

## PRÁCTICAS C++

- 1) **Numero (POO)**: Implementar una clase Numero, va a añadir funcionalidades a los números enteros con los métodos: esPar(), esImpar(), esPrimo(), también dispondremos de un constructor copia, un constructor por defecto y un constructor a partir de un valor entero, así como un método para recuperar el entero o modificar el entero. Después testear esta clase desde un main.
- 2) **CTime (POO)**: Implementar una clase Time que permita realizar operaciones con horas: sumar, restar, convertir a horas, convertir a segundos, se podrán construir horas a partir de hh, mm, ss. O indicando el tiempo en horas: 23.5 → 23:30:00. Tener en cuenta que no podemos tener más de 59 segundos ni 59 minutos. Queremos poder operar con la clase de esta manera:

```
Time h1(23, 56, 2);
```

```
Time h2(2.56);
```

```
Time suma, resta;
```

```
cout << h1.toString() << endl;
```

```
cout << "En horas: " << h1.toHoras() << endl;
```

```
cout << "En Segundos: " << h1.toSegundos() << endl << endl;
```

```
cout << h2.toString() << endl;
```

```
cout << "En horas: " << h2.toHoras() << endl;
```

```
cout << "En Segundos: " << h2.toSegundos() << endl << endl;
```

```
suma = h1 + h2;
```

```
cout << "La suma: " << suma.toString() << endl;
```

```
cout << "En horas: " << suma.toHoras() << endl;
```

```
cout << "En Segundos: " << suma.toSegundos() << endl << endl;
```

```
resta = h1 - h2;
```

```
cout << "La resta: " << resta.toString() << endl;
```

```
cout << "En horas: " << resta.toHoras() << endl;
```

```
cout << "En Segundos: " << resta.toSegundos() << endl << endl;
```

3) Implementar una clase **Punto2D**, nos tiene que permitir las siguientes operaciones:

- Constructor por defecto y con las dos componentes,
- Constructor copia.
- destructor,
- Métodos set y get (hacerlos inline).
- Un método para imprimir el punto por pantalla.
- multiplicar por un escalar (podemos modificar el mismo punto o devolver uno nuevo),
- saber que cuadrante ocupa según el signo de sus componentes (devolverá 1,2,3 o 4),
- calcular la distancia de un punto a otro punto:  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- sobrecargar operador +, -, =, ==
- definir 4 arrays desde lo más estático a lo más dinámico.
  - i. Matriz estática de objetos punto.
  - ii. Matriz dinámica de objetos punto.
  - iii. Matriz estática de punteros a objetos punto.
  - iv. Matriz dinámica de punteros a objetos punto.

4) **COMPOSICION:** Implementar la clase **Grado**, queremos poder ubicar cuerpos indicando su posición en grados, minutos y segundos. Las posibilidades que tenemos es que se coloquen entre 0° 0' 0" y 360° 0' 0".

Necesitamos constructores:

por defecto,

que reciba las componentes,

que se construya a partir de otro objeto Grado,

también si vienen los datos solo en grados representados por un double: 23.5 → representa 23° 30' 0" este reparto tiene que realizarlo la clase.

Cuando los grados superen los 360.0 habrá que hacer un ajuste. Por ejemplo el Grado 400° 0' 0" equivale a 40° 0' 0".

Necesitamos métodos para:

simplificar (que no se superen los 360°)

Y para traducir de double a Grado. (en un constructor)

Cuando esté implementada la clase tendremos un array con la posición de 10 **cuerpos**, dispondrán de una etiqueta string y una posición, los podemos leer de teclado y tendremos que indicar los cuerpos que forman entre ellos un ángulo de 30° 60° 90° o 120°.

Hacer un listado:

Cuerpo1	Posición1	Cuerpo2	Posición2	Angulo
Aaa	120°	BBB	180°	60°
...	...	...	...	...

Implementar los operadores ==, !=, +, -

- 5) **VECTOR**: Implementar la clase Vector que representa un array dinámico de n elementos float con los siguientes métodos:

Constructores: por defecto, con un número de elementos, a partir de un vector, y por copia.

Destructor.

Método para extraer un elemento del vector y otro para extraer la longitud.

Introducir un método clone() que nos devuelva un vector duplicado.

En otro fichero un main para probarla con una función para visualizarlo.

- 6) **Acertijos (String)**: Implementar el juego de los acertijos, el ordenador genera una palabra aleatoria utilizando el alfabeto de 4 caracteres (será una consonante, vocal, consonante y vocal). El usuario empieza a intentar adivinarla para ello utiliza el método adivina que le pasamos la secuencia que teclea el usuario y nos devuelve una cadena de la siguiente manera. Serán 4 caracteres: con X marca letras acertadas y bien colocadas, 0 una letra que está bien pero no está bien colocada y un guión medio para indicar que esa letra no pertenece a la palabra. ***Para la generación de número aleatorios utilizar la función rand() devuelve un entero aleatorio, del fichero <cstdlib>, y para inicializar el generador de números aleatorios tenemos: srand(time(NULL)); include de ctime.***

7) **Properties (Streams / Map / String)**: Implementar una clase Properties se cargará con un fichero de Texto con la siguiente sintaxis: clave=valor. Cada par se distribuye en una línea del fichero. La idea es poder almacenar dentro de nuestra clase parámetros de configuración que luego recuperaremos desde otra clase. La clase Properties dispondrá de un constructor que recibirá un fichero de propiedades, un método string getString(string clave). Se podrán modificar los valores de las propiedades para ello implementar un método putString y un método save que actualice el fichero original.

8) **ListaEnteros (Punteros, POO)** Implementar una clase que represente una lista enlazada, los datos que manejemos dentro pueden ser números. Añadir los siguientes métodos:

```
Lista();  
  
void insertar(int);  
  
bool eliminar(int);  
  
bool existe(int);  
  
bool vacia();  
  
int get(int);  
  
void set(int, int);  
  
int numeroElementos();  
  
void borrarTodos();  
  
void imprime();  
  
virtual ~Lista();
```

Probarla con un main.

9) **PlantillaLista (Template)** Generalizar la lista anterior con una plantilla para poder utilizarla con cualquier tipo de dato. Probar a crear varios tipos de lista a partir de la plantilla de tipos primitivos, objetos, etc.

10) **Figuras (Métodos Virtuales, POO)** Implementar una jerarquía de clases que nos permita representar figuras en 2D y en 3D, las operaciones que queremos tener son sencillas. Para 2D cálculo de áreas y para 3D cálculo de volúmenes. Todas las clases dispondrán de un método visualizar() que mostrará los datos de cada figura. Trabajar con Circulo, Cuadrado, Triangulo, Cubo y Cilindro. Trabajar a distintos niveles definiendo arrays de Figuras, de Figuras2D y Figuras3D.

11) **HerenciaMúltiple:** Utilizar la clase Time antes generada y crear una nueva clase Date que represente una fecha con su hora. Escribir constructores y el método toString() en la clase DateTime. Utilizar herencia múltiple.

12) **Banco (POO, vector, Stream, Herencia)** Implementar una clase Cuenta con nombre, numero, saldo y tipo de interés. Con los métodos: ingresar, retirar, consultar saldo. Controlar las cantidades a ingresar y retirar. Crear una cuenta Vivienda con un tope de años y una cantidad máxima a retirar. Agregar a la clase cuenta un método que el usuario pueda imprimir los movimientos de la cuenta (fecha, tipo movimiento [INGRESO, REINTEGRO], cantidad (indicar si es o no negativa), y Saldo actual. Los movimientos se imprimirán del último al primero (clase vector de la STL). En main, crear una cuenta y mediante un menú ofrecer las distintas operaciones. Después probar a grabar la cuenta en un fichero de texto. Al iniciar main se carga la cuenta del fichero y al terminar grabar la cuenta en el fichero. Mirar los posibles métodos que tenemos que añadir.

13) **POLIMORFISMO:** Se trata de implementar una clase Personal que gestiona todos los empleados de una empresa. Dentro de la empresa tenemos distintos perfiles: Director, Administrativo y Jefe de Proyecto. En común se almacena el nombre, apellidos, código de empresa y sueldo. A parte cada perfil añade más información, en el caso del Administrativo dispone de dos pagas extra. El jefe de proyecto, tiene su sueldo base y una parte variable: incentivos. El director tiene el sueldo base, una paga de beneficios y unos objetivos.

El cálculo del nuevo sueldo se hace según una tabla de varemos. Por ejemplo:

- **IPC: %**
- **extras: ± %**
- **objetivos: ± importe en €**
- **incentivos: ± importe en €**

. ¿Qué relaciones hay entre las clases y que tipo de relación?

**Desde main podríamos hacer algo así:**

**Personal p(10); // El número que indiquemos y sin número dejarlo a 5.**

**p.añadir(unEmpleado)**

**p.darDeBaja(codigo) // No se elimina físicamente se pone una marca. Devuelve true si lo ha encontrado.**

**p.listar() // Muestra la información de TODOS los empleados con todos sus datos. Podemos indicar los que están de baja con un \*.**

**p.subirSueldo(Tabla\_baremos)**

- 14) (**Map, String**) A partir de la clase Properties creada en el ejercicio 4, reutilizarla para poder internacionalizar una aplicación. El usuario indica el idioma y los mensajes de la aplicación y saldrán en el idioma elegido. Español / inglés.
- 15) Implementar un cajero sencillo que reproduzca las operaciones de ingresar, retirar y pedir saldo del cajero.
- 16) **Excepcion:** Definir una exception propia, implementar algún método que la lance y luego probar a capturarla. Por ejemplo en la clase persona, lanzar una excepcion si la edad de la persona no está entre 0 y 100. Implementar la clase de la excepción, asociarla al método setEdad y probarla desde main.
- 17) **Aplicación: POO.** Diseñar la factura de un operador de telefonía. Donde se registran los SMS (fecha, numero destino, hora) y las llamadas (fecha, hora de inicio, hora fin, numero destino). y los datos del usuario. Los importes se calcularán en base a unas tarifas, donde el SMS se tarifica por unidad y la llamada por SG. Al final se pueden añadir Llamadas internacionales donde se establece una cuota de roaming a parte del precio de la llamada.
- 18) **Agenda:** Definir una clase para organizar una Agenda por fecha, hora y evento.

```
CONTENIDO AGENDA:
FECHA: 20/07/2017
HORA: 09:00:00
EVENTO: Clase C++
-----
HORA: 14:00:00
EVENTO: Comida
-----
FECHA: 21/07/2017
HORA: 09:00:00
EVENTO: Clase C++
```