

f1



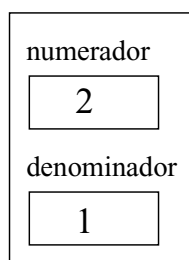
```
Fraccional Fraccional :: operator ++ ( )  
{  
  numerador = numerador + denominador ;  
  simplifica() ;  
  return *this ;  
}
```

Comportamiento

2 Método operador que devuelve un nuevo objeto

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

f1



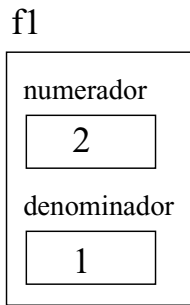
```
Fraccional Fraccional :: operator ++ ( )  
{  
  numerador = numerador + denominador ;  
  simplifica() ;  
  return *this ;  
}
```

Comportamiento

2 Método operador que devuelve un nuevo objeto

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

```
Fraccional Fraccional :: operator ++ ( )  
{  
  numerador = numerador + denominador;  
  simplifica();  
  return *this;  
}
```

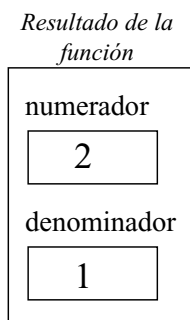
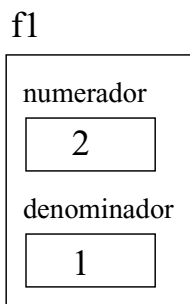


Comportamiento

2 Método operador que devuelve un nuevo objeto

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

```
Fraccional Fraccional :: operator ++ ( )  
{  
  numerador = numerador + denominador;  
  simplifica();  
  return *this;  
}
```



Devolución por valor
Se devuelve una copia del objeto receptor

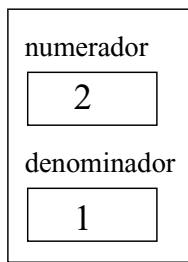
Comportamiento

2 Método operador que devuelve un nuevo objeto

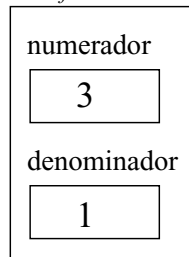
```
...  
Fraccional f1(1, 1);  
++(++f1);
```

```
Fraccional Fraccional :: operator ++ ( )  
{  
    numerador = numerador + denominador ;  
    simplifica() ;  
    return *this ;  
}
```

f1



Resultado de la función



tmp1

El segundo incremento se realiza sobre la copia que devuelve el primero.

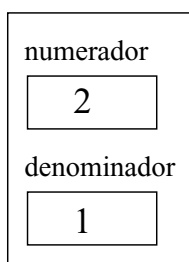
Comportamiento

2 Método operador que devuelve un nuevo objeto

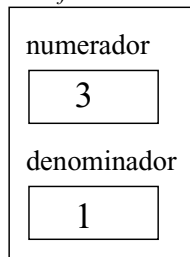
```
...  
Fraccional f1(1, 1);  
++(++f1);
```

```
Fraccional Fraccional :: operator ++ ( )  
{  
    numerador = numerador + denominador ;  
    simplifica() ;  
    return *this ;  
}
```

f1

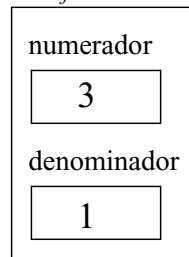


Resultado de la función



tmp1

Resultado de la función



tmp2

Devolución por valor
Se devuelve una copia del objeto receptor, es decir, se devuelve una copia de la copia.

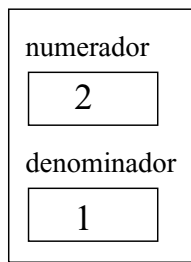
Comportamiento

2 Método operador que devuelve un nuevo objeto

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

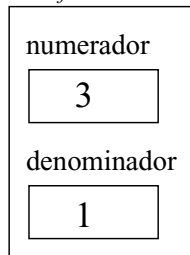
```
Fraccional Fraccional :: operator ++ ( )
{
    numerador = numerador + denominador ;
    simplifica() ;
    return *this ;
}
```

f1

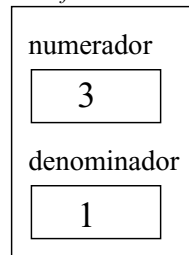


tmp1

Resultado de la función



Resultado de la función



tmp2

No se obtiene el resultado deseado, ya que el segundo incremento se realiza sobre la copia que devuelve el primero.

3 Método operador que devuelve el objeto receptor

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

Se devuelve el propio objeto receptor, no una copia.

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

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

Sin problemas

Modifica adecuadamente el objeto receptor y se devuelve el mismo como resultado de la ejecución del método, de forma que cualquier nueva operación se realiza sobre el resultado de ésta operación, se efectúa sobre ese mismo objeto.

Retorno de funciones

◆ Retorno estándar

En un retorno estándar, el retorno de la función es un objeto temporal que se crea para almacenar el valor que se devuelve. La existencia de éstos objetos temporales está ligada a la expresión donde se realiza la invocación.

◆ Retorno de referencias

El modificador **&** junto con el tipo que devuelve la función, indica éste tipo de retorno.

En un retorno de referencias, no se crea ningún objeto temporal, el valor que se devuelve se almacena en el objeto que realiza la invocación.

El operador +=

Devuelve el propio objeto receptor del mensaje

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

```
Fraccional & Fraccional :: operator += (Fraccional f )  
{  
    numerador = numerador * f.denominador +  
                denominador * f.numerador;  
    denominador = denominador * f.denominador;  
    simplifica();  
    return *this;
```

```
void main()  
{  
    Fraccional f1(1,2), f2(2,1);  
    f1 += f2 ;  
    ...
```

```
...  
    f1. operator += ( f2 ) ;  
    ...
```