**ENUNCIADO PRACTICAS: POO**

1. **Numero (POO)**: Implementar una clase Numero, va a añadir funcionalidades a los número 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;

1. 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: <cmath> funciones pow y sqrt Raiz((X1 – x2)^2 – (y1 – y2)^2)
   * sobrecargar operador **+**, **-, =, ==**
   * definir 4 arrays desde lo más estático a lo más dinámico.
     1. Matriz estática de objetos punto.
     2. Matriz dinámica de objetos punto.
     3. Matriz estática de punteros a objetos punto.
     4. Matriz dinámica de punteros a objetos punto.
2. **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 0º 0’ 0” y 360º 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 23º 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 400º 0’ 0” equivale a 40º 0’ 0”.

Necesitamos métodos para:

simplificar (que no se superen los 360º)

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 30º 60º 90º o 120º.

Hacer un listado:

Cuerpo1 Posición1 Cuerpo2 Posición2 Angulo

Aaa 120º BBB 180º 60º

… … … … …

Implementar los operadores ==, ¡=, +, -

1. **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 el operador = que nos devuelva un vector duplicado.

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

1. **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.

1. **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.
2. **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.
3. **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)**

1. Implementar un **cajero** sencillo que reproduzca las operaciones de ingresar, retirar y pedir saldo del cajero. Implementación con arrays.
2. A partir de la aplicación anterior crear una clase CajeroConsola que haga de interface entre el cajero y el usuario. Mostrará un menú con las operaciones típicas de ingresar, retirar y saldo. En esta versión no es necesario hacer la parte de identificación. Más adelante se ampliará la aplicación.