

1. Categorías de los métodos
2. Inicialización de miembros
3. Métodos constructores
4. Inicialización de objetos con y sin constructores
5. Sobrecarga de funciones constructoras
6. Argumentos implícitos en constructores
7. El constructor por defecto
8. Método destructor
9. ¿Cuándo se ejecuta el destructor?

Categorías de métodos

En las clases, se pueden identificar varias categorías de métodos:

- ◆ **Métodos inicializadores:** inicializan atributos.
- ◆ **Métodos accedentes ó selectores:** devuelven el valor de los atributos. Cada método accedente devuelve un atributo.
- ◆ **Métodos mutadores o modificadores:** permiten cambiar el valor de los atributos.
- ◆ **Métodos visualizadores:** muestran el objeto, es decir, el valor de los atributos.
- ◆ **Métodos operadores:** realizan cálculos y generan resultados.
- ◆ Otras categorías.

Categorías de los métodos: ejemplo 1

```
class Cuenta
```

```
{  
  private:  
    long int numero_cuenta;  
    float saldo;  
    float interes_anual;  
  public:  
    void inicializar( long int num);  
    float dar_saldo();  
    float dar_interes();  
    void saldo (float s);  
    void interes( float i );  
    void ingreso ( float cantidad);  
    bool reintegro ( float r);  
    void mostrar_datos ();  
};
```

Método inicializador

Accedentes

Mutadores o modificadores

Método visualizador

Categorías de los métodos: ejemplo 2

```
class Disco
```

```
{ private:  
  char titulo [30];  
  int num_canciones  
  float precio;  
  Date fecha_compra;  
  public:  
  void inicializar (char * tit);  
  void inicializar (char * tit, float price);  
  
  char * dev_titulo ( );  
  int dev_num_canciones ( );  
  float dev_precio ( );  
  Date dev_fecha ( );  
  
  void mod_num_canciones( int n);  
  void mod_precio ( float p);  
  void mod_fecha ( Date f);  
};
```

Métodos inicializadores

Accedentes

Mutadores o modificadores

Categorías de los métodos: ejemplo 3

```
class Coleccion
{
private:
    Disco discos [100];
    int indice;
public:
    void inicializar ( ) ;
    bool esta_llena ( ) ;
    bool esta_vacia ( ) ;

    bool añadir_disco ( Disco ) ;

    float coste_coleccion ( ) ;
    int buscar_disco( char * );
};
```

Método inicializador

Métodos operadores

Mutadores o modificadores

Métodos Operadores

Inicialización de miembros

En la definición de una clase, solo está permitido señalar el tipo y el nombre de los miembros que la componen.

```
class Complejo
{
private:
    float real = 0;
    float imaginaria;
public:
    void inicializar ( ) ;
    ....
};
```

ERROR !!!

El lugar idóneo para situar las asignaciones iniciales a miembros es en los métodos de la clase.

```
void Complejo:: inicializar( )
{
    real = 0;
    imaginaria = 0;
}
```

Las asignaciones iniciales tienen un sitio específico en el cuerpo de ciertos métodos especiales denominados **CONSTRUCTORES**

Métodos constructores

Los diseñadores del lenguaje decidieron asignar la tarea de inicializar los objetos a los métodos **constructores**. La consideraron tan importante que si el programador no declara ningún método constructor, el compilador se encarga de definir un **constructor de oficio**.

Un **método constructor** es una función miembro especial que lleva a cabo la inicialización automática de cada objeto de la clase en el momento en que se declara.

- Un constructor es una función miembro pública con el mismo nombre de la clase.
- Sin indicación de tipo devuelto (ni siquiera void).
- **Se ejecuta automáticamente al crearse un objeto de la clase.**

Métodos constructores

```
class MiClase
{
  private:
  ...
  public:
  MiClase();
  ...
};
```

Función miembro pública con el mismo nombre de la clase.

Método constructor

Sin indicación de tipo devuelto

```
class Complejo
{
  private:
  float real;
  float imaginaria;

  public:
  Complejo();
  ...
};
```

Método constructor

Métodos constructores

El constructor puede implementarse fuera de la declaración de la clase, al igual que cualquier otro método.

```

class MiClase
{
    private:
        ....
    public:
        MiClase();
        ....
};

MiClase :: MiClase()
{ ... }
    
```

```

class Cuenta
{
    private:
        long int numero_cuenta;
        float saldo;
        float interes_anual;
    public:
        Cuenta(long int num);
        ....
};

Cuenta ::Cuenta( long int num)
{
    numero_cuenta = num;
    saldo = 0;
    interes_anual = 0;
}
    
```

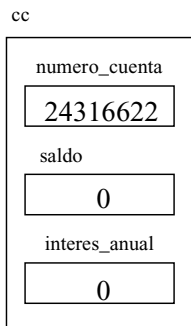
Inicialización de objetos sin constructores

```

class Cuenta
{
    private:
        long int numero_cuenta;
        float saldo;
        float interes_anual;
    public:
        void inicializar( long int num );
};

void Cuenta:: inicializar( long int num)
{
    numero_cuenta = num;
    saldo = 0;
    interes_anual = 0;
}
    
```

Función miembro específica para inicializar el objeto



```

...
void main();
{
    Cuenta cc;
    cc.inicializar( 24316622 ) ;
    ...
}
    
```

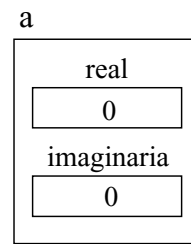
Paso de mensaje de inicialización

Inicialización de objetos sin constructores

```
class Complejo  
{  
private:  
float real ;  
float imaginaria;  
  
public:  
void inicializar( );  
....  
};
```

```
void Complejo:: inicializar( )  
{  
real = 0;  
imaginaria = 0;  
}
```

Función miembro específica para inicializar el objeto



```
...  
void main();  
{  
Complejo a;  
a.inicializar( );  
...  
}
```

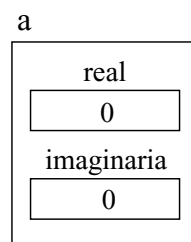
Paso de mensaje de inicialización

Inicialización de objetos con constructores

```
class Complejo  
{  
private:  
float real ;  
float imaginaria;  
public:  
Complejo( );  
....  
};
```

```
Complejo :: Complejo( )  
{  
real = 0;  
imaginaria = 0;  
}
```

Función miembro Constructora

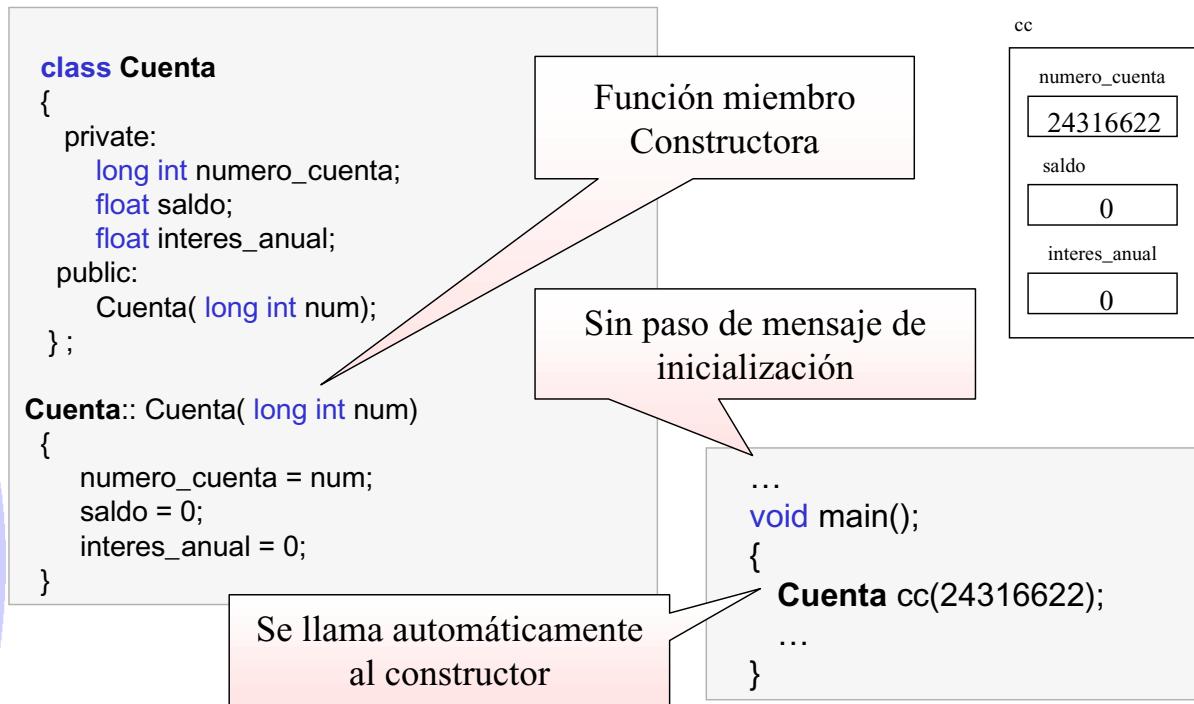


Sin paso de mensaje de inicialización

```
...  
void main();  
{  
Complejo a;  
...  
}
```

Se llama automáticamente al constructor

Inicialización de objetos con constructores



¿ Es más adecuado colocar el código de inicialización en un método constructor que en una función miembro como inicializa() ?

SI.

Así podemos estar seguros de que el código de inicialización se va a ejecutar siempre sobre cualquier objeto que se cree de esa clase.

Si la inicialización se encuentra en otro método, se nos puede olvidar enviar el mensaje correspondiente al objeto.

Sobrecarga de funciones constructoras

Al igual que para el resto de los métodos, podemos tener varias funciones constructoras; esto nos produce varias formas de inicialización.

```

Complejo :: complejo( )
{
    real = 0;
    imaginaria = 0;
}
Complejo :: complejo ( float a )
{
    real = a;
    imaginaria = 0;
}
Complejo :: complejo ( float a , float b )
{
    real = a;
    imaginaria = b;
}
    
```

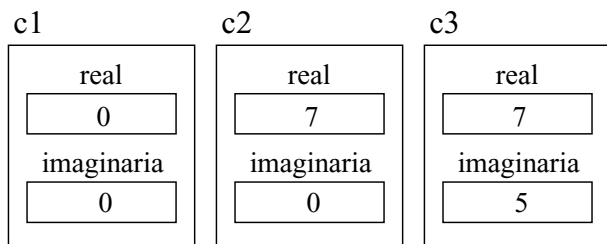
```

class Complejo
{
    private:
        float real ;
        float imaginaria;
    public:
        Complejo ( );
        Complejo ( float a );
        Complejo ( float a, float b );
        ....
};
    
```

Sobrecarga de funciones constructoras

```

Complejo :: complejo( )
{
    real = 0;
    imaginaria = 0;
}
Complejo :: complejo ( float a )
{
    real = a;
    imaginaria = 0;
}
Complejo :: complejo ( float a , float b )
{
    real = a;
    imaginaria = b;
}
    
```



```

...
void main();
{
    Complejo c1;
    Complejo c2 ( 7);
    Complejo c3 (7, 5);
}
    
```

Argumentos implícitos en constructores

Una forma de reducir el número de constructores, es utilizar constructores con argumentos por omisión. Pero hay que tener mucho cuidado para que no se dé ambigüedad:

```

class Complejo
{
private:
    float real ;
    float imaginaria;
public:
    Complejo ( );
    Complejo ( float a, float b = 0 );
    ....
};
    
```

Constructor sin argumentos

Constructor con 1 ó 2 argumentos

```

Complejo:: complejo( )
{
    real = 0;
    imaginaria = 0;
}
Complejo:: complejo ( float a , float b )
{
    real = a;
    imaginaria = b;
}
    
```

```

...
Complejo c1;
Complejo c2 ( 7);
Complejo c3 (7, 5);
...
    
```

Argumentos implícitos en constructores

```

class Complejo
{
private:
    float real ;
    float imaginaria;
public:
    Complejo ( );
    Complejo ( float a = 0, float b = 0 );
    ....
};
    
```

Constructor sin argumentos

Constructor con 0, 1 ó 2 argumentos

```

Complejo:: complejo( )
{
    real = 0;
    imaginaria = 0;
}
Complejo:: complejo ( float a , float b )
{
    real = a;
    imaginaria = b;
}
    
```

```

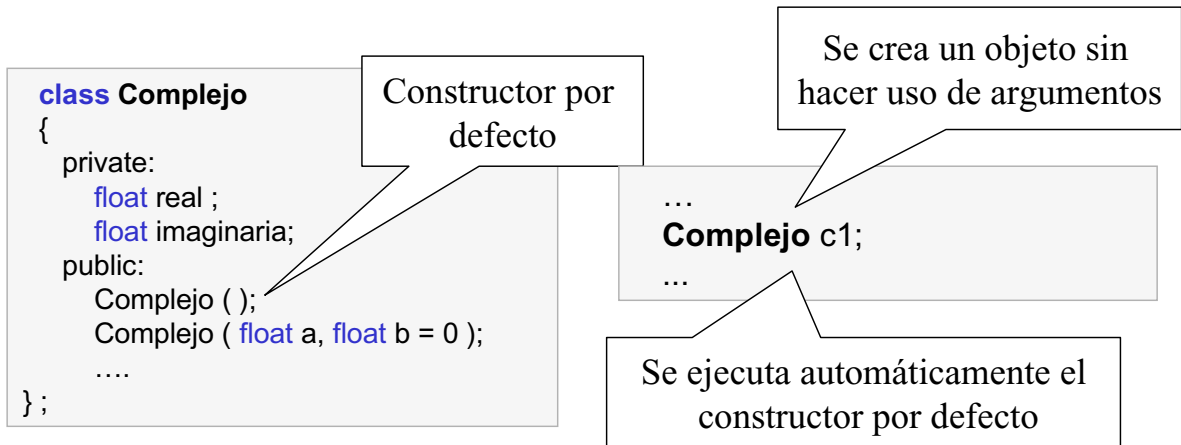
...
void main();
{
    Complejo c1;
}
    
```

AMBIGÜEDAD
¿A qué función constructora se ha de llamar?

El constructor por defecto

En las clases debe existir un constructor que no tenga argumentos, se le llamará *constructor por defecto* o *constructor predeterminado*.

El constructor por defecto, se ejecuta automáticamente cada vez que se crean objetos sin pasar argumentos.



¿Qué pasa si no se implementa el constructor por defecto?

Si en la clase no se proporciona ningún constructor, el compilador añade uno por defecto que se usará en la creación de objetos.

Si en clase se han definido otros constructores, pero ninguno es el constructor por defecto (sin argumentos), algunos compiladores pueden darnos errores y obligarnos a incluir uno (no para el C++BuilderX).

Destruyores

Todas las clases tienen uno y sólo un destructor.

Si no se indica lo contrario, el compilador genera un destructor por defecto cuya única función es liberar el espacio que ocupan los miembros estáticos (no reservados dinámicamente) del objeto que se destruye.

Los destructores son funciones miembro que realizan tareas de “limpieza”.

- Un destructor es una función miembro pública con el mismo nombre de la clase pero precedido por el símbolo ~
- Sin indicación de tipo devuelto (ni siquiera void).
- **Se ejecuta automáticamente cuando se destruye un objeto.**

Método destructor

```
class MiClase
{
  private:
  ...
  public:
  MiClase();
  ~MiClase();
  ...
};

MiClase :: ~ MiClase()
{ ... }
```

Función miembro pública con el mismo nombre de la clase precedido por el símbolo ~

Sin indicación de tipo devuelto

Método constructor

Método destructor

```
class Complejo
{
  private:
  float real ;
  float imaginaria;
  public:
  Complejo( );
  ~Complejo( )
  {
  }
  ...
};
```

Solo podemos definir un único destructor por clase y sin argumentos.

¿Cuándo se ejecutan los destructores?

Los objetos se construyen cuando se declaran. Así sabemos perfectamente cuándo se ejecutan los constructores.

¿Cuándo se ejecutan los destructores?

- ◆ Los destructores se ejecutan cuando termina la ejecución de bloque de código en el que se han creado los objetos.

```
void main();  
{  
    Complejo c1, c2;  
    ....  
}
```

Al terminar la ejecución de la función `main()` se destruyen los dos objetos `c1` y `c2`.

- ◆ También se pueden ejecutar mediante paso de mensajes.

```
void main();  
{  
    Complejo c1, c2;  
    ....  
    C1. ~Complejo( );  
}
```

Resumen

Constructores y destructores

- No retornan ningún valor ni siquiera **void**.
- Si no hay ninguno, se crea uno por defecto sin argumentos.

Constructores

- Pueden tener cualquier tipo y número de argumentos, incluso por defecto.
- Puede haber muchos en una clase.
- Se llaman cuando se declara un objeto.

Destructores

- Sólo hay un destructor por clase.
- El destructor no tiene argumentos.
- Tiene como misión principal la de liberar memoria.