

Constructores y Destructores

Constructores
Destructores
Funciones inline
Inicialización vs. Asignación

Ing. Armandina Leal Flores

Constructores

- ▶ Función que se utiliza para inicializar los elementos de la clase.
- ▶ Características:
 - Tiene el mismo nombre de la clase a la que pertenece.
 - Se ejecuta cuando el objeto se crea.
 - No tiene valor de retorno, ni siquiera es **void**.
 - La clase puede tener más de un constructor (se puede sobrecargar).
 - No se puede llamar directamente al constructor.

Ejemplo...

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

class Ejemplo
{
    private:
        int iA, iB;
    public:
        Ejemplo () { iA = 0; iB = 0; }
        Ejemplo ( int il, int ij ) { iA = il; iB = ij; }

    void Despliega ()
    { cout << "a=" << iA << " b = " << iB << endl; }
};
```

Obj1(); //error

```
#include <iostream>
#include "Ejemplo.h"
using namespace std;
```

```
int main()
{
```

Ejecuta el constructor

```
    Ejemplo Obj1, Obj2(4, 5);
    Obj1.Despliega();
    Obj2.Despliega();
```

```
    return 0;
}
```

Constructores con parámetros

- ▶ Los constructores pueden tener parámetros.
- ▶ Evitan una llamada extra a una función que inicialice los elementos de la clase.

```
#include <iostream>
#include "Ejemplo.h"
using namespace std;

int main()
{
    Ejemplo Obj1;
    Obj1.Inicializa(4, 5);
    Obj1.Despliega();

    return 0;
}
```

Ineficiente

```
#include <iostream>
#include "Ejemplo.h"
using namespace std;

int main()
{
    Ejemplo Obj1(4, 5);
    Obj1.Despliega();

    return 0;
}
```

Eficiente

Destructor

- ▶ Tiene el mismo nombre de la clase excepto que al nombre le precede el carácter `~`.
- ▶ Esta función no retorna un valor ni siquiera es `void`.
- ▶ No tiene argumentos por lo que no se puede sobrecargar.
- ▶ Se ejecuta automáticamente cuando el objeto es destruido como por ejemplo cuando se termina la ejecución del programa.
- ▶ Se utilizan para liberar memoria dinámica, cerrar archivos, etc.

Ejemplo...

```
#include <iostream>
#include <string>
using namespace std;
class unstring
{
private:
    char *sS;
public:
    unstring (char cC)
    {   sS = new char[2];
        sS[0] = cC;
        sS[1] = '\0';
    }
    unstring (const char *sOtro)
    {   int N = strlen(sOtro);
        sS = new char [N+1];
        strcpy(sS, sOtro);
    }
    ~unstring() { delete [] sS; }
    void show () { cout << sS << endl; }
};
```

```
#include <iostream>
#include "unstring.h"
using namespace std;

int main()
{
    char cX[] = "palabra";
    unstring Obj1('a'), Obj2(cX);
    Obj1.show();
    Obj2.show();
    return 0;
}
```

Cuándo se ejecutan...

- ▶ El **constructor** de un objeto local se ejecuta cuando se encuentra la declaración del objeto.
- ▶ El **destructor** de un objeto local se ejecuta cuando se termina el bloque en el que se declaró el objeto.
- ▶ Cuando hay más de un objeto, los **destructores** se ejecutan en orden inverso a como se construyeron.

Funciones inline

- ▶ Funciones cortas que no se llaman sino que su código se expande en el lugar en el que se invocan.
- ▶ Permiten crear un código muy eficiente.

Ejemplos...

```
#include <iostream>
class myclass
{
private:
    int iA, iB;
public:
    void init (int il, int ij);
    void show ();
};

inline void myclass::init (int il, int ij)
{   iA = il; iB = ij; }
inline void myclass::show()
{   cout << "a=" << iA << " b = " << iB; }
```

```
#include <iostream>

class myclass
{
private:
    int iA, iB;
public:
    myclass (int iI, int iJ) { iA = iI; iB = iJ; }
    void show ()
    {
        cout << "a=" << iA << " b = " << iB; }
};
```

El constructor y el método show son funciones inline ya que su definición está dentro de la clase.

Inicialización vs. Asignación

▶ Inicialización

- Ocurre cuando el objeto es creado con su valor.

▶ Asignación:

- Ocurre cuando un objeto existente se sobrescribe con el contenido de otro objeto.

▶ Siempre que se pueda es mejor Inicializar que Asignar.

Inicialización vs. Asignación

- ▶ En C++ se considera que todo es un objeto, incluso las variables de tipos básicos como int, char o double.
- ▶ Entonces cualquier variable (u objeto) tiene un constructor por default.
- ▶ Sólo los constructores admiten inicializadores.
- ▶ Ejemplo: El constructor:

```
stardate () { iYear = 0; iCentury = 0; iMillennium = 0; }
```

se puede sustituir por:

```
stardate (): iYear (0), iCentury(0), iMillennium (0) { }
```

- ▶ Es muy recomendable utilizar la inicialización en lugar de la asignación siempre que sea posible, ya que se desde el punto de vista de C++ es mucho más seguro (sobre todo cuando la clase tiene miembros de otra clase).

Ejemplo

```
class stardate
{
    private:
        int      iYear;
        int      iCentury;
        int      iMillennium;

    public:

    //constructors
    stardate (): iYear (0), iCentury(0), iMillennium (0) { }

    stardate( int y): iYear (y), iCentury(y/100 + 1), iMillennium (y/1000 + 1) { }
```



Inicialización de los
elementos de la clase