Project 2 – Programming Project

CS3330 Data Structures and Algorithms
Term 5 2017: May 29 – July 30
Dr. Jack Davault

Overview

This assignment consists of implementing an application using the techniques learned in the first half of the course. Examples on how to use file operations, the standard template library, and other features are provided in the *Sample Programs (for Project 2)* section located the *Learning Modules* area in Blackboard. These examples provide a starting point for your implementation. Everything you need to do this project is provided in these sample programs.

Your program must compile and you must provide all of the source code files so that I can also compile and run your program (i.e. provide me with all files ending in **.h** and **.cpp** that are necessary to compile your program). You are responsible for submitting your program correctly. Please ask if you have questions.

You may use any resources, books, or notes. **Apart from your textbook and provided examples, you must mention all resources that you use as comments next to the applicable lines of code or the appropriate functions within your source code or within the main comment at the top of your source code file.** You may also work with only one other classmate on this assignment if you want to, but you must mention all of your names as comments in the applicable sections of the source code showing where you collaborated. If you collaborate on the entire project, no problem, both of your names must be in the comment at the top of all of the source code files, and only one person needs to submit the assignment. **I do highly recommend that you work with one other person on this project, and begin working on this assignment as soon as possible. Don't wait until the last minute to begin working on this assignment.** Consider posting a message in the *Student Lounge* or sending out an email to your fellow classmates asking if anyone is willing to partner with you on this project. Two person teams only.

Note: Again, you may use outside resources or books, but you are not to post questions, comments, or source code related to this assignment in any public open forum other than the ones specified for this course in Blackboard (e.g. *Ask the Instructor* or *Student Lounge*). Also, note that pay for websites for programming solutions are not an acceptable method for completing this assignment. If you have a question, don't sit on it for too long; ask me in the *Ask the Instructor* forum or via e-mail. I am also available by appointment via *Blackboard IM*.

This assignment weighs 15% of your final course grade. When submitting your work, be sure to Zip up all source code files and documentation (if any) into a single file.

Implementation (20 points)

TrainSchedule.com has contracted you to implement a light rail scheduling system. The company will consider your program as a proof-of-concept prototype for future work on a larger system. The program will provide a simple text based interface that will allow the user to enter attributes associated with a given train schedule. The program will manage these attributes in an object, which will consist of an individual node within a single linked list stored in memory. The linked list can be a global variable for simplicity, as done in the Book Inventory sample program. The program must use the "list" API in the C++ standard template library (STL). The program must implement at least one class that will hold the following variables (input validation is only required where indicated):

- `scheduleId –` An integer variable to hold the schedule identifier. This must be a randomly generated 10-digit number. The `scheduleId` is a unique identifier for each train schedule in the linked list and repeats are not allowed.

- `scheduleDate -` A string variable to hold the date that the schedule will take effect. The date must be in the form **mm/dd/yyyy**. Where **mm** is the two-digit month, **dd** is the two-digit day, and **yyyy** is a four-digit year.
  Input Validation Required: The program must re-prompt the user to re-enter the date if it is not the valid format provided above.

- `scheduleDays -` An integer variable to hold the number of days that the schedule will be valid.
  Input Validation Required: The program must re-prompt the user to re-enter the number if it is not greater than 0.

- `trainNumber -` An integer variable to hold the train number.

- `originStation –` A string variable to hold the name of the station of origin. Note that spaces must be allowed in the string.

- `origDepartureTime -` A string variable that will hold the time that the train will depart from the station of origin. This variable must be in the form **HHMM**, where **HH** is a number from 00-23 and **MM** is a number from 00-59.
  Input Validation Required: The program must re-prompt the user to re-enter the time if it is not the valid format provided above.

- `destinationStation –` A string variable to hold the name of the destination station. Note that spaces must be allowed in the string.

- `arrivalTime -` A string variable that will hold the time that the train will arrive that the destination station. Input validation is recommended but is not required. This variable should be in the form **HHMM**.

- `departureTime -` A string variable that will hold the time that the train will depart from the station. Again, input validation is recommended but not required here. This variable should be in the form **HHMM**.

Provide the appropriate methods to set and get the data for each of these class variables. For example `setScheduleDate(string scheduleDate)` and `string getScheduleDate()`. The Book Inventory provides an example on how this can be done. In addition, the main program must provide the following functionality:

1. When the program is first started, it must read a data file called **schedule.dat**. The program will not prompt the user for the name of the data file. The name of the file must be hard-coded in the program. If the file exists, the program will load the data for each schedule into the global linked list. If the file does not exist, the program will start with an empty linked list.

2. The program will provide a simple text-based user interface that manages the all of the train schedules within a linked list. Each schedule must be placed in the linked list as an object that holds all of the attributes associated with it as mentioned above. The user interface will allow the user to perform the following:

(a) Enter Schedule – allows the user to enter all of the fields associated with a given train schedule, except for the `scheduleId`, which will be automatically generated by the program as previously mentioned. After the fields are entered, the program will place the schedule object in the global linked list.

(b) Display all Schedules – displays all of the schedules within the linked list along with their associated fields. In addition, this option must print the total number of schedules in the linked list.

(c) Search for Schedule – allows the user to find a schedule by its schedule identifier. The program will prompt the user to enter the `scheduleId` and will display all of the fields associated with the given schedule, if it is found. The program must display an error message if the schedule is not found in the linked list.

(d) Edit Schedule – allows the user to edit the fields for a given schedule that is in the linked list. The program must prompt the user to enter the `scheduleId` as the key to find the schedule to edit. Print a message if the schedule is not found in the linked list. For simplicity, the program may re-prompt the user to re-enter all of the fields associated with the given schedule; however, it must reuse the `scheduleId` value.

(e) Delete Schedule – allows the user to delete a schedule from the linked list using the `scheduleId` as the key. The program must display a message if the provided `scheduleId` does not find an associated schedule in the linked list.

(f) Exit System – before the program exits, it must save all of the data in the linked list to the data file. I recommend using a standard text file with one field in the object per line. At this point, if the file does not exist, the program will create it.

Hints and Tips

The Book Inventory program provides a good place to start and I recommend using it as a model. It provides a simple text based interface, shows how to create a class, and how to use the C++ STL library's linked list API for adding and displaying items in a linked list. You should rename its files, the class, and methods as a starting point for your program.

Start by breaking the program down into small pieces. First, work on the feature that allows you to enter a schedule along with all of its fields. Next, work on the display feature that will show all of the schedules and their associated fields contained within the linked list. Next, make sure your program can read and write one schedule and its fields to and from the data file. Then expand the program to read and write all of the schedules to and from the data file. The data file functions for this can be modeled after those in File Operations sample program, and the display and enter functions that you will create. Finally, work on the search, modify, and delete features. Be sure to review the sample code in the *Sample Programs (for Project 2)* section in the *Learning Activities* area in Blackboard.

There are no tricks or special techniques required for this assignment. Everything you need to successfully create this program is available in the provided sample code.

Other Comments

Please do not hesitate to let me know if you have questions in the *Ask the Instructor* forum or via e-mail. I also encourage you to discuss this project among yourselves in the *Student Lounge* area.