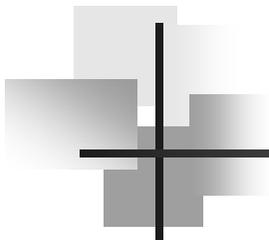


INTRODUCCIÓN A LA POO EN C++

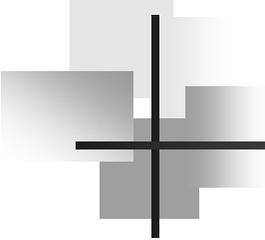
ÍNDICE DEL TEMA

- 1.- Introducción
- 2.- Diferencias C/C++
- 3.- Programación orientada a objetos
- 4.- Aspectos avanzados C++



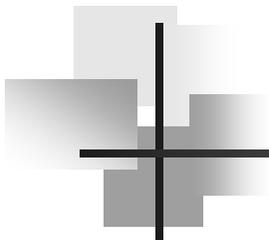
1. Introducción

- Lenguaje C
 - Lenguaje de propósito general
 - Originalmente para programación de sistemas
 - “no muy alto nivel”
 - pequeño, sencillo, eficiente, portable
- C++ es una evolución del lenguaje C:
 - Es un superconjunto mejorado del lenguaje C.
 - Facilidad para la abstracción de datos.
 - Orientado a objetos.



INTRODUCCIÓN A LA POO EN C++

- 1.- Introducción
- 2.- Diferencias C/C++
- 3.- Programación orientada a objetos
- 4.- Aspectos avanzados C++



1. Diferencias C/C++: Declaración de variables

- Ficheros código fuente
 - C: .c
 - C++: .cpp
- Declaración de variables:
 - C: Al principio de cada bloque.

```
int Funcion(int x)
{
    int x, y;
    float a, b;
        ...
        ...
    printf("\nIntroduzca a y b");
    scanf("%f", &a);
    scanf("%f", &b);
}
```

1. Diferencias C/C++: Declaración de variables

- C++: Se puede declarar una variable en cualquier lugar.

```
int main()  
{  
    int i;  
  
    i=4;  
    printf("%d\n", i);  
  
    int j=4;           // declaramos una nueva variable  
  
    for(int k=0; k<5; k++) // declara la variable k  
        printf("%d\n", k);  
}
```

1. Diferencias C/C++: Sobrecarga de funciones

- C: Dos funciones no pueden tener el mismo nombre. Desventaja si tenemos funciones similares que hacen operaciones similares.

```
int SumaVectorEnteros(int *vector, int n)
{
    int i, suma = 0;
    for(i=0;i<n;i++)
        suma +=vector[i];
    return suma;
}

float SumaVectorFloat(float *vector, int n)
{
    int i;
    float suma = 0.0;
    for(i=0;i<n;i++)
        suma +=vector[i];
    return suma;
}
```

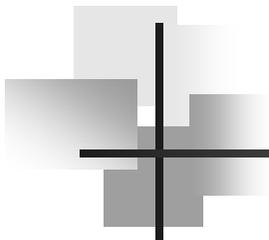
1. Diferencias C/C++: Sobrecarga de funciones

- C++: En C++ una función se puede redefinir tantas veces como se quiera (sobrecarga de funciones).

- Cada función se distingue por sus parámetros:

```
int SumaVector(int *vector, int n)
{
    ...
}
float SumaVector(float *vector, int n)
{
    ...
}
double SumaVector(double *vector, int n)
{
    ...
}
```

- Pueden tener parámetros de tipos diferentes e incluso un número diferente de ellos.
- Dos funciones no pueden distinguirse únicamente por el tipo de parámetro devuelto.
- El compilador utiliza una u otra implementación según el número y tipo de los parámetros.



1. Diferencias C/C++: Parámetros por defecto

- C: Se debe especificar el valor de todos los argumentos al llamar a una función.

```
main ()
{
    int suma, vector[50];
    ...
    //guardamos datos en vector
    suma = SumaVectorEnteros(vector, 50);
}
```

1. Diferencias C/C++ : parámetros por defecto

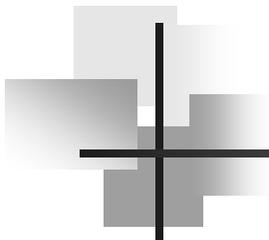
- C++: Se permite dar valores por defecto a los parámetros de una función.
- Los parámetros por defecto pueden ser omitidos en la llamada a la función. Deben estar siempre al final en la lista de parámetros.

```
int SumaVector(int *vector, int n=10); //prototipo
```

```
int SumaVector(int *vector, int n=10)
{
    ...
}
```

```
SumaVector(vector, 50);
```

```
SumaVector(vector); // Utiliza n = 10
```



1. Diferencias C/C++ : Prototipos en C++

- Utilizan el mismo formato de ANSI C pero en este caso son obligatorios.

```
int AbrirFichero(char nombre[]);
```

```
// Función con un parámetro de tipo  
// array de caracteres que devuelve  
// un entero.
```

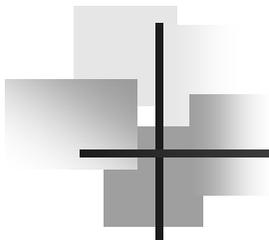
```
PintarCuadrado( int x, int y, int tam);
```

```
// se pasan tres parámetros enteros  
// y devuelve un entero (tipo por  
// defecto)
```

```
void LeerCadena();
```

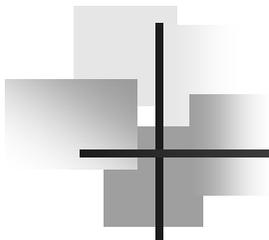
```
// función sin parámetros ni valores  
// devueltos
```

```
Equivale a void LeerCadena(void)
```



INTRODUCCIÓN A LA POO EN C++

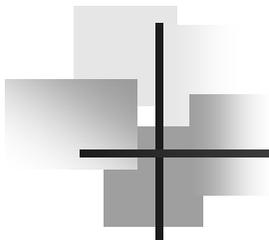
- 1.- Introducción
- 2.- Diferencias C/C++
- 3.- Programación orientada a objetos
- 4.- Aspectos avanzados C++



3. Programación orientada a objetos

- Programación estructurada
 - Descomposición del problema en funciones
 - Se identifican las funciones que se necesitan para resolver el problema.
 - El problema se resuelve con un conjunto de llamadas a estas funciones
 - Control de Flujo
 - A partir de unos datos de entrada hay que conseguir unos datos de salida
 - Identifica las estructuras de datos que se necesitan
 - El problema se resuelve haciendo transformaciones de esos datos
- Programación Orientada a Objetos (POO)
 - Se identifican los objetos abstractos que representan el problema
 - Se identifican las operaciones abstractas que hay que realizar con esos objetos
 - La solución al problema es una secuencia de llamadas a esos objetos

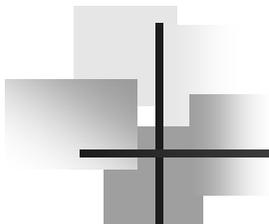
**Código más robusto,
fácil de mantener, seguro y
reutilizable**



3. POO

- CLASE

- Una clase define una variable, real o abstracta, acerca de la cual se almacenan datos y las funciones que manejan y controlan estos datos.
- Una clase define las funciones y atributos de un tipo de objetos.
- Ej: CLASE Coche
 - Atributos: Color, Velocidad, Carburante, etc.
 - Funciones: Andar, Parar, Girar, etc.



3. POO

■ OBJETO

- Un objeto es una instancia de una clase
 - Ej. Objetos: empleado Perez, alumno Luis, ...
- Se descomponen en:
 - Características del objeto
 - Funciones que actúan sobre estas características
- Una clase es una agrupación de un conjunto de objetos de similares características.
- Ej. Objeto alumno
 - Objeto_Luis, datos:
 - Nombre=Luis
 - Edad=20
 - Dirección=C/Lepanto
 - Calificaciones=...
 - Objeto_Marta, datos:
 - Nombre=Marta
 - Edad=22
 - Dirección=C/Velázquez
 - Calificaciones=...
 - Funciones:
 - Poner nombre
 - Poner calificación
 - Añadir foto

3. POO

- Clase:

- Podemos entender una clase de C++ a partir de una estructura de C.

Ejemplo:

```
struct alumno{
    char nombre[20];
    char direccion[20];
    float notas[3];
};
struct alumno pepe = {"Luis Perez", "Lepanto", 5.0, 5.0, 5.0};
struct alumno marta = {"Marta Lopez", "Velazquez", 7.0, 7.0, 7.0};
```

- Una clase la podemos entender como una generalización de las estructuras de C.
 - Las clases añaden cierto control de acceso sobre los datos de una clase.
 - Se añaden unas funciones que pueden operar sobre los datos de la clase.
- En C++ los objetos son las variables concretas que se crean de una determinada clase.

3. POO

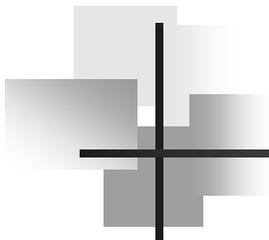
- Clase:

```
class CAumno{
    private: //Datos de la clase
    char nombre[20];
    char direccion[20];
    float notas[3];

public:
    CAumno(void); // constructor por defecto
    CAumno(char *nom, char*direcc, float* notas); // constructor a partir de dos reales
    ~CAumno(void); // destructor

    void PonerNombre(void);
    void PonerNotas(void);
    void VisualizarDatos(void);
};
```

- Una clase esta formada fundamentalmente por:
 - Unos datos de la clase
 - Unos métodos (funciones) que actúan sobre esos datos



INTRODUCCIÓN A LA POO EN C++

- 1.- Introducción
- 2.- Diferencias C/C++
- 3.- Programación orientada a objetos
- 4.- Aspectos avanzados C++

4. Aspectos avanzados de C++ : paso de parámetros a una función (1)

- Paso por Valor:

- Se pasa una copia temporal (no puede modificarse el original):

```
int VisualizarNota(int nota)
{
    nota=5;           // modificamos la copia temporal
    cout << "Nota: " << nota;
}
VisualizarNota(notaJuan); // notaJuan no se modifica
```

4. Aspectos avanzados de C++: paso de parámetros a una función (2)

- Paso por Dirección:

- Se pasa la dirección en memoria de la variable original (puede modificarse):

```
int PonerNota(int *pnota)
```

```
{
```

```
    *pnota=5;           // modificamos la variable original
```

```
    cout << "Nota: " << *pnota;
```

```
}
```

```
PonerNota(&notaJuan); //permite que se modifique notaJuan
```

4. Aspectos avanzados de C++ : paso de parámetros a una fun (3)

- Paso por Referencia (específico de C++):

- Se pasa una referencia (*nombre alternativo*) de la variable original (puede modificarse):

```
int PonerNota(int &nota)
```

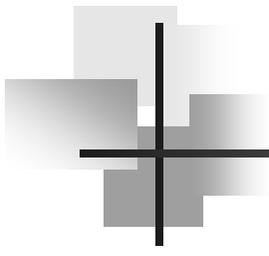
```
{
```

```
    nota=5;                // modificamos la variable original
```

```
    cout << "Nota: " << nota;
```

```
}
```

```
PonerNota(notaJuan); //permite que se modifique notaJuan
```



4. Aspectos avanzados de C++: paso de parámetros a una función (4)

- Paso por Referencia (Ventajas):
 - La sintaxis es más sencilla
 - Es similar al paso por valor.
 - No es necesario utilizar el operador indirección (*).
 - Permite modificar la variable pasada como parámetro
 - Es más eficiente que el paso por valor para estructuras de datos ya que no se realiza una copia de la variable original.

4. Aspectos avanzados de C++: asignación dinámica de memoria

- Nuevos operadores alternativos a las funciones malloc() y free():

- Operador new:

```
char *p;  
CPila *ppila;
```

```
p= new char[25]; // solicita 25 caracteres  
ppila= new CPila; // solicita un objeto de la clase CPila
```

- Operador delete:

```
delete []p; // elimina la memoria asignada  
delete ppila;
```

4. Aspectos avanzados de C++ : forzamiento de tipos (cast)

- Se puede utilizar la misma sintaxis de C:

```
int a=2; int b=4; float c;  
c= (float) a/b;      // fuerza el tipo de a a float
```
- C++ define un nuevo tipo de forzamiento:

```
c= float(a) /b;      // fuerza el tipo de 'a' a float
```
- Permite que el programador redefina nuevos forzamientos para tipos no estándar (se trata de la llamada a una función)

Primer programa C++

```
#include <iostream.h>
```

```
main( )
```

```
{
```

```
    cout << "Hola, mundo.\n";
```

```
}
```

- Nuevo sistema de entrada/salida basado en clases.
- El sistema coexiste con el clásico de C

- Cabecera de la librería de clases de entrada/salida

- cout Objeto de la clase ostream
- << operador escritura (genera un resultado de tipo ostream)

- Equivale a:

```
cout << "Hola, " << "mundo." << "\n";
```