**Train route finder**

You are to develop a train route finder program in C++.

The program is for a company which is in charge of five train stations in central England and users need to find journey times, routes (i.e., which stops link these five stations), and costs for journeys between the five stations. Information on routes (stops between stations) can only be added, or loaded from file, by an administrator when the program is running.

The following table provides the station names, costs and journey times for the journeys (first value cost in £, second value travel time in minutes; rows indicate the origin and columns the destinations: a journey from Leicester to Derby hence costs £13.00 while the return journey costs £13.70):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Train Stations** | Leicester | Loughborough | Nottingham | Derby | York |
| Leicester | - | 2.50/10 | 3.50/30 | 13.00/48 | 23.50/65 |
| Loughborough | 2.50/10 | - | 1.50/15 | 2.25/23 | 11.50/60 |
| Nottingham | 3.50/30 | 1.50/15 | - | 2.50/12 | 11.50/40 |
| Derby | 13.70/48 | 2.00/25 | 2.50/10 | - | 11.20/45 |
| York | 22.20/70 | 12.00/60 | 11.20/40 | 11.20/45 | - |

Your program should have an interface with seven options: Time, Price, Route, Split Ticket, Sort Routes, Admin and End (which terminates the program). The Admin option takes the user to another (Administrator) interface (no login/password is required for entering the Admin menu) with the following four options: Input route; Save route; Retrieve route; Exit (back to the previous menu).

Functionality for each of the menu options is explained in the following:

* Time: Provides the time of travel between two stations presented in hours and minutes.
* Price: The cost of travelling between any two of the five stations. Note, that if the user is travelling on the last day of the month, the fare should be reduced by 10% (and a message displayed to say that this has been done). This is a discount scheme offered by the train company. The system will therefore need to ask the user the date on which they are travelling. While dates should be entered as numbers (e.g. 26/5/2015), the output should be more verbose with the name of the month written out (e.g. 26 May 2015).
* Route: Displays a list of stops that are made between two stations. These stops are added by using the Admin option menu or retrieving the information from a previously saved data file. When the program is first run, it should check for a default data file containing route information; if such a file does not exist, there will not be any stops between stations.
* Split Ticket: As you can see from the table, e.g. going from Leicester to Loughborough and from there onwards to Derby is cheaper than going from Leicester to Derby directly. This option should hence allow the user to search for such cheaper journeys by splitting it into two separate journeys with a combined lower price. Note, that in here you are expected to arrive at such splits programmatically and not by reading off such splits from the provided table and hardcoding them in the application.
* Sort Routes: Should list all (20) routes sorted by a criterion specified by the user. The program should allow sorting by price, time, and number of stops en route.
* Input route: Allows the administrator to add as many stops as they like between any two stations. For example, one might add ‘Barrow on Soar’, ‘Rothley’, and ‘Quorn’ as stops between Leicester and Loughborough. You could ask the user how many stops they wish to add before they enter each of the stops in turn.
* Save route: Save all the information on journey stops (i.e. all stops for all routes) to a file in a format of your choosing (you do not need to save costs and journey times as they are fixed and we assume that although we can add stops this does not affect the overall journey time). The Admin should be given the option to save this information either in the default data file used by the program or in another route information file.
* Retrieve route: Read in all the information on journey stops from a user selected file (a file saved using the previous option). This should overwrite any existing data in the program when it is running.  Make sure that your program is able to cope appropriately with unexpected input (e.g. a user trying to find the cost of travel from station A to station A, or entering an invalid date), and that all the output is clear and neatly formatted.