

# SOBRECARGA DE OPERADORES

- \*Operadores=funciones  
→sobrecarga
- \* This; const; &

## *Sobrecarga de operadores.* El puntero **this** (I)

- Contiene la dirección de memoria del propio objeto sobre el que se aplica el método.
- Parámetro implícito en cualquier función miembro (solo se puede usar ahí).
- Ejemplo:

Ejemplo 5.1

```
1 TCoordenada::TCoordenada(int a, int b, int c) {  
2     x = a;  
3     this->y = b;  
4     (*this).z = c;  
5 }
```

**Se puede desreferenciar  
como un puntero normal**

## *Sobrecarga de operadores.* El puntero **this** (II)

- Usos:
  - Como retorno de una instancia del propio objeto, en una función miembro.
  - Para evitar que un objeto se asigne a sí mismo.
  - Permitir la llamada en cascada de varias funciones miembro sobre un mismo objeto.
- Es de sólo lectura :

## *Sobrecarga de operadores.* El puntero **this** (III)

### Ejemplo 5.2

```
1
2 TCoordenada::TCoordenada(const TCoordenada &c) {
3     x = c.x;
4     y = c.y;
5     z = c.z;
6     this = &c; ←
7 }
```

**Intentamos modificar el  
contenido de this y al  
compilar ...**

### Salida ejemplo 5.2

```
1 tcoordenada.cc: In copy constructor 'TCoordenada::TCoordenada(
2     const TCoordenada&)':
3 tcoordenada.cc:17: non-lvalue in assignment
```

*Sobrecarga de operadores.*

## Modificador **const** (I)

- Define valores que no se pueden modificar.
- Usos:
  - **Ctes. simbólicas** numéricas o literales. Se inclinan en su declaración.
  - En **argumentos** de una función indicando que no va a cambiar.
  - En **los métodos** de una clase para indicar que no va a modificar el objeto, **solo consulta**:  
*<tipo> <nombreFuncion (parametros)> const*

## *Sobrecarga de operadores.*

# Modificador **const** (II)

```
1 // En .h
2 void algo() const;
3
4 // En .cc
5 void
6 UnaClase::algo() const
7 {
8     ...
9 }
```

- Un objeto constante → Sólo puede invocar a funciones constantes.
- Un objeto no constante → Puede invocar a funciones constantes y no constantes

## *Sobrecarga de operadores.*

# Modificador **const** (III)

```
1 // En el .h
2 class UnaClase
3 {
4     public:
5         UnaClase();
6         void Modifica() const;
7
8     private:
9         int a;
10    };
11
12 // En el .cc
```

```
13 UnaClase::UnaClase()
14 {
15     a = 0;
16 }
17
18 void
19 UnaClase::Modifica() const
20 {
21     a++;
22 }
```

Salida ejemplo 5.2

```
1 unaclase.cc: In member function ‘void UnaClase::Modifica() const’:
2 unaclase.cc:23: increment of data-member ‘UnaClase::a’ in read-only
3                     structure
```

## *Sobrecarga de operadores.* **Paso por referencia (I)**

- Es un alias de una variable u objeto.
- Uso:  
En argumentos y valores de retorno de funciones y operadores sobrecargados.
- Iniciador:  
Ha de ser un valor del que se pueda obtener su dirección.  
Ejemplo:  
***int & a = 1 //Error: no se puede tener la referencia***

## *Sobrecarga de operadores.* **Paso por referencia (II)**

Salida ejemplo 5.2

```
1 int.cc: In function 'int main()':  
2 int.cc:20: initialization of non-const reference type 'int&' from rvalue  
3     of type 'int'
```

- Solución:

***const int & a = 1;***

Se crea una variable temporal invisible que almacena el iniciador de una referencia y perdura hasta el final del ámbito de la misma.

## *Sobrecarga de operadores.* Paso por referencia (III)

- Con el paso por parámetros en una función ocurre lo mismo:

Ejemplo 5.3

```
1 #include <iostream>
2
3 using namespace std;
4
5 int
6 algo()
7 {
8     int a;
9     // ...
10    return a;
11 }
```

```
13 void
14 otra(int& a)
15 {
16     // ...
17 }
18
19 int
20 main(void)
21 {
22     // Error
23     otra(algo());
24 }
```

**Espera un 'int &' ...  
... y recibe un 'int'**

## *Sobrecarga de operadores.* Paso por referencia (IV)

Salida ejemplo 5.3

```
1 int.cc: In function 'int main()':  
2 int.cc:23: could not convert 'algo()()' to 'int&'  
3 int.cc:15: in passing argument 1 of 'void otra(int&)'
```

```
1 void  
2 otra(const int& a)  
3 {  
4     ...  
5 }
```

**Solución**

*Sobrecarga de operadores.*

## Sobrecarga de operadores (I)

- Operadores en C++:
  - NO permite crear nuevos
  - Si permite sobrecargar los ya existentes
  - Algunos operadores ya sobrecargados (+, -, \*, etc)
- ¿Cómo se sobrecagan?  
***operator <operador> (parametros)***

```
1 | ... operator+(...) {  
2 | ...  
3 | }  
4 |
```

```
5 | ... operator=(...) {  
6 | ...  
7 | }
```

*Sobrecarga de operadores.*

## Sobrecarga de operadores (II)

```
1 | TCoordenada a, b, c;  
2 |  
3 | a = b + c;
```

```
4 | // Equivale a:  
5 | a.operator=(b.operator+(c));
```

El compilador invoca a la función  
asociada al operador

*Sobrecarga de operadores.*

## Restricciones al sobrecargar un operador (I)

- En un operador no se puede modificar:
  - La precedencia → en tal caso, uso de '( )'
  - La asociatividad
  - El número de operandos
  - El comportamiento con tipos predefinidos del lenguaje
- Los operadores `+=`, `==`, `=+`, `=-` se tienen que sobrecargar independientemente.

*Sobrecarga de operadores.*

## Restricciones al sobrecargar un operador (II)

- Operadores que se pueden sobrecargar:

+	-	*	/	%	^	&	
~	!	=	<	>	+ =	- =	* =
/ =	% =	^ =	& =	=	<<	>>	>> =
<< =	==	!=	< =	> =	&&		++
--	->*	,	->	[]	()	new	delete
new []	delete []						

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

.\* :: ?:

*Sobrecarga de operadores.*

## ¿Función miembro o función no miembro?

- Sobrecarga operadores (), [ ], -> y de asignación → Siempre como funciones miembro.
- El resto se puede hacer como miembro o no.
- En funciones miembro:
  - operando izquierda siempre es objeto de la clase  
*TCoordenada a; a + 3; // a.operator+(3);*
- En funciones no miembro:
  - Operando izquierda puede ser distinto de la clase.  
*Tcoordenada a; 3 + a; // operator+(3, a)*
  - Si necesitan acceso a parte privada de la clase → funciones amigas (friend)

## *Sobrecarga de operadores.* Consejos (I).

- Versión friend → Un parámetro más:  
*TCoordenada a; 3 + a; // operator+(3, a);*
- Respeto significado original del operador
- El operando de la izquierda (objeto):
  - Cuando se modifica (**a = b**)
    - Se almacena el resultado en objeto
    - Se devuelve el propio objeto y por referencia (más rápido)
  - Cuando no se modifica (**a + b**)
    - Se crea objeto temporal que almacena el resultado
    - Se devuelve el objeto temporal siempre por valor.
- El parámetro de la función → por referencia

## Sobrecarga de operadores. Operador asignación (I)

- Sin sobrecarga en una clase: copia bit a bit. → Mismo problema que con el constructor copia predeterminado.
- Consejo: sobrecargar siempre el operador =

```
1 // En el .h
2 TCoordenada& operator=(TCoordenada &);

3

4
5 // En el .cc
6 // Asignación: a = b
7 TCoordenada&
8 TCoordenada::operator=(TCoordenada &op2) {
9     (*this).~TCoordenada();
10
11     x = op2.x;
12     y = op2.y;
13     z = op2.z;
14
15     return *this;
16 }
```

Ejemplo 5.4

**Se pasa por referencia  
(faltaría pasarlo como const)**

**Se liberan primero todos  
los recursos**

## *Sobrecarga de operadores.* Operador asignación (II)

Ejemplo 5.5

```
1 #include <iostream>
2
3 using namespace std;
4
5 #include "tcoordenada.h"
6
7 int main(void) {
8     TCoordenada p1(1, 2, 3), p2(4, 5, 6);
9
10    p1.Imprimir();
11    cout << endl;
12    p2.Imprimir();
13    cout << endl;
```

```
14    cout << "p1 = p2" << endl;
15    p1 = p2;
16
17    p1.Imprimir();
18    cout << endl;
19    p2.Imprimir();
20    cout << endl;
21
22
23    return 0;
24 }
```

(1, 2, 3)  
(4, 5, 6)  
p1 = p2  
(4, 5, 6)  
(4, 5, 6)

## *Sobrecarga de operadores.* Operador asignación (III)

- ¿Y si asignamos a un objeto, el propio objeto ( $a=a$ )?

Ejemplo 5.6

```
1 #include <iostream>
2
3 using namespace std;
4
5 #include "tcoordenada.h"
6
7 int main(void) {
8     TCoordenada p1(1, 2, 3);
9 }
```

```
10    p1.Imprimir();
11    cout << endl;
12
13    cout << "p1 = p1" << endl;
14    p1 = p1;
15
16    p1.Imprimir();
17    cout << endl;
18
19    return 0;
20 }
```

Salida ejemplo 5.6

```
1 (1, 2, 3)
2 p1 = p1
3 (0, 0, 0)
```

¿ ?

## *Sobrecarga de operadores.* Operador asignación (IV)

- Corrección del error:

Ejemplo 5.7

```
1 TCoordenada&
2 TCoordenada::operator=(TCoordenada &op2) {
3     if(this != &op2) ←
4     {
5         (*this).~TCoordenada();
6
7         x = op2.x;
8         y = op2.y;
9         z = op2.z;
10    }
11
12    return *this;
13 }
```

**Comprobamos que no sean  
el mismo objeto**

*Sobrecarga de operadores.*

## Constructor de copia y operador asignación (I)

- Constructor de copia y operador de asignación comparten muchas líneas de código.
- La parte común se pone en una función auxiliar que se declara en la parte privada.
- Se invocará a la misma, desde el constructor copia y el operador de asignación.

*Sobrecarga de operadores.*

## Constructor de copia y operador asignación (II)

Ejemplo 5.10

```
1 // Constructor de copia
2 TCoordenada::TCoordenada(const TCoordenada &c) {
3     Copia(c);
4 }
5
6 // Sobrecarga del operador asignación
7 TCoordenada&
8 TCoordenada::operator=(TCoordenada &op2) {
9     if(this != &op2)
10    {
11        (*this).~TCoordenada(),
12        Copia(op2);
13    }
14
15    return *this;
16 }
17
18 void
19 TCoordenada::Copia(const TCoordenada &c) {
20     x = c.x;
21     y = c.y;
22     z = c.z;
23 }
```

**Invocaciones a la función auxiliar**

**Implementación función auxiliar  
(se declara en la parte privada  
de la clase)**

# *Sobrecarga de operadores.* Operadores aritméticos (I)

- No suelen modificar el objeto sobre el que actúan → necesidad de objeto temporal.

Ejemplo 5.11

```
1 // En el .h
2 TCoordenada operator+(TCoordenada &);

3

4

5 // En el .cc
6 // Suma: a + b
7 TCoordenada
8 TCoordenada::operator+(TCoordenada &op2) {
9     TCoordenada temp;
10
11     temp.x = x + op2.x;
12     temp.y = y + op2.y;
13     temp.z = z + op2.z;
14
15     return temp;
16 }
```

Ejemplo 5.12

```
1 // En el .h
2 TCoordenada operator-(TCoordenada &);

3

4

5 // En el .cc
6 // Resta: a - b
7 TCoordenada
8 TCoordenada::operator-(TCoordenada &op2) {
9     TCoordenada temp;
10
11     temp.x = x - op2.x;
12     temp.y = y - op2.y;
13     temp.z = z - op2.z;
14
15     return temp;
16 }
```

# Sobrecarga de operadores. Operadores aritméticos (II)

Ejemplo 5.13

```
1 #include <iostream>
2
3 using namespace std;
4
5 #include "tcoordenada.h"
6
7 int main(void) {
8     TCoordenada p1(1, 2, 3), p2(4, 5, 6), p3(7, 8, 9);
9
10    p1.Imprimir();
11    cout << endl;
12    p2.Imprimir();
13    cout << endl;
14    p3.Imprimir();
15    cout << endl;
16
17    cout << "p1 = p2 + p3" << endl;
18    p1 = p2 + p3;
19
20    p1.Imprimir();
21    cout << endl;
22    p2.Imprimir();
23    cout << endl;
24    p3.Imprimir();
25    cout << endl;
26
27    cout << "p1 = p2 - p3" << endl;
28    p1 = p2 - p3;
29
30    p1.Imprimir();
31    cout << endl;
32    p2.Imprimir();
33    cout << endl;
34    p3.Imprimir();
35    cout << endl;
36
37    return 0;
38 }
```

Salida ejemplo 5.13

```
1 main.cc: In function 'int main()':
2 main.cc:18: no match for 'TCoordenada& = TCoordenada' operator
3 tcoordenada.h:12: candidates are: TCoordenada&
4           TCoordenada::operator=(TCoordenada&)
5 main.cc:28: no match for 'TCoordenada& = TCoordenada' operator
6 tcoordenada.h:12: candidates are: TCoordenada&
7           TCoordenada::operator=(TCoordenada&)
```

# Sobrecarga de operadores. Operadores aritméticos (III)

- Consejo: Siempre que se pueda → parámetros de las funciones como **const**

Ejemplo 5.14

```
1 class TCoordenada {
2     public:
3         TCoordenada();
4         TCoordenada(int, int, int);
5         TCoordenada(const TCoordenada &);
6         ~TCoordenada();
7
8         TCoordenada& operator=(const TCoordenada &); operator=
9         TCoordenada operator+(const TCoordenada &); operator+
10        TCoordenada operator-(const TCoordenada &); operator-
11
12        void setX(int);
13        void setY(int);
14        void setZ(int);
15
16        int getX(void);
17        int getY(void);
18        int getZ(void);
19
20        void Imprimir(void);
21
22    private:
23        int x, y, z;
24 }
```

- No olvidar nunca que:  
**p1 = p2 + p3**

**Invoca a:**

**p1.operator=(p2.operator+(p3));**

**p1 = p2 - p3**

**Invoca a:**

**p1.operator=(p2.operator-(p3));**

## *Sobrecarga de operadores.*

# Operadores de incremento y decremento

- Operador `++` : dos formas, **pre** y **post** incremento. ¿Cómo distinguirlas?
- La versión post recibe un parámetro entero irrelevante.

Ejemplo 5.15

```
1 // En el .h
2 TCoordenada& operator++(void);

3

4
5 // En el .cc
6 // Preincremento: ++a
7 TCoordenada&
8 TCoordenada::operator++(void) {
9     x++;
10    y++;
11    z++;
12
13    return *this;
14 }
```

Ejemplo 5.16

```
1 // En el .h
2 TCoordenada operator++(int);
3
4
5 // En el .cc
6 // Postincremento: a++
7 TCoordenada
8 TCoordenada::operator++(int op2) {
9     TCoordenada temp(*this);
10
11    x++;
12    y++;
13    z++;
14
15    return temp;
16 }
```

# Sobrecarga de operadores. Operadores abreviados.

- Combinación asignación con operador aritmético ( $+=$ ,  $-=$ ,  $*=$ , etc.)

Ejemplo 5.17

```
1 // En el .h
2 TCoordenada& operator+=(const TCoordenada &);
3 TCoordenada& operator-=(const TCoordenada &);

4
5
6 // En el .cc
7 // a += b
8 TCoordenada&
9 TCoordenada::operator+=(const TCoordenada &op2) {
10    x += op2.x;
11    y += op2.y;
12    z += op2.z;
13
14    return *this;
15 }
```

```
16
17 // a -= b
18 TCoordenada&
19 TCoordenada::operator-=(const TCoordenada &op2) {
20    x -= op2.x;
21    y -= op2.y;
22    z -= op2.z;
23
24    return *this;
25 }
```

El propio objeto se modifica

# *Sobrecarga de operadores.*

## Operadores de comparación

- Devuelven un valor booleano (==, !=, <, >, <=, >=)

Ejemplo 5.18

```
1 // En el .h
2 bool operator==(const TCoordenada &);
3
4
5 // En el .cc
6 // a == b
7 bool
8 TCoordenada::operator==(const TCoordenada &op2) {
9     bool temp;
10
11     temp = (x==op2.x && y==op2.y && z==op2.z) ? true : false;
12
13     return temp;
14 }
```

Ejemplo 5.19

```
1 // En el .h
2 bool operator!=(const TCoordenada &);
3
4
5 // En el .cc
6 // a != b
7 bool
8 TCoordenada::operator!=(const TCoordenada &op2) {
9     return !(*this == op2);
10 }
```

Al ser operadores complementarios  
reutilizamos el código

# Sobrecarga de operadores. Operadores de entrada y salida

- `cin << , cout >>` : El operando izquierdo es objeto clase diferente (`istream` y `ostream`) → Obligado a usar **funciones amigas**.

Ejemplo 5.20

```
1 // En el .h
2 friend istream& operator>>(istream &, TCoordenada &);
3
4
5 // En el .cc
6 // cin >> a
7 istream&
8 operator>>(istream &s, TCoordenada &obj) {
9     cout << "Introducir coordenada x:";
10    s >> obj.x;
11
12    cout << "Introducir coordenada y:";
13    s >> obj.y;
14
15    cout << "Introducir coordenada z:";
16    s >> obj.z;
17
18    return s;
19 }
```

Ejemplo 5.21

```
1 // En el .h
2 friend ostream& operator<<(ostream &, const TCoordenada &);
3
4
5 // En el .cc
6 // cout << a
7 ostream&
8 operator<<(ostream &s, const TCoordenada &obj) {
9     s << "(" << obj.x << ", ";
10    s << obj.y << ", ";
11    s << obj.z << ")";
12
13    return s;
14 }
```

Para que funcionen expresiones del tipo:

**TCoordenada a, b;**

**cout << "Objeto a: " << a << " Objeto b: " << b << endl;**

## *Sobrecarga de operadores.* **Operador corchete (I)**

- Es especial. Se usa tanto para leer y otras para escribir (modificar) el objeto → Dos sobrecargas:
  - Permite la escritura:
    - Sobre objetos no constantes
    - Se devuelve una referencia al valor a leer/escribir
  - Permite sólo la lectura:
    - Sobre objetos constantes.
    - Se devuelve el valor a leer
- Debe comprobar que la posición es correcta. ¿Cómo?:
  - No comprobar nada y devolver contenido.
  - Mensaje de error y detener ejecución.
  - Devolver un valor de error.

## *Sobrecarga de operadores.* **Operador corchete (II)**

- Se suele adoptar la última opción:  
En el caso de la sobrecarga de lectura/escritura:  
necesidad de una variable error en la parte privada.
- Ejemplo con TCoordenada:  
En tcoordenada.h
  - En la parte pública:  
*int& operator[ ] (char);*  
*int operator[ ] (char) const;*
  - En la parte privada:  
*int error;*

## *Sobrecarga de operadores.* Operador corchete (III)

- En tcoordenada.cc:

\_\_\_\_\_ Ejemplo 5.24 \_\_\_\_\_

```
1 int&
2 TCoordenada::operator[](char c) {
3     if(c == 'x' || c == 'X')
4         return x;
5     if(c == 'y' || c == 'Y')
6         return y;
7     if(c == 'z' || c == 'Z')
8         return z;
9
10    error = 0;
11    return error;
12 }
```

```
13
14 int
15 TCoordenada::operator[](char c) const {
16     if(c == 'x' || c == 'X')
17         return x;
18     if(c == 'y' || c == 'Y')
19         return y;
20     if(c == 'z' || c == 'Z')
21         return z;
22
23     return 0;
24 }
```

- Sustituyen a setX(int), setY(int), setZ(int) y getX(),  
getY(), getZ()

# *Sobrecarga de operadores.*

## Operador corchete (IV)

Ejemplo 5.25

```
1 #include <iostream>
2
3 using namespace std;
4
5 #include "tcoordenada.h"
6
7 int main(void) {
8     TCoordenada p1(1, 2, 3);
9
10    cout << p1 << endl;
11
12    p1['x'] = 4;
13    p1['y'] = 5;
14    p1['z'] = 6;
15
16    cout << p1 << endl;
17
18    cout << "x: " << p1['x'] << endl;
19    cout << "y: " << p1['y'] << endl;
20    cout << "z: " << p1['z'] << endl;
21
22    return 0;
23 }
```

```
1 (1, 2, 3)
2 (4, 5, 6)
3 x: 4
4 y: 5
5 z: 6
```