

Metodologías y Técnicas de Programación II

Programación Orientada a Objeto (POO) C++

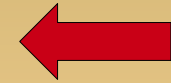
Sobrecarga de Funciones

Estado del Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

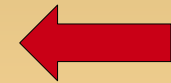
C++
Mi primera Clase



Repaso de Conceptos

Estándares de Programación
Punteros y Memoria

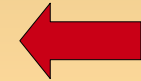
E/S
Control y Operadores



Clases y Objetos en C++

Uso y aplicación
Constructores
Constantes e "inline"

Funciones Amigas
Sobrecarga de Funciones



Sobrecarga

De Operadores

Creación Dinámica de Objetos

Herencia.

Tipos de Visibilidad

Herencia Múltiple

Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

Plantillas

Contenedores

Iteradores

8.1 Repaso de Sesiones Anteriores

Inicialización y Limpieza de Objetos

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()** Lo definimos cuando hay punteros

El constructor copia tiene este aspecto: **Alumno(const Alumno& a)**

Si creamos un constructor copia el compilador no realizará una copia bit a bit cuando cree un nuevo objeto de otro existente.

Las referencias necesitan un área de memoria que referenciar:

```
int& x = 1;      // Mal
```

```
const int& y = 1; // Bien
```

8.2 Sobrecarga de Funciones

Utilización de Nombres

Formas de decir lo mismo:

- Declaro una variable de un tipo.

- Creo un objeto de una clase.

- Instancio un objeto de la clase Z.

Estamos asignado un nombre a una zona de memoria.

En el caso de una función también es un nombre de una zona de memoria que el lugar de datos tiene instrucciones. Una función es un nombre para una acción.

Los nombres que ponemos a las variables y a las funciones hacen que nuestro código sea más legible o que sea incomprensible.

Reutilización: Es más fácil con un código fácil de comprender.

8.2 Sobrecarga de Funciones

Utilización de Nombres

En la mayoría de los lenguajes de programación:

`imprime_entero()` `imprime_float()` `imprime_cadena()`

tenemos que crear esas tres funciones de forma diferente.

Trabajo extra:

Para el Programador Creador

Para el Programador Cliente (Lector)

En C++ la sobrecarga es obligada porque el nombre del constructor está fijado. Si necesito un constructor sin argumentos y otro con dos enteros...

Además con los argumentos por defecto puedo ahorrarme funciones.

Llamadas a la misma función: `f("Hola")`, `f("Eh", 1)` y `f("Buenas", 2, 'c')`
`f(char* cad, int i=0, char='o');`

8.2 Sobrecarga de Funciones

Utilización de Nombres

Podemos utilizar el mismo nombre siempre que la lista de argumentos sea diferente.

El compilador utiliza:

Nombre de la función

Ámbito

Lista de argumentos

Esto no es sobrecarga. Son funciones diferentes por el ámbito:

```
void f();  
class X { void f(); };
```

No tenemos sobrecarga en el valor de retorno.

```
void f();  
int f();
```

¿Por qué?.....

8.2 Sobrecarga de Funciones

Argumentos por Defecto

Un argumento por defecto es un valor que se da en la declaración para que el compilador lo inserte automáticamente en el caso de que no se proporcione ninguno en la llamada a la función.

```
Pareja(int n);  
Pareja(int a, int b);
```

```
Pareja(int a, int b=-1);
```

```
Pareja a(3);  
Pareja b(1,2);
```

La sobrecarga y los argumentos por defecto nos permiten utilizar el mismo nombre de función para situaciones diferentes.

Si las funciones tiene comportamientos muy diferentes, normalmente no tiene sentido utilizar argumentos por defecto.

8.2 Sobrecarga de Funciones

Uso de la Sobrecarga

```
struct punto3D
{
    float x, y, z;
};

class punto
{
public:
    punto(float xi, float yi, float zi)
        : x(xi), y(yi), z(zi) {}
    punto(punto3D p)
        : x(p.x), y(p.y), z(p.z) {}
    void Asignar(float xi, float yi, float zi)
    {
        x = xi;
        y = yi;
        z = zi;
    }
};
```

```
void Asignar(punto3D p)
{
    Asignar(p.x, p.y, p.z);
}

void Ver()
{
    cout << "(" << x << ", " << y
        << ", " << z << ")" << endl;
}

private:
    float x, y, z;
};
```

8.2 Sobrecarga de Funciones

Uso de Argumentos por Defecto

```
class Punto
{
public:
    Punto(float xi, float yi, float zi) :
        x(xi), y(yi), z(zi) {}
    void asignar(float xi, float yi = 0, float zi = 0)
    {
        x = xi;
        y = yi;
        z = zi;
    }
private:
    float x, y, z;
};

P.Asignar(12);
P.Asignar(16,35);
P.Asignar(34,43,12);
```

8.2 Sobrecarga de Funciones

Resumen

No deberíamos utilizar argumentos por defecto si hay que incluir una condición en el código (un “if”). En este caso es mejor tener varias funciones sobrecargadas.

Uso típico de argumentos por defecto es cuando empieza con una función con un conjunto de argumentos, y después de utilizarla por un tiempo se da cuenta de que necesita añadir más argumentos.

```
f (int a);    f(int a, int b=0); // Es compatible con lo anterior.
```

Tanto la sobrecarga de funciones como el uso de argumentos por defecto nos hacen más fáciles las llamadas a las funciones y además son más fáciles de leer.