

Práctica 6: Programas C++ con registros, vectores y ficheros de texto

6.1. Objetivos de la práctica

En esta práctica se va a trabajar con datos de tipo registro, vectores de registros y ficheros de texto.

Será necesario definir tipos de datos para representar la información almacenada en unos ficheros de texto concretos, eligiendo cómo representarlos adecuadamente. También será necesario programar una colección de funciones que faciliten el trabajo con los tipos definidos y, finalmente, diseñar un conjunto de programas C++ que hagan uso de los ficheros de texto, tanto analizando su contenido, como creando otros nuevos.

Al igual que en las restantes prácticas, a la sesión asociada a esta práctica **hay que acudir con el trabajo que se describe a continuación razonablemente avanzado** y aprovecharla con un doble objetivo:

- Consultar al profesorado las dudas surgidas y las dificultades que hayan impedido resolver satisfactoriamente los problemas planteados y, con su ayuda, tratar de aclarar ideas y avanzar en la resolución de los problemas surgidos.
- Presentar al profesorado el trabajo realizado para que este lo pueda revisar, advertir de posibles errores y defectos y dar indicaciones sobre cómo mejorarlo y, en su caso, corregirlo.

6.2. Tecnología y herramientas

6.2.1. El portal de datos abiertos Renfe Data

Renfe Operadora, a través de su portal de datos abiertos Renfe Data¹, pone a disposición del público en general (ciudadanía, empresas y otros organismos) conjuntos de datos con el objeto de fomentar la reutilización de los mismos, permitiendo que terceros puedan construir productos y servicios alrededor de los datos ofertados. Entre los conjuntos de datos publicados en Renfe Data, hay varios relacionados con los horarios de los servicios de viajeros prestados por Renfe. Estos están en formato General Transit Feed Specification² (GTFS), compuesto por un conjunto de ficheros de texto agrupados en un directorio que, en su conjunto, describen de forma precisa horarios, recorridos y tarifas de servicios de transporte público.

¹<https://data.renfe.com/>

²<https://gtfs.org/>

En esta práctica, vamos a trabajar exclusivamente con el contenido de dos ficheros que forman parte de una colección GTFS: los ficheros «stops.txt» y «stop_times.txt».

El fichero «stops.txt» contiene información sobre los lugares en los que los pasajeros de un operador de transporte público pueden acceder a un vehículo o descender del mismo. En el caso de Renfe, este fichero recoge información sobre estaciones de tren y responde a la siguiente sintaxis:

```
<estaciones> ::= <cabecera> { <estacion> }
<cabecera> ::= "stop_id,stop_name,stop_lat,stop_lon,wheelchair_boarding"
               <fin-línea>
<estacion> ::= <código> <separador> <nombre> <separador> <latitud> <separador>
               <longitud> <separador> <accesibilidad> <fin-línea>
<código> ::= literal-entero
<nombre> ::= literal-cadena
<latitud> ::= literal-real
<longitud> ::= literal-real
<accesibilidad> ::= "0" | "1" | "2"
<separador> ::= ","
<fin-línea> ::= "\n"
```

Cada fichero «stops.txt» comienza con una línea de cabecera a la que le sigue la información sobre estaciones. La cabecera es una línea de texto fijo. En los programas solicitados en esta práctica tenemos que tener en cuenta que esta línea está presente en los ficheros, pero podemos ignorar su contenido concreto por completo.

A continuación, aparece una secuencia de estaciones utilizadas por Renfe. Estas estaciones figuran cada una en una línea distinta y contienen, en este orden, información sobre el código numérico que identifica la estación, el nombre de la estación, su latitud y longitud geográficas y su grado de accesibilidad, codificado con un entero que indica que la estación es accesible (2), no es accesible (1) o que el grado de accesibilidad es desconocido (0).

El fichero «stop_times.txt» contiene información sobre las horas y lugares de las paradas programadas como parte de los distintos servicios del operador de transporte público. Responde a la siguiente sintaxis:

```
<paradas-programadas> ::= <cabecera> { <parada-programada> }
<cabecera> ::= "trip_id,arrival_time,departure_time,stop_id,stop_sequence"
               <fin-línea>
<estacion> ::= <código-servicio> <separador> <hora-llegada> <separador> <hora-salida>
               <separador> <código-estacion> <separador> <orden-parada> <fin-línea>
<código-servicio> ::= literal-cadena
<hora-llegada> ::= <hora>
<hora-salida> ::= <hora>
<hora> ::= <h> ":" <m> ":" <s>
<h> ::= literal-entero
<m> ::= literal-entero
<s> ::= literal-entero
<código-estacion> ::= literal-entero
<orden-parada> ::= literal-entero
<separador> ::= ","
<fin-línea> ::= "\n"
```

Cada fichero «stop_times.txt» comienza con una línea de cabecera a la que le sigue la información sobre paradas programadas. Como en el caso anterior, no vamos a utilizar el contenido de esta línea de cabecera, aunque tendremos que tener en cuenta que está presente en los ficheros.

A continuación, aparece una secuencia de paradas de trenes en estaciones programadas por Renfe. Estas paradas figuran cada una en una línea distinta y contienen, en este orden, el código numérico que identifica el servicio (viaje) al que pertenece la parada, la hora prevista de llegada a la estación, la hora prevista de salida de la estación, el código de la estación en la que se produce la parada y el número de orden de la parada en el servicio al que pertenece.



En esta práctica vamos a trabajar con los ficheros «stops.txt» y «stop_times.txt» correspondientes a dos colecciones GTFS proporcionadas por Renfe:

- El fichero «cercanias.zip»³, y contiene el horario de los servicios de viajeros prestados por los trenes de Cercanías y Rodalies desde el 17 de noviembre hasta el 17 de diciembre de 2024. El contenido de los ficheros «stops.txt» y «stop_times.txt» ha sido ligeramente reordenado para ajustarse a los objetivos de esta práctica.
- El fichero «ave.zip»⁴, y contiene el horario de los servicios de viajeros prestados por los trenes de Alta Velocidad, Larga Distancia y Media Distancia desde el 23 de septiembre de 2024 hasta el 13 de diciembre de 2025. El contenido de los ficheros «stops.txt» y «stop_times.txt» ha sido modificado para coincidir con la estructura de los ficheros equivalentes del fichero «cercanias.zip» y ajustarse así a los objetivos de esta práctica.

6.3. Trabajo a desarrollar en esta práctica

En esta sección se propone el desarrollo de tres programas de complejidad creciente que trabajan con datos de horarios de Renfe. Cada uno de ellos se desarrollará en un módulo principal diferente dentro de un directorio de nombre «practica6», creado en el directorio «programacion1».

En el repositorio <https://github.com/prog1-eina/practica6> tienes un área de trabajo para esta práctica, con la estructura de directorios y ficheros necesaria para que el fichero «Makefile» y las tareas de compilación, ejecución y depuración de Visual Studio Code funcionen adecuadamente. Sin embargo, a diferencia de otras prácticas, **en esta práctica la estructura de ficheros y el fichero «Makefile» no están completos. En la tarea 3 se comenta lo que hay que completar en el fichero «Makefile».**

Puedes descargar el área de trabajo (botón   de la web del repositorio), y descomprimirla en tu directorio «programacion1» como «practica6» (borra el sufijo «-main» que añade GitHub al preparar el fichero comprimido para su descarga).

En el directorio del área de trabajo «practica6» hay una carpeta denominada «data» donde se encuentran los ficheros comprimidos «ave.zip» y «cercanias.zip». Descomprímelos en los directorios «ave» y «cercanias», respectivamente, dentro del directorio «data». Asegúrate de los directorios «ave» y «cercanias» contienen los ficheros de texto pertenecientes al formato GTFS, en particular, los ficheros «stops.txt» y «stop_times.txt» con los que vamos a trabajar.

³Se puede descargar como «fomento_transit.zip» desde <https://data.renfe.com/dataset/horarios-cercanias/>

⁴Se puede descargar como «google_transit.zip» desde <https://data.renfe.com/dataset/horarios-de-alta-velocidad-larga-distancia-y-media-distancia>

6.3.1. Tarea 1. Proyecto «datos-pruebas»

Los ficheros «stop_times.txt» tienden a ser muy grandes. En el caso del fichero «stop_times.txt» del directorio «ave», contiene 58 085 líneas, mientras que el del directorio «cercanias» contiene más de 1 700 000 líneas. Este volumen puede suponer un problema a la hora de realizar pruebas con los programas que se piden en las dos tareas siguientes. Es recomendable realizar pruebas iniciales con ficheros más pequeños, de forma que sea más sencillo controlar que los resultados producidos son los esperados y con los que el tiempo de ejecución de los programas será sensiblemente menor.

En esta primera tarea, **desarrolla un primer programa que genere ficheros para hacer pruebas más pequeños que los originales**. En concreto, el programa solicitado realizará las siguientes tareas:

Tarea 1.1 Generar, en un nuevo directorio denominado «test-10» que se ubicará dentro del directorio «data», dos nuevos ficheros «stops.txt» y «stop_times.txt». El contenido del nuevo fichero «stops.txt» estará generado a partir del contenido del fichero «data/cercanias/stops.txt» y contendrá únicamente las 3 primeras líneas de este (cabecera y 2 estaciones). El contenido del nuevo fichero «stop_times.txt» estará generado a partir del contenido del fichero «data/cercanias/stop_times.txt» y contendrá únicamente las 10 primeras líneas de este (cabecera y 10 paradas programadas).

Tarea 1.2 Generar, en otro directorio nuevo denominado esta vez «test-200», dos nuevos ficheros «stops.txt» y «stop_times.txt». El contenido del nuevo fichero «stops.txt» estará generado a partir del contenido de «data/ave/stops.txt» y contendrá únicamente las 23 primeras líneas de este (cabecera y 22 estaciones). El contenido del nuevo fichero «stop_times.txt» estará generado a partir del contenido del fichero «data/ave/stop_times.txt» y contendrá únicamente las 200 primeras líneas de este (cabecera y 199 paradas programadas).

Diseña dicho programa en el fichero denominado «1-datos-pruebas.cpp».

El programa **no tiene que ser interactivo**, bastando con que realice exactamente las dos tareas enunciadas en el listado anterior sin necesidad de solicitar información adicional al usuario. En todo caso, se informará al usuario si se han generado los ficheros o no, en el caso de que se haya producido un error creando o abriendo alguno de los ficheros.

El programa tampoco tiene necesidad de conocer la estructura de cada línea del fichero: basta con que copie de un fichero a otro el número de líneas establecido, sin necesidad de procesar su contenido de acuerdo a ninguna regla sintáctica.

Al escribir el código del programa solicitado en esta sección, hay que tener en cuenta cómo se resuelven los nombres de ficheros o rutas de acceso a un fichero cuando se utilizan desde un programa:

- Una *ruta absoluta* es la especificación completa de la ubicación de un determinado fichero, partiendo desde la raíz del sistema de ficheros e incluyendo los nombres de todos los directorios que hay que atravesar hasta llegar al fichero. Por ejemplo, en Windows la ruta absoluta «C:\Users\Usuario\Documentos\Prog1\practica6\src\1-datos-pruebas.cpp» podría especificar la ubicación del fichero de código fuente en el que tienes que escribir el programa solicitado en esta tarea, mientras que en Linux una ubicación equivalente podría ser «/home/usuario/prog1/practica6/src/1-datos-pruebas.cpp».
- Una *ruta relativa* es una especificación parcial de la ubicación de un determinado fichero, partiendo desde el directorio especificado por una segunda ruta. Por ejemplo, la ruta «src\1-datos-pruebas.cpp», relativa al directorio «C:\Users\Usuario\Documentos\Prog1\practica6» representaría la ubicación del fichero de

ejemplo de Windows del punto anterior. Del mismo modo, la ruta «src/1-datos-pruebas.cpp», relativa al directorio «/home/usuario/prog1/practica6», representaría la ubicación del ejemplo de Linux del punto anterior.

Todo programa en ejecución está asociado a un directorio del sistema de ficheros, denominado *directorio de trabajo* o *directorio actual*, cuyo valor inicial se establece cuando se inicia el programa. Cuando el nombre de un fichero no se especifica a través de una ruta absoluta, por defecto se considera que el nombre de fichero o ruta son relativos al directorio de trabajo actual del programa.

En el caso de Visual Studio Code, cuando se ha abierto un determinado directorio a través de la opción **File >> Open Folder...**, el directorio de trabajo se establece en dicho directorio abierto. El fichero «Makefile» proporcionado para esta práctica y las tareas configuradas de Visual Studio Code asumen que es así, por lo que el directorio de trabajo de los programas que ejecute a través de dichas tareas será el directorio «practica6».

Dado el árbol de directorios que se propone en estas prácticas, para que el programa «datos-pruebas» acceda a los ficheros de paradas y estaciones podría utilizarse una ruta de acceso absoluta (por ejemplo, «Z:\programacion1\practica6\data\cercanias\stop_times.txt») o una ruta de acceso relativa al directorio de trabajo del programa («data/cercanias/stop_times.txt»).

Solo se recomienda la segunda opción (la utilización de rutas relativas), tanto en el programa que se pide en esta tarea, como en los siguientes. Las rutas absolutas son, en muchos casos, específicas de la configuración de un determinado equipo y contraproducentes para la experiencia de usuario con los programas. Si los programas que entregas utilizan rutas absolutas, para ejecutarlos tendríamos que ubicar tu programa en la ruta concreta que tú hubieras especificado en el código fuente (*cosa que no haremos*).

No se recomienda utilizar la extensión Code Runner de Visual Studio Code para ejecutar este programa una vez escrito, puesto que esta extensión modifica el directorio de trabajo del programa que ejecuta, lo que afecta a las rutas relativas de acceso a los ficheros que se hayan utilizado en el código del programa. Se recomienda utilizar la tarea “Ejecutar «datos-pruebas»” configurada en Visual Studio Code o compilar con la orden `make datos-pruebas` y ejecutar con la orden `bin/datos-pruebas.exe`. Cualquiera de estas dos alternativas permitirán que el directorio de ejecución sea «programacion1/practica6». Adicionalmente, la receta del fichero «Makefile» para el objetivo `datos-pruebas` incluye la creación de los subdirectorios «test-10» y «test-200» en el directorio «data».

6.3.2. Tarea 2a. Proyecto «contar-usos»

Escribe un programa que solicite al usuario el nombre de un directorio que contiene ficheros GTFS y el código de una estación y, a continuación, muestre el número de paradas programadas en la estación de código introducido según el contenido del fichero «stop_times.txt» del directorio solicitado por el usuario.

Se muestran a continuación **cuatro ejemplos de ejecución** del programa solicitado:

Escriba el nombre de un directorio: cercanias

Escriba el código de una estación: 4040

Número de paradas en la estación 4040: 2346

Escriba el nombre de un directorio: ave
Escriba el código de una estación: 60000
Número de paradas en la estación 60000: 2068

Escriba el nombre de un directorio: test-10
Escriba el código de una estación: 4040
Número de paradas en la estación 4040: 6

Escriba el nombre de un directorio: test-200
Escriba el código de una estación: 60000
Número de paradas en la estación 60000: 0

El usuario solo tendrá que escribir el nombre del directorio con cuyos datos GTFS quiera trabajar, siendo el programa el responsable de obtener el acceso al fichero «stop_times.txt» correspondiente utilizando una ruta relativa al mismo desde el directorio de ejecución del programa (mira los pasos de desarrollo más abajo).

Para desarrollar este programa, sigue los siguientes pasos:

Paso 1. Crea y completa el fichero de implementación «pedir-directorio.cpp» correspondiente al fichero de interfaz «pedir-directorio.hpp». El objetivo de la función pedirDirectorio que se declara en el fichero de interfaz es facilitar la labor de solicitar al usuario el nombre de un directorio (como «cercanías» o «ave») y convertirlo en dos rutas de acceso relativas al directorio de ejecución del programa para acceder a los ficheros «stops.txt» y «stop_times.txt».

Si utilizas Visual Studio Code, no deberías necesitar modificar el valor de las constantes RUTA_DATOS, NOM_FICH_PARADAS y NOM_FICH_ESTACIONES que se declaran en el fichero de interfaz. En otro caso, puedes modificarla si es preciso.

Paso 2. El fichero «parada-programada.hpp» define la interfaz de un módulo para trabajar con registros que representan paradas programadas. Completa en él la definición del tipo registro ParadaProgramada para que represente los siguientes datos de una parada programada:

- el código de la estación correspondiente a la parada programada
- la hora de llegada
- la hora de salida

Nota: La definición de este tipo registro no es estrictamente necesaria para la resolución de este problema, aunque facilita las operaciones de lectura de los ficheros en los programas solicitados en esta tarea y las siguientes.

Paso 3. Crea y completa el fichero de implementación «parada-programada.cpp» correspondiente al fichero de interfaz «parada-programada.hpp».

Mientras estés resolviendo esta tarea 2a, puedes olvidarte del parámetro paradasProgramadasPorHora del procedimiento contarParadasProgramadas. Lo necesitaremos para resolver la tarea 2b, pero no ahora.

Paso 4. Escribe el módulo principal del proyecto con su función `main()` en el fichero «2- contar-paradas.cpp».

6.3.3. Tarea 2b. Ampliación del proyecto «contar-usos»

Modifica el programa de la tarea anterior para que, además del número de paradas programadas en la estación de código introducido por el usuario, muestre también un desglose horario de esas paradas, según se muestra en los siguientes ejemplos de ejecución del programa solicitado:

Escriba el nombre de un directorio: cercanias

Escriba el código de una estación: 4040

Número de paradas en la estación 4040: 2346

Desglose por horas:

00:00 - 01:00	0
01:00 - 02:00	0
02:00 - 03:00	0
03:00 - 04:00	0
04:00 - 05:00	0
05:00 - 06:00	0
06:00 - 07:00	77
07:00 - 08:00	113
08:00 - 09:00	270
09:00 - 10:00	175
10:00 - 11:00	68
11:00 - 12:00	136
12:00 - 13:00	88
13:00 - 14:00	93
14:00 - 15:00	177
15:00 - 16:00	116
16:00 - 17:00	93
17:00 - 18:00	217
18:00 - 19:00	113
19:00 - 20:00	169
20:00 - 21:00	126
21:00 - 22:00	191
22:00 - 23:00	93
23:00 - 24:00	31

Escriba el nombre de un directorio: ave

Escriba el código de una estación: 60000

Número de paradas en la estación 60000: 2068

Desglose por horas:

00:00 - 01:00	11
01:00 - 02:00	0
02:00 - 03:00	0
03:00 - 04:00	0
04:00 - 05:00	0
05:00 - 06:00	0
06:00 - 07:00	69
07:00 - 08:00	104
08:00 - 09:00	115
09:00 - 10:00	248
10:00 - 11:00	129
11:00 - 12:00	103
12:00 - 13:00	68
13:00 - 14:00	60
14:00 - 15:00	103
15:00 - 16:00	174
16:00 - 17:00	95
17:00 - 18:00	205
18:00 - 19:00	142
19:00 - 20:00	164
20:00 - 21:00	103
21:00 - 22:00	75
22:00 - 23:00	61
23:00 - 24:00	39

Escriba el nombre de un directorio: **test-10**

Escriba el código de una estación: **4040**

Número de paradas en la estación 4040: 6

Desglose por horas:

00:00 - 01:00	0
01:00 - 02:00	0
02:00 - 03:00	0
03:00 - 04:00	0
04:00 - 05:00	0
05:00 - 06:00	0
06:00 - 07:00	0
07:00 - 08:00	0
08:00 - 09:00	2
09:00 - 10:00	0
10:00 - 11:00	0
11:00 - 12:00	0
12:00 - 13:00	0
13:00 - 14:00	0
14:00 - 15:00	0
15:00 - 16:00	2
16:00 - 17:00	0
17:00 - 18:00	0
18:00 - 19:00	0
19:00 - 20:00	0
20:00 - 21:00	1
21:00 - 22:00	1
22:00 - 23:00	0
23:00 - 24:00	0

Escriba el nombre de un directorio: **test-200**

Escriba el código de una estación: **60000**

Número de paradas en la estación 60000: 0

Observa que, cuando no hay paradas programadas en la estación cuyo código ha escrito el usuario, el programa no muestra el desglose horario.

Para desarrollar este programa, sigue los siguientes pasos:

Paso 1. En el fichero de implementación «parada-programada.cpp», encárgate ahora de rellenar convenientemente las componentes del vector `paradasProgramadasPorHora` del procedimiento `contarParadasProgramadas`.

Ten en cuenta al implementar que en el fichero «stop_times.txt», cuando un tren que ha iniciado su recorrido en las últimas horas del día y tiene paradas programadas pasada la medianoche, puede tener expresada su hora de llegada y de salida con horas iguales o superiores a 24 (por ejemplo, 24:15:00). Puedes utilizar una expresión aritmética muy simple para que esto no suponga un problema.

Paso 2. Modifica el módulo principal del proyecto en el fichero «2- contar-paradas.cpp» para mostrar el desglose horario.

6.3.4. Tarea 3. Proyecto «usos-por-usuario»

En esta tarea se pide escribir un programa que, al igual que el del anterior, **solicite al usuario el nombre de un directorio que contiene ficheros GTFS y, a continuación, escriba en la pantalla el número de estaciones presentes en el fichero «stops.txt» y un listado con las 15 estaciones que más paradas programadas tengan.** Para cada una de estas estaciones, indicará el número de paradas programadas, el código de la estación y el nombre. Este listado de 15 estaciones deberá aparecer **ordenado de mayor a menor número de paradas programadas.**

Se muestran a continuación dos ejemplos de ejecución del programa solicitado:

Escriba el nombre de un directorio: cercanias

Número estaciones: 973

Paradas	Código	Nombre de la estación
=====	=====	=====
29203	18000	Estación de tren Madrid - Atocha Cercanías
27632	17000	Estación de tren Madrid-Chamartín-Clara Campoamor
21463	18002	Estación de tren Madrid-Nuevos Ministerios
20171	71801	Estación de tren Barcelona-Sants
14928	37001	Estación de tren Madrid-Villaverde Alto
12163	72305	Estación de tren L'hospitalet De Llobregat
12162	78804	Estación de tren Barcelona-Arc De Triomf
12162	78805	Estación de tren Barcelona-Plaça De Catalunya
11960	18003	Estación de tren M.alvaro-P
10949	18101	Estación de tren Madrid-Sol
10669	60100	Estación de tren Madrid-Villaverde Bajo
10501	18001	Estación de tren Madrid-Recoletos
8997	79009	Estación de tren Barcelona-El Clot
8941	13200	Estación de tren Bilbao-Abando Indalecio Prieto
8484	78802	Estación de tren Barcelona-Fabra I Puig

Escriba el nombre de un directorio: ave

Número estaciones: 793

Paradas	Código	Nombre de la estación
2068	60000	Estación de tren Madrid-Puerta de Atocha
1618	71801	Estación de tren Barcelona-Sants
1415	17000	Estación de tren Madrid-Chamartin
1211	10600	Estación de tren Valladolid
1178	4040	Estación de tren Zaragoza-Delicias
801	50500	Estación de tren Cordoba
767	78400	Estación de tren Lleida
712	60911	Estación de tren Alicante/alacant
697	4104	Estación de tren Camp Tarragona
664	14100	Estación de tren Palencia
612	8004	Estación de tren Segovia Guiomar
538	3208	Estación de tren Cuenca Fernando Zobel
533	60600	Estación de tren Albacete-Los Llanos
513	65000	Estación de tren Valencia-Estacio del Nord
499	51003	Estación de tren Sevilla-Santa Justa

Como en el programa solicitado en la tarea 2, **el usuario solo tendrá que escribir el nombre del directorio que contiene los datos GTFS, siendo el programa el responsable de obtener las rutas de acceso relativas a los ficheros «stops.txt» y «stop_times.txt».**

Para desarrollar este programa, sigue los siguientes pasos:

- Paso 1.** La infraestructura para resolver este problema está incompleta en el repositorio de GitHub que os hemos suministrado: faltan algunos ficheros de implementación y el fichero «Makefile» no está completo.
- Crea en el directorio «src» los dos ficheros que faltan, denominados «3-paradas-por-estacion.cpp» y «estacion.cpp». En ellos escribirás, en los siguientes pasos, el código necesario para resolver este problema.
- Paso 2.** Completa el fichero «Makefile» modificando el contenido de la regla usos-por-usuario para que compile un ejecutable a partir de los ficheros objeto resultantes de los distintos módulos de los que depende este programa:
- «3-paradas-por-estacion»
 - «parada-programada»
 - «estacion»
 - «pedir-directorio»
- Añade también la regla que falta para que el módulo «3-paradas-por-estacion» se compile individualmente.
- Ya hay reglas para la compilación de los módulos que escribiste en la tarea 2 («parada-programada», «estacion» y «pedir-nombre-fichero»). Puedes basarte en ellas para escribir las reglas que faltan.
- Paso 3.** En el fichero de interfaz «estacion.hpp», define los campos del tipo registro Estacion para que represente los siguientes datos de una estación: su código numérico, nombre, latitud, longitud, accesibilidad y número de paradas programadas.

Paso 4. Escribe el código del fichero de implementación «estacion.cpp».

A diferencia del caso del tipo registro ParadaProgramada, para resolver el problema que se plantea en esta tarea, la definición del tipo Estacion sí que es necesaria, ya que para determinar el número de paradas programadas en una estación va a ser necesario trabajar con vectores de registros de tipo Estacion.

Paso 5. Escribe el módulo principal del proyecto con su función main en el fichero «3-paradas-por-estacion.cpp».

Se recomienda la aplicación de la metodología de diseño descendente a la hora de escribir este programa y la minimización del esfuerzo de desarrollo, haciendo un uso adecuado de las funciones que los módulos «parada-programada» y «estacion» implementan.

Va a ser necesario trabajar con un vector de registros de tipo Estacion en el que ir recogiendo, para cada estación, la información que se vaya extrayendo de los ficheros «stops.txt» y «stop_times.txt». En concreto, la información sobre código numérico, nombre, latitud, longitud y accesibilidad se va a obtener directamente del fichero «stops.txt», mientras que la relativa al número de paradas programadas se extraerá procesando el fichero «stop_times.txt».

Es razonable descomponer el problema que tiene que resolver el programa en las siguientes funciones:

- a) Un procedimiento que, dado el nombre de un fichero de texto de estaciones, copie el contenido del mismo en un vector de registros de tipo Estacion. Tienes definida su cabecera en el módulo «estacion»: leerEstaciones.
- b) Una función que, **dados un vector de registros de tipo Estacion, el número de estaciones presentes en el fichero «stops.txt» y el código de una estación, busque la estación identificada por ese código**. La función debería devolver el índice que ocupa en el vector. Puedes aplicar el esquema de búsqueda lineal en un vector con garantía de éxito, puesto que todos los códigos de estación que aparecen en el fichero «stop_times.txt» se corresponden con una estación presente en el fichero «stops.txt».
- c) Un procedimiento denominado obtenerParadasPorEstacion que, **dada la ruta de acceso relativa al fichero «stop_times.txt», almacene en un vector de registros de tipo Estacion el resultado de calcular el número de paradas programadas en cada estación**.
- d) Un procedimiento que **ordene el contenido del vector de registros de tipo Estacion**. Puedes aplicar el esquema de ordenación por selección directa visto en clase, adaptándolo al hecho de que no va a ser necesario tener ordenado todo el vector, sino solo las 15 estaciones con más paradas programadas.
- e) Un procedimiento que **escriba en la pantalla los resultados**.

La descomposición anterior es una mera sugerencia, pero en todo caso, es importante descomponer el problema que se plantea y resolverlo apoyándose en funciones auxiliares.

Para facilitar la comprobación de que los resultados obtenidos por el programa, a los ejemplos de ejecución suministrados al comienzo de esta sección, se añaden los correspondientes a los directorios «test-10» y «test-200.txt», generados como resultado de la tarea 1:

Escriba el nombre de un directorio: **test-10**

Número estaciones: 2

Paradas	Código	Nombre de la estación
6	4040	Estación de tren Zaragoza-Delicias
3	10206	Estación de tren Santa María De La Alameda-Peguerinos

Escriba el nombre de un directorio: **test-200**

Número estaciones: 22

Paradas	Código	Nombre de la estación
20	17000	Estación de tren Madrid-Chamartin
20	18000	Estación de tren Madrid - Atocha Cercanias
12	30000	Estación de tren Monfrague
12	35001	Estación de tren Leganes
12	35105	Estación de tren Torrijos
12	35200	Estación de tren Talavera de La Reina
12	35203	Estación de tren Oropesa de Toledo
12	35206	Estación de tren Navalморal de La Mata
12	35400	Estación de tren Caceres
12	37500	Estación de tren Merida
12	37606	Estación de tren Badajoz
8	50300	Estación de tren Linares-Baeza
8	60400	Estación de tren Alcazar de San Juan
6	37603	Estación de tren Montijo
5	56004	Estación de tren Jodar-Ubeda

6.3.5. Tarea 4 (Opcional). Mejoras de los proyectos anteriores

Mejora en el proyecto «contar-usos»

En el proyecto «contar-usos» se solicita al usuario el código de una estación y, cuando se muestran los resultados, solo se muestra el código de esa estación. Desde el punto de vista de la usabilidad, el programa se puede mejorar indicando en los resultados el nombre de la estación cuyo código ha escrito el usuario e informando, cuando el código introducido por el usuario no se corresponda con una estación del fichero «stops.txt», de esta circunstancia.

Modifica el código de «2- contar-paradas.cpp» para que contemple las consideraciones anteriores. Se muestran a continuación dos trazas de ejecución del programa con dicha mejora incorporada:

```
Escriba el nombre de un directorio: cercanias
Escriba el código de una estación: 4040

Número de paradas en Estación de tren Zaragoza-Delicias: 2346
Desglose por horas:
00:00 - 01:00      0
01:00 - 02:00      0
...
22:00 - 23:00      93
23:00 - 24:00      31
```

(Se ha omitido en esta traza el desglose completo horario. El programa mejorado debe mostrar el desglose completo.)

```
Escriba el nombre de un directorio: ave
Escriba el código de una estación: 16

No existe la estación de código 16.
```

Si implementas esta mejora, contribuirá a la calificación del apartado «Pruebas con carácter voluntario» establecido en la guía docente de la asignatura.

Mejora en el proyecto «usos-por-usuario»

Dependiendo de la potencia del computador en el que estés desarrollando, puedes haber notado que el tiempo de ejecución del programa «usos-por-usuario» con los datos del directorio «cercanias» es elevado. Esto es debido a que, al tener el fichero «stop_times.txt» más de un millón de líneas, la búsqueda lineal que se sugiere implementar en el punto a) del paso 5 de la sección anterior es lenta.

Puedes mejorar la eficiencia del programa «usos-por-usuario» sustituyendo esa búsqueda lineal por una búsqueda dicotómica. Para ello, va a ser necesario que el vector de estaciones esté ordenado inicialmente por un criterio favorable para la búsqueda de una estación a partir de un código de estación: el vector de estaciones tiene que estar ordenado por códigos crecientes de estación mientras estés procesando el contenido del fichero de paradas programadas.

Para ello, después de leer el fichero de estaciones, vas a tener que ordenar el vector resultante por códigos de estación crecientes.

Si realizas esta parte y consigues realmente disminuir el tiempo de ejecución del programa aplicando búsqueda dicotómica, indícalo haciendo que tu programa escriba el mensaje **"RESUELTO_DE_FORMA_EFICIENTE_CON_BÚSQUEDA_DICOTÓMICA"**. Al corregirlo, lo revisaremos y, si la implementación es correcta, contribuirá a la calificación del apartado «Pruebas con carácter voluntario» establecido en la guía docente de la asignatura.

6.4. Entrega de la práctica

Antes del **sábado 14 de diciembre a las 18:00**, se deben haber subido a Moodle los siguientes ficheros:

- «parada-programada.hpp» y «parada-programada.cpp»
- «estacion.hpp» y «estacion.cpp»
- «pedir-directorio.cpp»
- «1-datos-pruebas.cpp»
- «2-contar-paradas.cpp»
- «3-paradas-por-estacion.cpp»

En esta práctica se admitirán entregas no completas, pero, en todo caso, debe tenerse en cuenta que al evaluar la misma, el programa solicitado en la tarea 3 tendrá un peso igual o superior al 50 % en la valoración de la práctica.