

Cuarta Sesión

Metodologías y Técnicas de Programación II Programación Orientada a Objeto (POO)

C++

Profesor:

José Luis Marina
jmarina@nebrija.es

Laboratorios:

José Luis Marina
jmarina@nebrija.es
Borland Builder 6.0

Puntuación:

Prácticas Laboratorio:	20%
Diarias	20%
Trabajo Laboratorio	80%

Exámen Parcial	15%
----------------	------------

Exámen Final	65%
--------------	------------

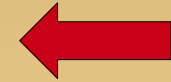
Exámen Final Extraordinario	70%
------------------------------------	------------

4.0 Estado del Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

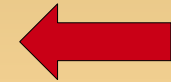
C++
Mi primera Clase



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

4.1 Programación Orientada a Objeto

Conceptos

```
int i;           ClaseEnteros mi_entero;  
i=33;          mi_entero.asignar(33);  
cout << i;     mi_entero.imprime_lo_que_vales();
```

Todo es un Objeto:

Un objeto es como una variable mejorada.

Cogemos un componente conceptual de nuestro problema y lo representamos como un objeto en nuestro programa (perro, edificio, tuerca, ventana,...)

Los objetos se comunican mediante mensajes:

Haremos programas que serán grupos de objetos enviando mensajes a otros para decirles qué hacer. Petición de invocación a una función que pertenece a un objeto en particular.

Datos Propios:

Podemos crear un nuevo tipo de objeto haciendo un paquete con otros objetos. Oculto la complejidad. Coche (ruedas, motor,etc)

Cada objeto es de un tipo:

Un clase define el tipo de un objeto. Clase Enteros -> un_entero.

Un objeto es una instancia de una clase determinada.

1.3 Programación Orientada a Objeto

Conceptos II

Todos los Objetos de un tipo en particular pueden recibir los mismos mensajes:

Esto es muy potente. Nos abre un montón de posibilidades para REUTILIZAR CÓDIGO.

Clase Figura

Clase Círculo

Clase Cuadrado

Clase Triángulo

Un objeto del tipo o clase Círculo también es del tipo Figura. Está garantizado que un círculo recibirá los mensajes de Figura.

Si hacemos código que habla con objetos de tipo Figura, SIN TOCAR NADA, ese código también funciona con objetos de la clase Círculo.

```
mi_figura.dibujar_en_pantalla();
```

1.3 Programación Orientada a Objeto

Conceptos III

Clases:

Una clase describe un conjunto de objetos que tienen:

Las mismas características. > Datos

Los mismos comportamientos. > Funciones

Una clase es un tipo de datos. Un entero también tiene unas características y unas funciones específicas.

Crearemos nuevas clases y el lenguaje de programación las tratará como si fueran tipos de datos propios.

Una vez definida una clase podremos crear tantos objetos como queramos.

```
Circulo    circulo1, circulo2;  
Circulo    my_nuevo_circulo;
```

1.3 Programación Orientada a Objeto

Características

Abstracción:

Nos permite representar una realidad compleja en términos de un modelo simplificado.

Encapsulación:

Nos permite ocultar cómo está construido un objeto. Otros componentes de nuestro sistema no necesitan conocer los datos y el funcionamiento interno del objeto. Datos y Funciones se empaquetan juntos. Los diferentes componentes, como no pueden acceder a sus partes internas, **se comunican mediante mensajes.**

Herencia:

Permite que un objeto incorpore todo o parte de su definición de otro objeto. Parte de la definición de Felino se corresponde con la definición de Mamífero.

Polimorfismo:

Propiedad que poseen algunas operaciones de tener un comportamiento diferente dependiendo del objeto (o tipo de dato) sobre el que se aplican

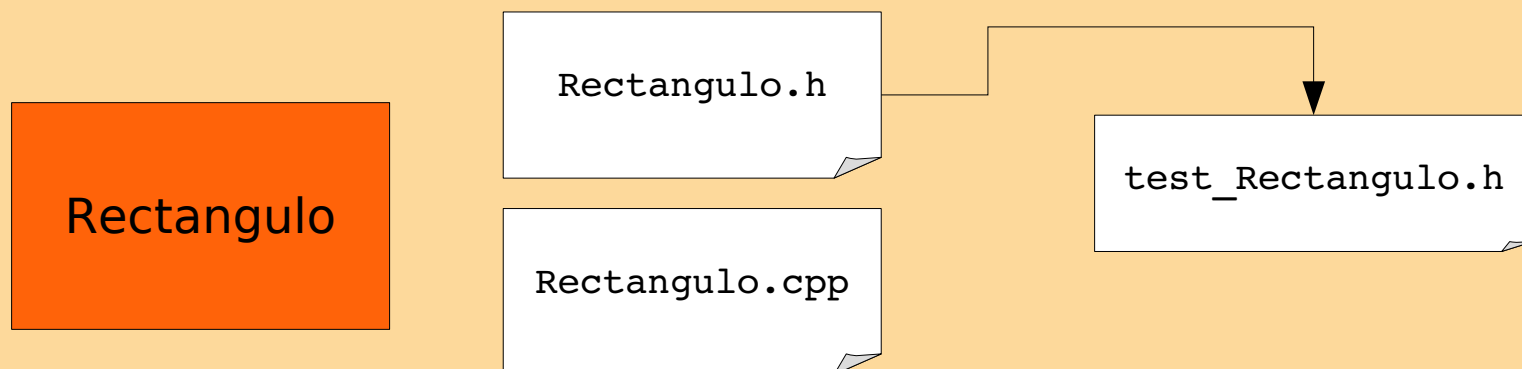
4.1 Programación Orientada a Objeto

`nombre_clase.h` contiene la definición de la clase.

`nombre_clase.cpp` contiene la definición de las funciones y operadores de la clase.

`programa.cpp` contiene un programa principal con algunas acciones sobre la clase.

Organización bastante habitual en C++



4.2 Clases

Estructuras

- Tipo de datos agregados usando otros tipos de datos.

```
struct Time
{
    int hour;
    int minute;
    int second;
};
```

- Nombres de los miembros de la Estructura
 - En el mismo **struct**: Nombres únicos y diferentes.
 - En diferentes **structs**: Pueden tener el mismo nombre
- La definición de un struct debe terminar en “;”

4.2 Clases

Estructuras

- Acceso a los miembros de una estructura:

Operador punto (.) para acceso a los miembros.

Operador flecha (->) para acceso a través de punteros.

```
cout << timeObject.hour;
```

ó

```
timePtr = &timeObject;  
cout << timePtr->hour;
```

```
Time *timePtr = &timeObject;  
timePtr->hour = 20; // Lo mismo que  
                    // (*timePtr).hour
```

4.2 Clases

Estructuras

```
// Definición
struct Time
{
    int hour;        // 0-23
    int minute;     // 0-59
    int second;     // 0-59
};

void printUniversal( const Time & );
void printStandard( const Time & );
```

```
// Uso
Time dinnerTime;
dinnerTime.hour = 18;
dinnerTime.minute = 30;
dinnerTime.second = 0;

cout << "La cena sera a las:";
printUniversal( dinnerTime );
cout << " formato universal, y ";
printStandard( dinnerTime );
cout << " en formato estandar.\n";
dinnerTime.hour = 29; // Invalido
dinnerTime.minute = 73; // Invalido

cout << "\nHora invalida: ";
printUniversal( dinnerTime );
cout << endl;
```

4.2 Clases

Estructuras

```
// Definición
struct Time
{
    int hour;        // 0-23
    int minute;     // 0-59
    int second;     // 0-59
};

void printUniversal( const Time & );
void printStandard( const Time & );
```

```
// Uso
Time dinnerTime;
dinnerTime.hour = 18;
dinnerTime.minute = 30;
dinnerTime.second = 0;

cout << "La cena sera a las:";
printUniversal( dinnerTime );
cout << " formato universal, y ";
printStandard( dinnerTime );
cout << " en formato estandar.\n";
dinnerTime.hour = 29; // Invalido
dinnerTime.minute = 73; // Invalido

cout << "\nHora invalida: ";
printUniversal( dinnerTime );
cout << endl;
```

4.2 Clases

Clases

- Modelan Objetos con
 - Atributos (Datos)
 - Comportamientos (Métodos o funciones)
- Utilizamos la palabra **class**
- Funciones Miembro
 - Métodos
 - Se les invoca para responder a mensajes.
- Especificadores de acceso a los miembros.
 - **public:**
Se puede acceder en cualquier lugar del ámbito del objeto.
 - **private:**
Sólo accesible por las funciones miembro
 - **protected:**
Sólo accesible por las funciones miembro o clases derivadas.

4.2 Clases

Clases

- Constructores

Funciones miembro especiales

Se utiliza para inicializar los datos internos.

Tiene el mismo nombre que la clase.

Se llama al crear o instanciar un objeto de la clase.

Puede haber varios constructores (¿Con el mismo nombre?)

Se utiliza la sobrecarga de funciones.

No devuelven nada

4.2 Clases

Uso de Clases

```
class Time
{
public:
    Time();                // constructor
    void setTime( int, int, int ); // Pone hora, minutos, segundos.
    void print24H();       // Imprime la hora en formato 24H
    void print12H();      // Imprime la hora en formato 12 horas (pm)

private:
    int hour;             // 0 - 23
    int minute;          // 0 - 59
    int second;          // 0 - 59
};
```

Después de la definición: Tengo un nuevo tipo de datos (extendemos el lenguaje C++) y puedo declarar objetos, arrays, punteros y referencias.

4.2 Clases

Uso de Clases

```
Time sunset;                // Objeto de tipo Time
Time arrayOfTimes[ 5 ];    // Array
Time *pointerToTime;      // Puntero
Time &dinnerTime = sunset; // Referencia
```

Funciones Miembro

Definidas fuera de la declaración de la clase:

Se utiliza el operador de “resolución de ámbito” (**::**)

Identifica de forma única a una función de una clase determinada.

Diferentes clases pueden tener funciones idénticas.

Formato:

```
Tipo_Devuelto Nombre_de_la_Clase::nombre_de_la_funcion(...arg...)
{
    ...
}
```

Es independiente del tipo de acceso (público , privado,...)

Puedo poner la implementación dentro de la declaración (**inline**)

4.2 Classes

Uso de Clases

```
// -----  
Time::Time()  
{  
    hour = minute = second = 0;  
}  
// -----  
{  
    hour = (h>=0 && h<24) ? h : 0;  
    minute = (m>=0 && m<60) ? m : 0;  
    second = (s>=0 && s<60) ? s : 0;  
}
```

```
// -----  
void Time::print24H()  
{  
    cout << setfill( '0' ) << setw( 2 )  
        << hour << ":"  
        << setw( 2 ) << minute << ":"  
        << setw( 2 ) << second;  
}  
// -----  
void Time::print12H()  
{  
    cout << (  
        (hour==0||hour==12 )?12 : hour % 12)  
        << ":" << setfill( '0' ) << setw(2)  
        < minute << ":" << setw( 2 )  
        << second  
        <<(hour<12 ? " AM" : " PM" );  
}
```

4.2 Clases

```
// Mostramos valores iniciales.
cout << " El tiempo Inicial es";
t.print24H();    // 00:00:00

cout << "\nTiempo en formato 12H ";
t.print12H();    // 12:00:00 AM
t.setTime( 13, 27, 6 );    // Cambiamos

// Mostramos nuevos valores
cout << "\n\nTiempo 24H es ";
t.print24H();    // 13:27:06
cout << "\nTiempo 12H es ";
t.print12H();    // 1:27:06 PM

t.setTime( 99, 99, 99 );    // Valores invalidos
// Salida despues de valores invalidos
cout << "\n\nHora despues de valores invalidos:"
    << "\nFormato 24H: ";
t.print24H();    // 00:00:00
```

4.2 Clases

Acceso a los Datos

A los miembros de una Clase

Por defecto el acceso es **privado** (private)

De forma explícita los podemos poner como:

private, public o *protected*.

A los miembros de una Estructura

Por defecto el acceso es **público** (public)

De forma explícita los podemos poner como:

private, public o *protected*.

Acceso a los miembros privados de una clase

Funciones tipo “get” y tipo “set”

Funciones o métodos de acceso..

4.2 Clases

Acceso a métodos o funciones

Funciones de Acceso

Públicas

Leen, asignan y muestran datos.

Chequeos de condiciones.

Funciones de utilidad.

¿Tiene sentido un método private?

¿Por qué?

4.2 Clases

Destructores

Igual que es útil tener una función que se llame antes de utilizar un objeto para inicializaciones y preparación de uso, también es útil que cuando deje de utilizarse haya una función para liberar recursos usados o dejar las cosas como estaban.

Tienen el mismo nombre la clase precedido por tilde (~)

Sin argumentos.

No puede sobrecargarse

Labores de limpieza y fin de uso.

4.3 Ejemplo completo de Clases

```
class Rectangulo    // **** VER FUENTES COMPLETAS ****
{
private:
    int  altura_; // Comentario sobre la altura.
    int  anchura_; // Comentario sobre la anchura.
    int  comprueba_valor(int valor);
public:
    void inicializar(void);
    Rectangulo();
    int  altura(void)
    {
        return altura_; // Como esta en la declaración
    } // es como si fuera "inline"
    int  anchura(void);
    int  area(void);
    void area(int& a);
    void altura(int alto);
    void anchura(int ancho);
};
```

4.2 Clases

Ventajas hasta ahora

Acceso a datos controlado
(Funciones públicas)

Funciones de inicialización finalización incorporadas
(Constructores y Destrcutores)

No tengo que pasar un puntero a
la estructura en cada función