Universidad de Granada
Departamento de Ciencias de la Computación
e Inteligencia Artificial

Metodología de la Programación (2014/15)
1º Grado en Ingeniería Informática (Teoría)
7 de Septiembre de 2015

DECSAI

Normas para la realización del examen: **Duración: 2.5 horas**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

Los dispositivos GPS registran posiciones periódicamente y las almacenan. La posición de un punto está determinada por tres datos: su coordenada sobre la superficie terrestre (*latitud* y *longitud*) y su *altura*. Una sucesión de puntos registrados consecutivamente forman un recorrido (en inglés, *track*). A partir de un recorrido se pueden calcular distintas medidas que permiten analizarlo, visualizarlo en mapas, utilizarlo como guía para repetir el mismo recorrido, ...

El alumno deberá implementar diferentes métodos para clases diseñadas para la gestión de recorridos.

Se proponen las siguientes clases *secundarias*:

```
class Fecha {
    private:
        int dia;
        int mes;
        int anio;
    public:
        ......
};
```

```
class Hora {
    private:
        int hora;
        int minuto;
        int segundo;
    public:
        ......
};
```

```
class Instante {
    private:
        Fecha fecha;
        Hora  hora;
    public:
        ......
};
```

```
class Punto {
    private:
        double latitud;
        double longitud;
        double altura;
    public:
        ......
};
```

Nota: Suponemos disponibles los métodos públicos *Get...* en las clases enumeradas anteriormente para obtener los valores de los campos privados. Por ejemplo, para la clase `Hora` disponemos de `GetHora`, `GetMinuto` y `GetSegundo`.

Estos tipos se utilizan para poder crear el tipo de dato *principal* que será el encargado de gestionar recorridos. La representación propuesta es la siguiente:

```
class Recorrido {
    private:
        PuntoDeRecorrido *puntos; // Puntero a un array de "num_puntos" datos "PuntoDeRecorrido"
        int  num_puntos;          // Número de puntos del recorrido
        bool activo;              // Indica si el recorrido está activo
    public:
       ......
};
```

donde el tipo `PuntoDeRecorrido` se define como sigue:

```
struct PuntoDeRecorrido {
     Punto    punto;
     Instante instante;
};
```

Al iniciar la actividad se crea un objeto de la clase `Recorrido`. En ese momento, el campo `activo` se pone a `true` y periódicamente se van añadiendo puntos -datos de la clase `PuntoDeRecorrido`- hasta que se da por finalizada la actividad. Entonces el campo `activo` se pone a `false` y se pueden calcular diferentes parámetros del recorrido.

◁ Ejercicio 1 ▷ Constructores y destructor. Método `FinRecorrido` [1.5 puntos]

- Implemente un **constructor** para la clase `Recorrido` con un parámetro de tipo booleano de manera que:
  - si es `false` se crea un recorrido *nulo* (con ningún elemento `PuntoDeRecorrido`).
  - si es `true` indicará que el objeto que se va a crear debe incluir un primer elemento de tipo `PuntoDeRecorrido`, cuyos valores se calculan a partir de las coordenadas del punto en el que se está situado y con la fecha y hora actual. Para poder obtener estos valores suponga que existen las funciones globales:
    ```
    Punto     GetPosicion (void); // (Clase Punto) Devuelve la posición actual
    Instante  GetInstante (void); // (Clase Instante) Devuelve la fecha y hora actual
    ```
  Note que en ambos casos el recorrido queda activo.
- Implemente el **destructor** de la clase `Recorrido`.
- Implemente el **método** `FinRecorrido`.
  Termina un recorrido añadiendo el último dato de tipo `PuntoDeRecorrido` con la posición e instante actual y poniendo `activo` a `false`. **Muy importante:** este método no elimina el objeto, simplemente impide la adición de nuevos puntos.

◁ Ejercicio 2 ▷ Constructor de copia y operador de asignación [1 punto]

Implemente el **constructor de copia** y el **operador de asignación** para la clase `Recorrido`.

¿Es necesario y/o conveniente implementar estos operadores para las clases *secundarias*? ¿Por qué?

◁ Ejercicio 3 ▷ Métodos `DistanciaRecorrido`, `TiempoRecorrido` y `VelocidadMedia` [1 punto]

Escribir tres **métodos** de la clase `Recorrido`:

- `DistanciaRecorrido`: calcula y devuelve la distancia total recorrida (en metros). La distancia total se calcula como la suma de las distancias entre cada par de puntos consecutivos.
- `TiempoRecorrido`: calcula y devuelve el tiempo empleado en el recorrido (en segundos).
- `VelocidadMedia`: calcula y devuelve la velocidad media de un recorrido expresada en km/hora.

Suponga que existen los métodos de las clases `Punto` e `Instante`, respectivamente:

```
double DistanciaAPunto (const Punto & otro) const;    // Distancia en metros a otro punto
double IntervaloTiempo (const Instante & otro) const; // Tiempo en segundos a otro instante
```

◁ Ejercicio 4 ▷ Almacenar y recuperar recorridos [1.5 puntos]

Un recorrido se almacena en un fichero de **texto** con el siguiente formato:

```
RECORRIDO_MP
dia_inicio mes_inicio anio_inicio
hora_inicio minuto_inicio segundo_inicio
dia_fin mes_fin anio_fin
hora_fin minuto_fin segundo_fin
num_puntos
longitud_1 latitud_1 altura_1 dia_1 mes_1 anio_1 hora_1 minuto_1 segundo_1
longitud_2 latitud_2 altura_2 dia_2 mes_2 anio_2 hora_2 minuto_2 segundo_2
......
longitud_n latitud_n altura_n dia_n mes_n anio_n hora_n minuto_n segundo_n
```

Implemente dos **métodos** en la clase `Recorrido` para la lectura y escritura, respectivamente de recorridos.

En ambos casos, los métodos reciben el nombre del archivo (el argumento formal es una cadena *clásica* –tipo C–) y devuelven un booleano que indica si ha tenido éxito.

Nota: Suponga que existen los constructores con parámetros para las clases secundarias.

◁ Ejercicio 5 ▷ Uso de la clase [1 punto]

Escriba un programa (`recorrido_mas_rapido.cpp`) que lea una serie de ficheros de recorridos e indique en cual de ellos se tiene la mayor velocidad media. Mostrará su *nombre* y el *valor* de la mayor velocidad media.

El programa debe recibir en la línea de órdenes nombres de fichero de recorrido (uno al menos).

Si alguno de los ficheros no puede procesarse, se pasará al siguiente (el programa no detiene su ejecución). Mostrar los mensajes de error adecuados a cada problema.