PRÁCTICA FINAL:
RUTAS AÉREAS

Juan Manuel Rodríguez Gómez

Doble Grado en Ingeniería Informática y Matemáticas

Estructuras de Datos

Curso 2020 – 2021

1. Implementación de las clases Punto, Ruta y Almacen_Rutas

1.1. Clase Punto

A continuación, se muestra la parte privada de la clase *Punto*:

```
private:

    double latitud;          /**< Latitud del punto */
    double longitud;         /**< Longitud del punto */
    string descripcion;      /**< Descripcion del lugar descrito por el punto */
```

donde, *latitud* y *longitud* representan las coordenadas en el mapa del punto y *descripcion* representa una pequeña explicación asociada a aquellos "puntos de interés".

Por otro lado, en la parte pública de la clase, se han implementado los siguientes métodos, donde cada método viene acompañado de sus comentarios en Doxygen, explicando principalmente qué hace cada método:

```
public:

    /**
     * @brief Constructor por defecto.
     */
    Punto();

    /**
     * @brief Constructor con parametros.
     */
    Punto(double lat, double lng, string dcn);

    /**
     * @brief Devuelve la latitud del punto del mapa.
     * @return Latitud del punto del mapa.
     */
    double GetLatitud() const;

    /**
     * @brief Devuelve la longitud del punto del mapa.
     * @return Longitud del punto del mapa.
     */
    double GetLongitud() const;

    /**
     * @brief Operador de igualdad.
     * @param p Punto del mapa a comparar.
     * @return True (1) si los puntos del mapa son iguales y False (0) si no lo son.
     */
    bool operator==(const Punto &p) const;

    friend istream &operator>>(istream &is, Punto &p);

    friend ostream &operator<<(ostream &os, const Punto &p);
```

Nótese que al final de la parte pública de la clase se observan dos funciones “amigas” de dicha clase. Dichas funciones son la sobrecarga de los operadores de entrada/salida para un objeto de la clase *Punto*:

```
/**
 * @brief Operador de entrada. Lee un punto del mapa.
 * @param is Flujo de entrada.
 * @param p Punto del mapa a leer.
 * @return Flujo de entrada "is".
 */
istream &operator>>(istream &is, Punto &p);

/**
 * @brief Operador de salida. Muestra por pantalla un punto del mapa.
 * @param os Flujo de salida.
 * @param p Punto del mapa a mostrar.
 * @return Flujo de salida "os".
 */
ostream &operator<<(ostream &os, const Punto &p);
```

1.2. Clase Ruta

La parte privada de la clase *Ruta* es la siguiente:

```
private:

    list<Punto> puntos;     /**< Lista de puntos del mapa */
```

Hemos elegido usar el tipo de dato *list* de la STL para así poder insertar puntos tanto al principio como al final de la ruta, consiguiendo así una mayor flexibilidad a la hora de añadir/eliminar puntos de la ruta.

En la parte pública de la clase, se han implementado los siguientes métodos, donde, al igual que ocurre en la clase Punto, cada método viene acompañado de sus comentarios en Doxygen:

```
public:

    /**
     * @brief Constructor por defecto.
     */
    Ruta() = default;

    /**
     * @brief Constructor de copia.
     * @param R Ruta a copiar.
     */
    Ruta(const Ruta &R);

    /**
     * @brief Operador de asignacion.
     * @param R Ruta a asignar.
     * @return *this Ruta actual tras la asignacion.
     */
    Ruta &operator=(const Ruta &R);

    /**
     * @brief Destructor.
     */
    ~Ruta() = default;

    /**
     * @brief Comprueba si la ruta esta vacia.
     * @return True (1) si la ruta esta vacia y False (0) si no lo esta.
     */
    bool Vacia();

    /**
     * @brief Añade un nuevo punto del mapa al principio de la ruta.
     * @param p Punto del mapa a añadir al principio de la ruta.
     */
    void AniadirAlPrincipio(const Punto &p);
```

```
/**
 * @brief Elimina un punto del mapa del principio de la ruta.
 */
void QuitarAlPrincipio();

/**
 * @brief Añade un nuevo punto del mapa al final de la ruta.
 * @param p Punto del mapa a añadir al final de la ruta.
 */
void AniadirAlFinal(const Punto &p);

/**
 * @brief Elimina un punto del mapa del final de la ruta.
 */
void QuitarAlFinal();

/**
 * @brief Operador de igualdad.
 * @param R Ruta a comparar.
 * @return True (1) si las rutas son iguales y False (0) si no lo son.
 */
bool operator==(const Ruta &R) const;

/**
 * @brief Inserta un nuevo punto del mapa a la ruta en la posicion apuntada por el iterador "it".
 * @param it Iterador que apunta a una posición de la lista de puntos del mapa.
 * @param p Punto del mapa a insertar en la posicion apuntada por el iterador "it".
 */
void Insertar(const list<Punto>::iterator &it, const Punto &p);

/**
 * @brief Elimina un punto del mapa de la ruta situado en la posicion apuntada por el iterador "it".
 * @param it Iterador que apunta a una posición de la lista de puntos del mapa.
 */
void Eliminar(const list<Punto>::iterator &it);
```

```
/**
 * @brief Limpia la ruta totalmente, es decir, la deja vacia.
 */
void Limpiar();

/**
 * @brief Devuelve el numero de elementos de la ruta.
 * @return Numero de elementos de la ruta.
 */
int NumeroDePuntos() const;
```

```
/**
 * @brief Devuelve un iterador que apunta al inicio de la ruta.
 * @return Iterador que apunta al inicio de la ruta.
 */
iterator begin();

/**
 * @brief Devuelve un iterador constante que apunta al inicio de la ruta.
 * @return Iterador constante que apunta al inicio de la ruta.
 */
const_iterator cbegin() const;

/**
 * @brief Devuelve un iterador que apunta al final de la ruta.
 * @return Iterador que apunta al final de la ruta.
 */
iterator end();

/**
 * @brief Devuelve un iterador constante que apunta al final de la ruta.
 * @return Iterador constante que apunta al final de la ruta.
 */
const_iterator cend() const;

/**
 * @brief Busca un punto del mapa en la ruta.
 * @param p Punto del mapa a buscar.
 * @return Iterador que apuntara al punto del mapa buscado si se ha encontrado o
 * al final de la ruta si no se ha encontrado.
 */
iterator find(const Punto &p);

friend istream &operator>>(istream &is, Ruta &R);

friend ostream &operator<<(ostream &os, Ruta &R);
```

Al igual que en la clase *Punto*, al final de la parte pública de la clase se observan dos funciones “amigas”, las cuales son la sobrecarga de los operadores de entrada/salida para un objeto de la clase *Ruta*:

```
/**
 * @brief Operador de entrada. Lee una ruta.
 * @param is Flujo de entrada.
 * @param R Ruta a leer.
 * @return Flujo de entrada "is".
 */
istream &operator>>(istream &is, Ruta &R);

/**
 * @brief Operador de salida. Muestra por pantalla una ruta.
 * @param os Flujo de salida.
 * @param R Ruta a mostrar.
 * @return Flujo de salida "os".
 */
ostream &operator<<(ostream &os, Ruta &R);
```

También se ha implementado, en la parte pública de la clase, dos clases llamadas *iterator* y *const_iterator*. Estas clases se realizan con el fin de recorrer un objeto de la clase *Ruta*, que recordemos que no es más que una lista de objetos de la clase *Punto*.

```
/**
 * @class iterator
 * @brief Clase iteradora (version no constante) creada para recorrer la ruta.
 */
class iterator {

    private:

        list<Punto>::iterator vit;

    public:

        /**
         * @brief Constructor por defecto.
         */
        iterator() = default;

        /**
         * @brief Constructor de copia.
         * @param it Iterador a copiar.
         */
        iterator(const iterator &it);

        /**
         * @brief Operador de asignacion.
         * @param it Iterador a asignar.
         * @return *this Iterador actual tras la asignacion.
         */
        iterator &operator=(const iterator &it);

        /**
         * @brief Operador de incremento prefijo.
         * @return *this Iterador actual tras el incremento.
         */
        iterator &operator++();
```

```
/**
 * @brief Operador de decremento prefijo.
 * @return *this Iterador actual tras el decremento.
 */
iterator &operator--();

/**
 * @brief Operador de igualdad.
 * @param it Iterador a comparar.
 * @return True (1) si los iteradores son iguales y False (0) si no lo son.
 */
bool operator==(const iterator &it);

/**
 * @brief Operador de desigualdad.
 * @param it Iterador a comparar.
 * @return True (1) si los iteradores no son iguales y False (0) si lo son.
 */
bool operator!=(const iterator &it);

/**
 * @brief Operador de consulta.
 * @return Elemento al que apunta el iterador.
 */
const Punto &operator*() const;

friend class Ruta;
friend class const_iterator;
```

```
/**
 * @class const_iterator
 * @brief Clase iteradora (version constante) creada para recorrer la ruta.
 */
class const_iterator {

    private:

        list<Punto>::const_iterator vit;

    public:

        /**
         * @brief Constructor por defecto.
         */
        const_iterator() = default;

        /**
         * @brief Constructor de copia.
         * @param it Iterador constante a copiar.
         */
        const_iterator(const const_iterator &it);

        /**
         * @brief Operador de asignacion.
         * @param it Iterador constante a asignar.
         * @return *this Iterador constante actual tras la asignacion.
         */
        const_iterator &operator=(const const_iterator &it);

        /**
         * @brief Operador de incremento prefijo.
         * @return *this Iterador constante actual tras el incremento.
         */
        const_iterator &operator++();
```

```
/**
 * @brief Operador de decremento prefijo.
 * @return *this Iterador constante actual tras el decremento.
 */
const_iterator &operator--();

/**
 * @brief Operador de igualdad.
 * @param it Iterador constante a comparar.
 * @return True (1) si los iteradores constantes son iguales y False (0) si no lo son.
 */
bool operator==(const const_iterator &it);

/**
 * @brief Operador de desigualdad.
 * @param it Iterador constante a comparar.
 * @return True (1) si los iteradores constantes no son iguales y False (0) si lo son.
 */
bool operator!=(const const_iterator &it);

/**
 * @brief Operador de consulta.
 * @return Elemento al que apunta el iterador constante.
 */
const Punto &operator*() const;

friend class Ruta;
```

1.3. Clase Almacen_Rutas

Finalmente, hemos implementado la clase *Almacen_Rutas*, cuya parte privada es la siguiente:

```
private:

    map<string,Ruta> rutas;     /**< Identificadores y rutas */
```

Para esta clase, hemos elegido usar el tipo de dato *map* de la STL. De esta forma, cada ruta estará identificada a una sola clave. En este caso, las claves serán datos de tipo *string*.

En la parte pública de la clase, se han implementado los siguientes métodos acompañados de su respectiva documentación en Doxygen:

```
public:

    /**
     * @brief Constructor por defecto.
     */
    Almacen_Rutas() = default;

    /**
     * @brief Constructor de copia.
     * @param Ar Almacen de rutas a copiar.
     */
    Almacen_Rutas(const Almacen_Rutas &Ar);

    /**
     * @brief Operador de asignacion.
     * @param Ar Almacen de rutas a asignar.
     * @return *this Almacen de rutas actual tras la asignacion.
     */
    Almacen_Rutas &operator=(const Almacen_Rutas &Ar);

    /**
     * @brief Destructor.
     */
    ~Almacen_Rutas() = default;

    /**
     * @brief Operador de indexacion (version no constante). Devuelve una ruta del almacen de rutas.
     * @param identificador Identificador asociado a la ruta que se quiere devolver.
     * @return Ruta asociada al identificador.
     */
    Ruta &operator[](const string &identificador);
```

```
/**
 * @brief Devuelve la ruta asociada a un identificador dado.
 * @param identificador Identificador asociado a la ruta a la que se quiere acceder.
 * @return Ruta asociada al identificador.
 */
Ruta GetRuta(const string &identificador);

/**
 * @brief Comprueba si el almacen de rutas esta vacio.
 * @return True (1) si el almacen de rutas esta vacio y False (0) si no lo esta.
 */
bool Vacio();

/**
 * @brief Añade una nueva ruta junto con su identificador asociado.
 * @param p Pair con la ruta y su identificador asociado.
 * @return Pair donde first apunta al nuevo elemento insertado y second es
 * un bool, el cual es True (1) si se ha insertado el nuevo elemento
 *  y False (0) si no se ha insertado.
 */
pair<map<string,Ruta>::iterator,bool> Aniadir(pair<string,Ruta> p);

/**
 * @brief Elimina un identificador.
 * @param identificador Identificador a eliminar.
 * @note En caso de que fuese un multimap eliminaria todos los elementos con ese identificador.
 */
void Eliminar(const string &identificador);

/**
 * @brief Limpia el almacen de rutas totalmente, es decir, lo deja vacio.
 */
void Limpiar();
```

```
/**
 * @brief Devuelve el numero de elementos del almacen de rutas.
 * @return Numero de elementos del almacen de rutas.
 */
int NumeroDeRutas() const;

/**
 * @brief Operador de igualdad.
 * @param Ar Almacen de rutas a comparar.
 * @return True (1) si los almacenes de rutas son iguales y False (0) si no lo son.
 */
bool operator==(const Almacen_Rutas &Ar);
```

```
/**
 * @brief Devuelve un iterador que apunta al inicio del almacen de rutas.
 * @return Iterador que apunta al inicio del almacen de rutas.
 */
iterator begin();

/**
 * @brief Devuelve un iterador constante que apunta al inicio del almacen de rutas.
 * @return Iterador constante que apunta al inicio del almacen de rutas.
 */
const_iterator cbegin() const;

/**
 * @brief Devuelve un iterador que apunta al final del almacen de rutas.
 * @return Iterador que apunta al final del almacen de rutas.
 */
iterator end();

/**
 * @brief Devuelve un iterador constante que apunta al final del almacen de rutas.
 * @return Iterador constante que apunta al final del almacen de rutas.
 */
const_iterator cend() const;

/**
 * @brief Busca una ruta en el almacen de rutas.
 * @param R Ruta a buscar.
 * @return Iterador que apuntara a la ruta buscada si se ha encontrado o
 *         al final del almacen de rutas si no se ha encontrado.
 */
iterator find(Ruta &R);

friend istream &operator>>(istream &is, Almacen_Rutas &Ar);

friend ostream &operator<<(ostream &os, Almacen_Rutas &Ar);
```

Como en las clases *Punto* y *Ruta*, se observa al final de la clase la sobrecarga de los operadores de entrada/salida para un objeto de la clase *Almacen_Rutas*:

```
/**
 * @brief Operador de entrada. Lee un almacen de rutas.
 * @param is Flujo de entrada.
 * @param Ar Almacen de rutas a leer.
 * @return Flujo de entrada "is".
 */
istream &operator>>(istream &is, Almacen_Rutas &Ar);

/**
 * @brief Operador de salida. Muestra por pantalla un almacen de rutas.
 * @param os Flujo de salida.
 * @param Ar Almacen de rutas a mostrar.
 * @return Flujo de salida "os".
 */
ostream &operator<<(ostream &os, Almacen_Rutas &Ar);
```

Cabe destacar que también hemos tenido que sobrecargar el operador de lectura para un tipo de dato *pair* de la STL que contiene un *string* y un objeto de la clase *Ruta*:

```
/**
 * @brief Operador de entrada. Lee un tipo de dato pair.
 * @param is Flujo de entrada.
 * @param p Pair a leer.
 * @return Flujo de entrada "is".
 */
istream &operator>>(istream &is, pair<string,Ruta> &p);
```

Como se ha hecho en la clase *Ruta*, hemos implementado, en la parte pública de la clase, dos clases llamadas *iterator* y *const_iterator* con el fin de recorrer un objeto de la clase *Almacen_Rutas*:

```
/**
 * @class iterator
 * @brief Clase iteradora (version no constante) creada para recorrer el almacen de rutas.
 */
class iterator {

    private:

        map<string,Ruta>::iterator vit;

    public:

        /**
         * @brief Constructor por defecto.
         */
        iterator() = default;

        /**
         * @brief Constructor de copia.
         * @param it Iterador a copiar.
         */
        iterator(const iterator &it);

        /**
         * @brief Operador de asignacion.
         * @param it Iterador a asignar.
         * @return *this Iterador actual tras la asignacion.
         */
        iterator &operator=(const iterator &it);

        /**
         * @brief Operador de incremento prefijo.
         * @return *this Iterador actual tras el incremento.
         */
        iterator &operator++();
```

```
/**
 * @brief Operador de decremento prefijo.
 * @return *this Iterador actual tras el decremento.
 */
iterator &operator--();

/**
 * @brief Operador de igualdad.
 * @param it Iterador a comparar.
 * @return True (1) si los iteradores son iguales y False (0) si no lo son.
 */
bool operator==(const iterator &it);

/**
 * @brief Operador de desigualdad.
 * @param it Iterador a comparar.
 * @return True (1) si los iteradores no son iguales y False (0) si lo son.
 */
bool operator!=(const iterator &it);

/**
 * @brief Operador de consulta.
 * @return Elemento al que apunta el iterador.
 */
pair<const string, Ruta> &operator*();

friend class Almacen_Rutas;
friend class const_iterator;
```

```
/**
 * @class const_iterator
 * @brief Clase iteradora (version constante) creada para recorrer el almacen de rutas.
 */
class const_iterator {

    private:

        map<string,Ruta>::const_iterator vit;

    public:

        /**
         * @brief Constructor por defecto.
         */
        const_iterator() = default;

        /**
         * @brief Constructor de copia.
         * @param it Iterador constante a copiar.
         */
        const_iterator(const const_iterator &it);

        /**
         * @brief Operador de asignacion.
         * @param it Iterador constante a asignar.
         * @return *this Iterador constante actual tras la asignacion.
         */
        const_iterator &operator=(const const_iterator &it);

        /**
         * @brief Operador de incremento prefijo.
         * @return *this Iterador constante actual tras el incremento.
         */
        const_iterator &operator++();
```

```
/**
 * @brief Operador de decremento prefijo.
 * @return *this Iterador constante actual tras el decremento.
 */
const_iterator &operator--();

/**
 * @brief Operador de igualdad.
 * @param it Iterador constante a comparar.
 * @return True (1) si los iteradores constantes son iguales y False (0) si no lo son.
 */
bool operator==(const const_iterator &it);

/**
 * @brief Operador de desigualdad.
 * @param it Iterador constante a comparar.
 * @return True (1) si los iteradores constantes no son iguales y False (0) si lo son.
 */
bool operator!=(const const_iterator &it);

/**
 * @brief Operador de consulta.
 * @return Elemento al que apunta el iterador constante.
 */
const pair<const string, Ruta> &operator*() const;

friend class Almacen_Rutas;
```

2. Ejemplo de compilación y ejecución del programa

2.1. Compilación

Para compilar el programa disponemos del siguiente *makefile*:

```
SRC = src
INC = include
OBJ = obj
BIN = bin
CXX = g++
CPPFLAGS = -Wall -g  -I$(INC) -c

# ************ Compilación ************

$(BIN)/rutas_aereas: $(OBJ)/rutas_aereas.o $(OBJ)/imagen.o $(OBJ)/imagenES.o $(OBJ)/Pais.o $(OBJ)/Paises.o $(OBJ)/Punto.o $(OBJ)/Ruta.o $(OBJ)/Almacen_Rutas.o
    echo Creando el ejecutable rutas_aereas
    g++ $(OBJ)/rutas_aereas.o $(OBJ)/imagen.o $(OBJ)/imagenES.o $(OBJ)/Pais.o $(OBJ)/Paises.o $(OBJ)/Punto.o $(OBJ)/Ruta.o $(OBJ)/Almacen_Rutas.o -o $(BIN)/rutas_aereas

$(OBJ)/rutas_aereas.o: $(SRC)/rutas_aereas.cpp
    echo Creando rutas_aereas.o
    g++ -g -c -I./$(INC) $(SRC)/rutas_aereas.cpp -o $(OBJ)/rutas_aereas.o -std=c++11

$(OBJ)/imagen.o: $(SRC)/imagen.cpp $(INC)/imagen.h
    echo Creando imagen.o
    g++ -g -c -I./$(INC) $(SRC)/imagen.cpp -o $(OBJ)/imagen.o -std=c++11

$(OBJ)/imagenES.o: $(SRC)/imagenES.cpp $(INC)/imagenES.h
    echo Creando imagenES.o
    g++ -g -c -I./$(INC) $(SRC)/imagenES.cpp -o $(OBJ)/imagenES.o -std=c++11

$(OBJ)/Pais.o: $(SRC)/Pais.cpp $(INC)/Pais.h
    echo Creando Pais.o
    g++ -g -c -I./$(INC) $(SRC)/Pais.cpp -o $(OBJ)/Pais.o -std=c++11

$(OBJ)/Paises.o: $(SRC)/Paises.cpp $(INC)/Paises.h
    echo Creando Paises.o
    g++ -g -c -I./$(INC) $(SRC)/Paises.cpp -o $(OBJ)/Paises.o -std=c++11

$(OBJ)/Punto.o: $(SRC)/Punto.cpp $(INC)/Punto.h
    echo Creando Punto.o
    g++ -g -c -I./$(INC) $(SRC)/Punto.cpp -o $(OBJ)/Punto.o -std=c++11

$(OBJ)/Ruta.o: $(SRC)/Ruta.cpp $(INC)/Ruta.h
    echo Creando Ruta.o
    g++ -g -c -I./$(INC) $(SRC)/Ruta.cpp -o $(OBJ)/Ruta.o -std=c++11

$(OBJ)/Almacen_Rutas.o: $(SRC)/Almacen_Rutas.cpp $(INC)/Almacen_Rutas.h
    echo Creando Almacen_Rutas.o
    g++ -g -c -I./$(INC) $(SRC)/Almacen_Rutas.cpp -o $(OBJ)/Almacen_Rutas.o -std=c++11
```

```
# ************ Generación de documentación ******************

documentacion:
    doxygen doc/doxys/Doxyfile

# ************ Limpieza ************

clean :
    -rm $(OBJ)/* $(SRC)/*~ $(INC)/*~ ./*~

mrproper : clean
    -rm $(BIN)/* doc/html/*
```

A continuación, se muestra cómo se compila el programa en la terminal de Linux usando el *makefile* anterior:

```
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$ make clean
rm obj/* src/*~ include/*~ ./*~
rm: no se puede borrar 'src/*~': No existe el archivo o el directorio
rm: no se puede borrar 'include/*~': No existe el archivo o el directorio
rm: no se puede borrar './*~': No existe el archivo o el directorio
Makefile:54: fallo en las instrucciones para el objetivo 'clean'
make: [clean] Error 1 (no tiene efecto)
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$ make
echo Creando rutas_aereas.o
Creando rutas_aereas.o
g++ -g -c -I./include src/rutas_aereas.cpp -o obj/rutas_aereas.o -std=c++11
echo Creando imagen.o
Creando imagen.o
g++ -g -c -I./include src/imagen.cpp -o obj/imagen.o -std=c++11
echo Creando imagenES.o
Creando imagenES.o
g++ -g -c -I./include src/imagenES.cpp -o obj/imagenES.o -std=c++11
echo Creando Pais.o
Creando Pais.o
g++ -g -c -I./include src/Pais.cpp -o obj/Pais.o -std=c++11
echo Creando Paises.o
Creando Paises.o
g++ -g -c -I./include src/Paises.cpp -o obj/Paises.o -std=c++11
echo Creando Punto.o
Creando Punto.o
g++ -g -c -I./include src/Punto.cpp -o obj/Punto.o -std=c++11
echo Creando Ruta.o
Creando Ruta.o
g++ -g -c -I./include src/Ruta.cpp -o obj/Ruta.o -std=c++11
echo Creando Almacen_Rutas.o
Creando Almacen_Rutas.o
g++ -g -c -I./include src/Almacen_Rutas.cpp -o obj/Almacen_Rutas.o -std=c++11
echo Creando el ejecutable rutas_aereas
Creando el ejecutable rutas_aereas
g++ obj/rutas_aereas.o obj/imagen.o obj/imagenES.o obj/Pais.o obj/Paises.o obj/Punto.o obj/Ruta.o obj/Almacen_Rutas.o -o bin/rutas_aereas
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$
```

2.2. Ejecución

Una vez compilado el programa, procedemos a ejecutarlo. Para ello, hemos creado el siguiente *script*:

```
#!/bin/bash
#Autor: Juan Manuel Rodríguez Gomez

./bin/rutas_aereas ./datos/paises.txt ./datos/imagenes/mapas/mapa1.ppm ./datos/imagenes/banderas/ ./datos/almacen_rutas.txt ./datos/imagenes/aviones/avion1.ppm ./datos/imagenes/aviones/mascara_avion1.pgm
```

Finalmente, ejecutamos el programa en la terminal de Linux usando el *script* anterior. Nos mostrará las diferentes rutas y tendremos que elegir una. En este ejemplo hemos elegido la ruta R2.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$ chmod +x ./ejecucion.sh
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$ ./ejecucion.sh
Las rutas:
R1 5 (34.5204,69.2008) (52.5079,13.4261) (7.40665,12.3446) (-0.186596,-78.4305) (40.4
005,-3.59165)
R2 8 (58.6954,-96) (35.0869,-103.723) (-12.0553,-77.0452) (40.4005,-3.59165) (37.9438
,104.136) (-27.7871,133.281) (35.6735,139.71) (62.8865,61.5512)
R3 5 (17.2464,-19.6706) (4.28364,-74.224) (51.5289,-0.101599) (62.8865,61.5512) (37.9
438,104.136)
R4 11 (14.4225,-87.6343) (48.8589,2.34706) (24.7259,46.8225) (58.6954,-96) (35.0869,-
103.723) (-12.0553,-77.0452) (40.4005,-3.59165) (37.9438,104.136) (-27.7871,133.281)
(35.6735,139.71) (62.8865,61.5512)
R5 5 (52.7608,8.74761) (-19.0519,29.1528) (-34.6159,-58.4333) (58.6954,-96) (52.7608,
8.74761)
Dime el codigo de una ruta
R2
Canada EstadosUnidos Peru España China Australia Japon Rusia
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$
```

Obtendremos como resultado la imagen *R2.ppm*, la cual se tiene que encontrar en el directorio */rutas_aereas* y su contenido es el siguiente:

3. Observaciones y dificultades de la práctica

Cabe destacar que no se ha modificado el código proporcionado por el profesorado para realizar la práctica, solo se ha comentado con Doxygen.

Hay métodos de las clases implementadas (la clase *Punto*, la clase *Ruta* y la clase *Almacen_Rutas*) que realmente no se acaban utilizando en esta práctica, pero se han añadido para aportar una mayor funcionalidad a las clases por si se utilizan en la realización de otro programa.

Finalmente, ha habido una dificultad a la hora de realizar la práctica. Dicha dificultad consistía en que, a la hora de ejecutar el script del programa, había veces que sí funcionaba (mostraba las rutas y pedía introducir una) y otras que no (al lanzar el comando *./ejecución.sh* no mostraba nada y se quedaba como en un "bucle infinito"). Tras ejecutar Valgrind para ver cuál podía ser el fallo y dónde se encontraba, obtuvimos lo siguiente:

```
juanma@juanma-VirtualBox:~/Escritorio/rutas_aereas$ valgrind --tool=memcheck --track-origins=yes ./bin/rutas_aereas ./datos/paises.txt ./datos/imagenes/mapas/mapa1.ppm ./datos/imagenes/ban
lmacen_rutas.txt ./datos/imagenes/aviones/avion1.ppm ./datos/imagenes/aviones/mascara_avion1.pgm ./bin/rutas_aereas ./datos/paises.txt ./datos/imagenes/mapas/mapa1.ppm ./datos/imagenes/ban
==3028== Memcheck, a memory error detector
==3028== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3028== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3028== Command: ./bin/rutas_aereas ./datos/paises.txt ./datos/imagenes/mapas/mapa1.ppm ./datos/imagenes/banderas/ ./datos/almacen_rutas.txt ./datos/imagenes/aviones/avion1.ppm ./datos/im
ascara_avion1.pgm
==3028==
==3028== Conditional jump or move depends on uninitialised value(s)
==3028==    at 0x409A19: operator>>(std::istream&, Ruta&) (Ruta.cpp:207)
==3028==    by 0x40B02F: operator>>(std::istream&, std::pair<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, Ruta>&) (Almacen_Rutas.cpp:15)
==3028==    by 0x40B7BA: operator>>(std::istream&, Almacen_Rutas&) (Almacen_Rutas.cpp:219)
==3028==    by 0x40332B: main (rutas_aereas.cpp:175)
==3028==  Uninitialised value was created by a stack allocation
==3028==    at 0x4099B5: operator>>(std::istream&, Ruta&) (Ruta.cpp:197)
==3028==
```

Nos dice que hay un salto condicional que depende de un valor no inicializado en la implementación de la sobrecarga del operador de lectura de la clase *Ruta*. La implementación de dicho operador era la siguiente:

```
istream &operator>>(istream &is, Ruta &R) {

  Ruta aux;

  int num_puntos;

  is >> num_puntos;

  Punto p;

  for(int i = 0; i < num_puntos; ++i) {
    is >> p;
    aux.AniadirAlFinal(p);
  }

  R = aux;

  return is;
}
```

Para solucionar dicho problema cambiamos la línea "*int* num_puntos" por "*int* num_puntos = 0", es decir, había que inicializar la variable de tipo entero *num_puntos*.

```
istream &operator>>(istream &is, Ruta &R) {

  Ruta aux;

  int num_puntos = 0;

  is >> num_puntos;

  Punto p;

  for(int i = 0; i < num_puntos; ++i) {
    is >> p;
    aux.AniadirAlFinal(p);
  }

  R = aux;

  return is;
}
```

Si ahora ejecutamos Valgrind, veremos que el problema anterior se ha resuelto y que el programa siempre se ejecuta perfectamente: