

Metodologías y Técnicas de Programación II

Programación Orientada a Objeto (POO) C++

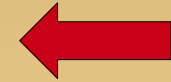
Inicialización y Limpieza I

6.0 Estado del Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

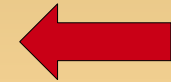
C++
Mi primera Clase



Repaso de Conceptos

Estándares de Programación
Punteros y Memoria

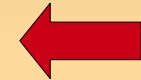
E/S
Control y Operadores



Clases y Objetos en C++

Uso y aplicación
Constructores

Funciones Amigas
Constantes e "inline"



Sobrecarga

De Operadores

De Funciones

Herencia.

Tipos de Visibilidad

Herencia Múltiple

Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

Plantillas

Contenedores

Iteradores

6.1 Repaso de Sesiones Anteriores

Relaciones de Amistad

Si A -> B y B -> C	NO A->C
Si A -> B	NO B->A
Si A -> B y b <<B	NO A->b (b es el hijo de B)

Podemos definir relaciones de amistades con una función, con un método de otra clase o con una clase entera.

Friend = “Puedes acceder a mis atributos privados como si fueras uno de mis métodos”

```
friend void g(X*, int); // Global friend
friend void Y::f(X*); // Un método de la clase Y
friend class Z; // Toda Z es mi amiga
friend void h();
```

6.1 Repaso de Sesiones Anteriores

Las relaciones de Amistad

Son una forma de saltarme el control de acceso
(De forma controlada)

Pueden resultar útiles por razones de rendimiento por ejemplo
(Función `aprobar_asignatura()` a la que permito que acceda a
nota directamente.
`alumno.nota = 8)`

6.2 Inicialización de Objetos

Constructores y Destructores

C++, al contrario que C, permite incluir funciones en el interior de las estructuras. Normalmente estas funciones tienen la misión de manipular los datos incluidos en la estructura.

En realidad nosotros utilizamos clases.

Dos funciones muy particulares son las de inicialización y limpieza, o **constructor**, y **destructor**.

El constructor es una función sin tipo de retorno y con el mismo nombre que la estructura. **Alumno()**

El destructor tiene la misma forma, salvo que el nombre va precedido el operador "~". **~Alumno()**

6.2 Inicialización de Objetos

Constructores y Destructores

```
class Punto
{
    int x_;
    int y_;
    Punto();           // Declaración del constructor
    Punto(int x, int y); // Otro constructor.
};

Punto::Punto()
{
    x_ = y_ = 0;
}

Punto::Punto(int x, int y)
{
    x_ = x;
    y_ = y;
}
```

6.2 Inicialización de Objetos

Constructores

- Sirven para inicializar un objeto de una determinada clase al mismo tiempo que se declara.
- Tienen el mismo nombre que la clase.
- No retornan ningún valor.
- No pueden ser heredados.
- En general deben ser públicos ¿Por qué?

6.2 Inicialización de Objetos

Constructores

```
class Pareja
{
public:
    // Constructor
    Pareja(int a, int b);
    // Funciones miembro o Metodos
    void Lee(int &a, int &b);
    void Guarda(int a, int b);
private:
    // Datos miembro o Atributos
    int a_, b_;
};

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

```
void Pareja::Lee(int &a, int &b)
{
    a_ = a;
    b_ = b;
}

void Pareja::Guarda(int a, int b)
{
    a_ = a2;
    b_ = b2;
}

Pareja par1(12, 32);
int x, y;
par1.Lee(x, y);
cout <<"Valor par1.a:"<< x << endl;
cout <<"Valor par1.b:"<< y << endl;
Pareja par1; // Ilegal ¿Por que?
Pareja par1(); // Ilegal (función)
```

6.2 Inicialización de Objetos

Modo simplificado de inicialización

Se basa en que en C++ todo son objetos (int también lo es).
Cualquier variable (u objeto) tiene un constructor por defecto

Inicializadores.

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 ":".

```
pareja::pareja(int a, int b)
{
    a_ = a;
    b_ = b;
}
```

```
// Es más seguro y eficiente así:
pareja::pareja(int a,int b) : a_(a), b_(b) {}
```

6.2 Inicialización de Objetos

Sobrecarga de Constructores

Podemos añadir varios constructores a las clase. Es decir: la función del constructor puede sobrecargarse.

```
class Pareja
{
public:
    // Constructor
    Pareja(int a, int b);
    Pareja() : a_(0), b_(0) {}
    // Funciones miembro o Metodos
    void Lee(int &a, int &b);
    void Guarda(int a, int b);
private:
    // Datos miembro o Atributos
    int a_, b_;
};
```

Estamos simulando el **constructor por defecto**.

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.

6.2 Inicialización de Objetos

Constructores y argumentos por defecto

Podemos asignar valores por defecto a los argumentos del constructor, de este modo reduciremos el número de constructores necesarios.

```
class Pareja
{
public:
    // Constructores
    Pareja(int a, int b);
    Pareja() : a_(0), b_(0){};

    // Metodos
    void Lee(int &a, int &b);
    void Guarda(int a, int b);
private:
    // Atributos
    int a_, b_;
};
```

```
class Pareja
{
public:
    // Constructor
    Pareja(int a=0,int b=0) :a_(0),b_(0){};

    // Metodos
    void Lee(int &a, int &b);
    void Guarda(int a, int b);
private:
    // Datos miembro o Atributos
    int a_, b_;
};
```

Constructores

**Seguiremos con el
Constructor Copia**

...