Trenes en Ferrovistán

En la república de Ferrovistán quieren hacer honor a su nombre construyendo una amplia red de trenes. Han decidido comenzar construyendo las vías de los trenes, formando líneas ferroviarias, para luego construir estaciones sobre la red de vías. Una misma estación puede dar servicio a varias líneas, para que los viajeros puedan hacer transbordo de una línea a otra.

Se pide implementar un TAD `Ferrovistan` con las operaciones que se describen a continuación. Todas las excepciones descritas son de la clase `domain_error`.

- `void nueva_linea(const string &nombre)`

  Añade una nueva línea al sistema ferroviario. El parámetro indica el nombre que tendrá la línea. Si ya existe una línea con ese nombre, se lanzará una excepción con el mensaje `Linea existente`.

- `void nueva_estacion(const string &linea, const string &nombre, int posicion)`

  Añade una nueva estación para que dé servicio a una `linea` dada. La estación tendrá el `nombre` indicado como segundo parámetro. El tercer parámetro (`posicion`) es la distancia que hay entre la cabecera de la línea y la estación. Si la línea no existe, se lanzará una excepción con mensaje `Linea no existente`. Si la línea pasada como parámetro ya tiene una estación con el `nombre` dado, se lanzará una excepción con mensaje `Estacion duplicada en linea`. No obstante, sí se permite tener una misma estación dando servicio a varias líneas. Si la línea ya tenía una estación en la `posicion` dada, se lanzará una excepción con el mensaje `Posicion ocupada`.

- `void eliminar_estacion(const string &estacion)`

  Elimina de la red ferroviaria la `estacion` pasada como parámetro. Esta estación dejará de dar servicio a todas las líneas en las que se encuentra. Si la estación no existe en la red, se debe lanzar una excepción con mensaje `Estacion no existente`.

- `vector<string> lineas_de(const string &estacion) const`

  Devuelve una lista de las líneas a las que da servicio la `estacion` dada. La lista devuelta debe estar ordenada de manera ascendente. Si la `estacion` no existe, se debe lanzar una excepción con el mensaje `Estacion no existente`.

- `string proxima_estacion(const string &linea, const string &estacion) const`

  Devuelve el nombre de la próxima estación que se encontraría un tren que saliese de la `estacion` dada al circular por la `linea` pasada como parámetro, suponiendo que el tren va en dirección opuesta a la cabecera de la línea. Si la `linea` no existe, se lanzará una excepción con el mensaje `Linea no existente`. Si la línea no contiene una estación con el nombre dado, se lanzará una excepción con el mensaje `Estacion no existente`. Si la `estacion` pasada como parámetro es la última de la línea, se lanzará una excepción con el mensaje `Fin de trayecto`.

Recuerda que:

1. La implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD, e indicar el coste de las operaciones.
2. Los métodos del TAD no deben mostrar nada por pantalla. El manejo de la entrada y salida de datos se realizará en funciones externas al TAD.

Entrada

La entrada consta de varios casos de prueba. Cada caso está formado por una serie de líneas, en las que se muestran las operaciones a llevar a cabo, una por cada línea: el nombre de la operación seguido de sus argumentos. La palabra `FIN` en una línea indica el final de cada caso.

Los nombres de las líneas y las estaciones son cadenas de caracteres sin espacios en blanco.

Salida

Las operaciones `nueva_linea`, `nueva_estacion` y `eliminar_estacion` no producen salida, salvo en caso de error. Con respecto a las restantes:

- Tras llamar a la operación `lineas_de XXX` debe imprimirse una línea con el texto `Lineas de XXX:` seguida de los nombres de las líneas devueltas, separados por espacios.
- Tras llamar a la operación `proxima_estacion` debe imprimirse una línea con el nombre de la estación devuelta por el método.

Si una operación produce un error, entonces se escribirá una línea con `ERROR:`, seguido del error que devuelve la operación, y no se escribirá nada más para esa operación.

Cada caso termina con una línea con tres guiones (`---`).

Entrada de ejemplo

```
nueva_linea A
nueva_linea B
nueva_estacion A Sevistan 10
nueva_estacion A Cadistan 50
nueva_estacion B Malaguistan 30
nueva_linea C
nueva_estacion C Sevistan 10
nueva_estacion C Malaguistan 20
nueva_estacion C Cordobistan 40
lineas_de Sevistan
proxima_estacion A Sevistan
proxima_estacion B Sevistan
proxima_estacion C Sevistan
eliminar_estacion Malaguistan
proxima_estacion C Sevistan
FIN
nueva_linea A
nueva_linea A
nueva_linea B
nueva_estacion A Almeristan 10
nueva_estacion A Almeristan 20
nueva_estacion A Granadistan 10
nueva_estacion A Almeristan 10
eliminar_estacion Inexistan
lineas_de Inexistan
proxima_estacion A Almeristan
proxima_estacion B Almeristan
proxima_estacion C Almeristan
proxima_estacion A Inexistan
FIN
```

Salida de ejemplo

```
Lineas de Sevistan: A C
Cadistan
ERROR: Estacion no existente
Malaguistan
Cordobistan
---
ERROR: Linea existente
ERROR: Estacion duplicada en linea
ERROR: Posicion ocupada
ERROR: Estacion duplicada en linea
ERROR: Estacion no existente
ERROR: Estacion no existente
ERROR: Fin de trayecto
ERROR: Estacion no existente
ERROR: Linea no existente
ERROR: Estacion no existente
---
```