Relación Tema 2

Ejercicio 1

```
#include <iostream>
#include <string>

using namespace std;

/**
*
* @brief T.D.A. Servidor
*
* @brief INVARIANTE DE REPRESENTACION
*
* Para cada instancia del T.D.A. Servidor el dato miembro
* ip debe contener unicamente numeros comprendidos entre 0
* y 255. Si no resultaría una ip inválida.
*
*
* @brief FORMA DE USO
*
* #include 'servidor.h'
*
*
* @autor Juan Carlos Ruiz García
* @date Diciembre 2016
*
*/

class Servidor{
 private:
  /* OTRO TIPO REP
   unsigned int ip[4];
  */
  unsigned char ip[4];
 public:
  /**
  * @brief Constructor por defecto. Se iniciaría el vector de char
ip como 127.0.0.1
  */
  Servidor();
  /**
  * @brief Constructor con parametros.
  * @param Recibe un vector de char y se encarga de asignar cada
valor al vector de char ip
  * @pre Asumimos que el vector recibido contiene numeros y que es
de tamaño 4
```

```
  */
  Servidor(unsigned char *ip2);
  /**
  * @brief Devuelve la ip del Servidor
  * @return Devuelve un string que contiene la ip actual del
Servidor.
  */
  string getIp();
  /**
  * @brief Cambia la antigua ip del servidor por una nueva.
  * @param Vector de char que contiene la nueva ip
  */
  void setNewIp(unsigned char *new_ip);
}
```

Ejercicio 2

```
#include <iostream>
#include <string>

#include <servidor.h>

using namespace std;

/**
*
* @brief T.D.A. Subred
*
* @brief INVARIANTE DE REPRESENTACION
*
* Para cada instancia del T.D.A. Subred el mapa subr
* tiene que tener como key una ip valida con numeros
* entre 0 y 255.
*
*
* @brief FORMA DE USO
*
* #include 'subred.h'
*
*
* @autor Juan Carlos Ruiz García
* @date Diciembre 2016
*
*/

class Subred{
 private:
  /* OTRO TIPO REP
```

```
   set<unsigned char[4]>; //Ip que identifica a la subred
   vector<Servidor>; // Conjunto de servidores pertenecientes a una
subred
   int n_servidores; // Numero de servidores de la subred
  */
  map<unsigned char[4],vector<Servidor>> subr;
  int n_servidores;
 public:
  /**
  * @brief Constructor por defecto. Se iniciaría la ip de la subred
a 127.0.0.1
  * y el n_servidores a 0.
  */
  Subred();
  /**
  * @brief Constructor con parametros.
  * @param Recibe una nueva ip para crear la nueva subred.
  */
  Subred(unsigned char *ip);
  /**
  * @brief Devuelve un servidor especifico
  * @return Devuelve el servidor cuya ip es la pasada por
parametros.
  * @param Recibe una ip buscar dentro del mapa de servidores
  * @pre Asumimos que la ip que buscamos pertenece a algun servidor
  */
  Servidor getServidor(unsigned char *i);
  /**
  * @brief Añade un nuevo servidor al mapa de subred
  * @param Recibe el nuevo servidor que quiere insertarse en la
Subred.
  */
  void aniadirServidor(const Servidor &serv);
  /**
  * @brief Devuelve el numero de servidores que contiene la subred
  */
  int getNServidores();
}
```

Ejercicio 3

```
#include <iostream>
#include <string>

using namespace std;

/**
```

```
*
* @brief T.D.A. Punto Geografico
*
* @brief INVARIANTE DE REPRESENTACION
*
* Para cada instancia del T.D.A. Punto Geografico el float
* latitud tiene que tener un valor comprendido entre -90 y 90,
* y ademas el float longitud tiene que tener un valor comprendido
* entre -180 y 180. Si no las coordenadas del punto geografico
* serian erroneas.
*
*
* @brief FORMA DE USO
*
* #include 'punto_geografico.h'
*
*
* @autor Juan Carlos Ruiz García
* @date Diciembre 2016
*
*/

class PuntoGeografico{
 private:
  /* OTRO TIPO REP
  pair<float,float>; // La key seria la latitud y el valor la
longitud
  */
  float latitud;
   float longitud;
 public:
  /**
  * @brief Constructor por defecto. Se iniciaría la longitud a 0 y
la latitud a 0.
  */
  PuntoGeografico();
  /**
  * @brief Constructor con parametros
  * @param Recibe una latitud y una longitud
  * @pre Asumimos que lat está entre -90 y 90, y ademas que lon
esta entre -180 y 180
  */
  PuntoGeografico(float lat, float lon);
  /**
  * @brief Devuelve la latitud del punto geografico
  * @return Devuelve el float latitud
  */
  float getLatitud();
  /**
```

```
  * @brief Devuelve la latitud del punto geografico
  * @return Devuelve el float latitud
  */
  float getLongitud();
  /**
  * @brief Asigna una nueva longitud a un punto geografico
  * @param Recibe una nueva longitud.
  * @pre Asumimos que new_lon esta entre -180 y 180
  */
  void setLongitud(float new_lon);
  /**
  * @brief Asigna una nueva latitud a un punto geografico
  * @param Recibe una nueva latitud.
  * @pre Asumimos que new_lat esta entre -90 y 90
  */
  void setLatitud(float new_lat);
}
```

Ejercicio 4

```
#include <iostream>
#include <string>

#include "punto_geografico.h"

using namespace std;

/**
*
* @brief T.D.A. Ruta
*
* @brief INVARIANTE DE REPRESENTACION
*
* Para cada instancia del T.D.A. Ruta todos
* los puntos geograficos que forma la ruta
* deben ser correctos. Si no obtendriamos una
* ruta incoherente.
*
*
* @brief FORMA DE USO
*
* #include 'punto_geografico.h'
*
*
* @autor Juan Carlos Ruiz García
* @date Diciembre 2016
*
*/
```

```
//BASADO EN LA FORMULA DE HAVERSINE
const float RADIO_TIERRA = 6378.0F;

class Ruta{
 private:
  /* OTRO TIPO REP
  PuntoGeografico partida; // PuntoGeografico de partida
  list<PuntoGeografico>; // Lista que contiene todos los puntos geograficos de la ruta
  int nPuntos; // Numero de puntos de la ruta
  float distanciaKmRuta; // Distancia total en km de la ruta
  */
  map<PuntoGeografico,vector<PuntoGeografico>> ruta;
  int nPuntos;
  float distanciaKmRuta;
  /**
  * @brief Se encarga de calcular la distantica total de la ruta en Km utilizando
  * la formula de Haversine. Asigna dicho computo a la variable float distanciaKmRuta.
  */
  void calcularKmRuta();
 public:
  /**
  * @brief Constructor por defecto. Se iniciaría nPuntos a 0 y distanciaKmRuta a 0.
  * Se llamara al metodo
  */
  Ruta();
  /**
  * @brief Constructor con parametros. Inicializa el nPuntos a 1, la distanciaKmRuta a 0
  * y asignara al mapa ruta la clave partida.
  * @param Recibe un punto geografico que es el partida.
  */
  Ruta(const PuntoGeografico *partida);
  /**
  * @brief Constructor con parametros. Inicializa el nPuntos a n_puntos, añade al mapa la
  * clave partida y a los datos el vector de PuntoGeograficos v y calcula la distancia en km mediante
  * la funcion calcularKmRuta().
  * @param Recibe un PuntoGeografico que es el partida, un vector de puntos geograficos y el numero de puntos;
  */
  Ruta(const PuntoGeografico *partida,const vector<PuntoGeografico> &v, int n_puntos);
```

```
  /**
  * @brief Devuelve el numero de puntos geograficos que tiene la
ruta
  * @return Devuelve el int nPuntos
  */
  int getNPuntos()const;
  /**
  * @brief Devuelve el punto de inicio de la ruta, es decir el
punto de partida.
  * @ret Devuelve la clave del mapa ruta.
  */
  PuntoGeografico getPuntoPartida();
  /**
  * @brief Devuelve la distancia en KM de la ruta
  * @return Devuelve el float distanciaKmRuta
  */
  int getDistanciaKm()const;
  /**
  * @brief Añade un nuevo punto al final del vector de puntos y
vuelve a calcular la distancia en km de la ruta
  * mediante la funcion calcularKmRuta().
  * @param Recibe un nuevo punto geografico
  */
  void aniadePunto(const PuntoGeografico &p);
}
```