

# Constructores

Los constructores son funciones miembro especiales que sirven para inicializar un objeto de una determinada clase al mismo tiempo que se declara.

Los constructores tienen el mismo nombre que la clase, no retornan ningún valor y no pueden ser heredados. Además deben ser públicos, no tendría ningún sentido declarar un constructor como privado, ya que siempre se usan desde el exterior de la clase, ni tampoco como protegido, ya que no puede ser heredado.

Añadamos un constructor a nuestra clase pareja:

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

class pareja {
public:
    // Constructor
    pareja(int a2, int b2);
    // Funciones miembro de la clase "pareja"
    void Lee(int &a2, int &b2);
    void Guarda(int a2, int b2);
private:
    // Datos miembro de la clase "pareja"
    int a, b;
public:
};

pareja::pareja(int a2, int b2) {
    a = a2;
    b = b2;
}

void pareja::Lee(int &a2, int &b2) {
    a2 = a;
    b2 = b;
}

void pareja::Guarda(int a2, int b2) {
    a = a2;
    b = b2;
}

void main() {
    pareja parl(12, 32);
    int x, y;

    parl.Lee(x, y);
    cout << "Valor de parl.a: " << x << endl;
    cout << "Valor de parl.b: " << y << endl;

    cin.get();}
```

Si una clase posee constructor, será llamado siempre que se declare un objeto de esa clase, y si requiere argumentos, es obligatorio suministrarlos.

Por ejemplo, las siguientes declaraciones son ilegales:

```
pareja par1;  
pareja par1();
```

La primera porque el constructor de "pareja" requiere dos parámetros, y no se suministran.

La segunda es ilegal por otro motivo más complejo. Aunque existiese un constructor sin parámetros, no se debe usar esta forma para declarar el objeto, ya que el compilador lo considera como la declaración de un prototipo de una función que devuelve un objeto de tipo "pareja" y no admite parámetros. Cuando se use un constructor sin parámetros para declarar un objeto no se deben escribir los paréntesis.

Y las siguientes declaraciones son válidas:

```
pareja par1(12,43);  
pareja par2(45,34);
```

Cuando no especifiquemos un constructor para una clase, el compilador crea uno por defecto sin argumentos. Por eso el ejemplo del capítulo anterior funcionaba correctamente. Cuando se crean objetos locales, los datos miembros no se inicializarían, contendrían la "basura" que hubiese en la memoria asignada al objeto. Si se trata de objetos globales, los datos miembros se inicializan a cero.

Para declarar objetos usando el constructor por defecto o un constructor que hayamos declarado sin parámetros no se debe usar el paréntesis:

```
pareja par2();
```

Se trata de un error frecuente cuando se empiezan a usar clases, lo correcto es declarar el objeto sin usar los paréntesis

```
pareja par2;
```

## Inicialización de objetos:

Hay un modo simplificado de inicializar los datos miembros de los objetos en los constructores.

Se basa en la idea de que en C++ todo son objetos, incluso las variables de tipos básicos como int, char o float.

Según eso, cualquier variable (u objeto) tiene un constructor por defecto, incluso aquellos que son de un tipo básico.

Sólo los constructores admiten inicializadores. Cada inicializador consiste en el nombre de la variable miembro a inicializar, seguida de la expresión que se usará para inicializarla entre paréntesis. Los inicializadores se añadirán a continuación del paréntesis cerrado que encierra a los parámetros del constructor, antes del cuerpo del constructor y separado del paréntesis por dos puntos ":".

Por ejemplo, en el caso anterior de la clase "pareja":

```
pareja::pareja(int a2, int b2) {  
    a = a2;  
    b = b2;  
}
```

Podemos sustituir el constructor por:

```
pareja::pareja(int a2, int b2) : a(a2), b(b2) {}
```

Por supuesto, también pueden usarse inicializadores en línea, dentro de la declaración de la clase.

Ciertos miembros es obligatorio inicializarlos, ya que no pueden ser asignados, por ejemplo las constantes o las referencias. Es muy recomendable usar la inicialización siempre que sea posible en lugar de asignaciones, ya que se desde el punto de vista de C++ es mucho más seguro.

Veremos más sobre este tema cuando veamos ejemplos de clases que tienen como miembros objetos de otras clases.

## Sobrecarga de constructores:

Además, también pueden definirse varios constructores para cada clase, es decir, la función constructor puede sobrecargarse. La única limitación es que no pueden declararse varios constructores con el mismo número y el mismo tipo de argumentos.

Por ejemplo, añadiremos un constructor adicional a la clase "pareja" que simule el constructor por defecto:

```
class pareja {
public:
    // Constructor
    pareja(int a2, int b2) : a(a2), b(b2) {}
    pareja() : a(0), b(0) {}
    // Funciones miembro de la clase "pareja"
    void Lee(int &a2, int &b2);
    void Guarda(int a2, int b2);
private:
    // Datos miembro de la clase "pareja"
    int a, b;
public:
};
```

De este modo podemos declarar objetos de la clase pareja especificando los dos argumentos o ninguno de ellos, en este último caso se inicializarán los datos miembros con ceros.

## Constructores con argumentos por defecto:

También pueden asignarse valores por defecto a los argumentos del constructor, de este modo reduciremos el número de constructores necesarios.

Para resolver el ejemplo anterior sin sobrecargar el constructor suministraremos valores por defecto nulos a ambos parámetros:

```
class pareja {
public:
    // Constructor
    pareja(int a2=0, int b2=0) : a(a2), b(b2) {}
    // Funciones miembro de la clase "pareja"
    void Lee(int &a2, int &b2);
    void Guarda(int a2, int b2);
private:
    // Datos miembro de la clase "pareja"
    int a, b;
public:
};
```

## Asignación de objetos:

Probablemente ya lo imaginas, pero la asignación de objetos también está permitida. Y además funciona como se supone que debe hacerlo, asignando los valores de los datos miembros.

Con la definición de la clase del último ejemplo podemos hacer lo que se ilustra en el siguiente:

```
void main() {
    pareja par1(12, 32), par2;
    int x, y;

    par2 = par1;
    par2.Lee(x, y);
    cout << "Valor de par2.a: " << x << endl;
    cout << "Valor de par2.b: " << y << endl;

    cin.get();}
```

La línea "par2 = par1;" copia los valores de los datos miembros de par1 en par2.

## Constructor copia:

Un constructor de este tipo crea un objeto a partir de otro objeto existente. Estos constructores sólo tienen un argumento, que es una referencia a un objeto de su misma clase. En general, los constructores copia tienen la siguiente forma para sus prototipos:

```
tipo_clase::tipo_clase(const tipo_clase &obj);
```

De nuevo ilustraremos esto con un ejemplo y usaremos también "pareja":

```
class pareja {
public:
    // Constructor
    pareja(int a2=0, int b2=0) : a(a2), b(b2) {}
    // Constructor copia:
    pareja(const pareja &p);

    // Funciones miembro de la clase "pareja"
    void Lee(int &a2, int &b2);
    void Guarda(int a2, int b2);
private:
    // Datos miembro de la clase "pareja"
    int a, b;
public:
};

pareja::pareja(const pareja &p) : a(p.a), b(p.b) {}
```

Para crear objetos usando el constructor copia se procede como sigue:

```
void main() {
    pareja par1(12, 32)
    pareja par2(par1); // Uso del constructor copia: par2 = par1
    int x, y;

    par2.Lee(x, y);
    cout << "Valor de par2.a: " << x << endl;
    cout << "Valor de par2.b: " << y << endl;

    cin.get();
}
```

También en este caso, si no se especifica ningún constructor copia, el compilador crea uno por defecto, y su comportamiento es exactamente el mismo que el del definido en el ejemplo anterior. Para la mayoría de los casos esto será suficiente, pero en muchas ocasiones necesitaremos redefinir el constructor copia.