Project 4: Gate Control

This project implements software to control access to a company’s facility. The company has several buildings and a parking lot for its staff, all surrounded by a fence that has an electrically-operated gate for employees to drive through if they have an authorized access card. The gate has a card reader that is connected to a computer that will receive a card’s identifying number, validate the access, and operate the gate operator’s motor to open the gate. The computer will record each transaction - the date and time of each access, allowed or denied.

**The GateControl project involves the use of the C++ Standard Library map for the software’s authorization table.** Each table record is identified by a four-digit card number and contains the cardholder’s name and the permissible access. A cardholder’s access can be restricted to a specific time range such as 8 AM to 5 PM.

The GateControl software is surrounded by other software, outside the scope of this project, that provides functions such as managing the authorization database, checking the transaction log, interfacing to the card reader, and operating the motor that opens the gate. This project’s main.cpp simulates this other software, calling GateControl functions to determine if a card that has been read is authorized, to retrieve authorization records, or to retrieve transaction records.

For GateControl.hpp:

**Data members are already provided.**

**map<CardNumber, Authorization> authorizationMap\_;**

**vector<Transaction> transactionVector\_;**

You should complete the following eight functions:

1. **accessAllowed**, whose input argument is a card number. It validates access and returns *true* if access is permitted and *false* if it is not. The attempted access, allowed or denied, is recorded in a transaction log. The format of a transaction record is defined in GateControl.hpp.
2. **addAuthorization**, whose input arguments are a card number, a cardholder name, and a time range (a start time / end time pair). It returns a boolean success/failure status. Failure (false) means the item was already present and couldn’t be added. If the item was already not present, this function should add the item to the map and return success(true).
3. **deleteAuthorization**, whose argument is a card number. It returns a boolean success/failure status. Failure (false) means the card was not found and couldn’t be deleted. Success (true) means the card was found and is deleted by the function.
4. **changeAuthorization**, whose arguments are a card number, a cardholder name, and a time range. It returns a boolean success/failure status; failure (false) means the card was not found and couldn’t be changed. When the card is found, the function changes the name and time range of the existing card.
5. **getAllAuthorizations**, which has one argument: the address of a vector that stores all of the authorization records. The function doesn’t return anything. The vector should be cleared if there are no authorization records. If there are authorization records: the vector should be filled by all authorization records. The format of an authorization record is defined in GateControl.hpp.
6. **getCardAuthorization**, whose one of the arguments is a card number. It has another argument: the address of an authorization record. It returns a boolean success/failure status; failure (false) means the card was not found. On success, true is returned. The authorization record is filled corresponding to the card number. The format of an authorization record is defined in GateControl.hpp.
7. **getAllTransactions**, which has one argument: the address of a vector to receive the complete set of transaction records. The format of a transaction record is defined in GateControl.hpp. If there are no transactions, the vector will be cleared.
8. **getCardTransactions**, whose one of the arguments is a card number. It has another argument: the address of a vector to receive transaction records for the specified card. The format of a transaction record is defined in GateControl.hpp. If there are no transactions, the vector will be cleared. Else, vector will be filled with the transactions for the corresponding card number.

### 

### Date and Time

Date and time are needed by the **accessAllowed** function to determine whether access should be allowed or denied and to mark transaction records.

It will be available through two global string variables that the Main.cpp test driver will set as needed to test the GateControl functions. These strings have these formats:

* Date: YYYYMMDD, where YYYY is a four-digit year, MM is a two-digit month, and DD is a two-digit day.
* Time: HHMM, where HH is a two-digit hour from 00 to 23 and MM is a two-digit minute.

These formats allow dates and times to be compared by simply using the standard operators such as ==, <, and > on the strings. There is no need to extract fields from the strings and compare the individual parts of dates or times.

Each test will be defined to occur at a specific date and time; having main.cpp set these times makes the tests repeatable and not dependent on the actual time-of-day.

### 

### Hints

You will find it helpful to implement GateControl’s functions in small steps, building up from simple functions to more complex ones, testing each one as you go.

Here’s a suggested order:

1. addAuthorization
2. getAllAuthorizations
3. deleteAuthorization
4. changeAuthorization
5. getCardAuthorization
6. accessAllowed
7. getCardTransactions
8. getAllTransactions

### 

In main.cpp testing can also be done in the following suggested order:

1. TestAdd
2. TestDelete
3. TestChange
4. TestCard
5. TestAccess

First test TestAdd by commenting out all others in the main function. Then uncomment the next one as you proceed.

### Source Code Files

You are given “skeleton” code files with function definitions that are incomplete and don’t have any implementation. Implement the code and ensure that all the tests in main.cpp pass successfully.

* **GateControl.hpp: This file needs to be completed**
* main.cpp: The main function tests the output of your functions. You may wish to add additional tests. During grading your main.cpp file will be replaced with the original one you were provided with.
* README.md: You must edit this file to include your name and CSUF email. Also include your project partner’s name and CSUF email, if you are working in a group. This information will be used so that we can enter your grades into Titanium.

### Obtaining and submitting code

The skeleton code is available in Project 3 folder in Titanium. When you have completed the project, upload the completed file in the Project 3 folder.

#### Submitting the project

**Only submit the following two files on Titanium in the link provided for Project 3:**

**1) GateControl.hpp**

**2) README.md**

### Testing (either of the two below)

1. **On Tuffix:**

Use the following command to compile your program:  
**clang++ -g -std=c++17 main.cpp -o test**  
  
To attempt to run the compiled test program, use the following command:  
**./test**

##### You can use Visual Studio:

Free download: Community 2019

<https://visualstudio.microsoft.com/downloads/>

### You can write, debug, run your project in Visual Studio.

### Group work

You can work on this project individually or in a group. The maximum group size allowed is 2. If you decide to work in a group, each member of the group still has to submit the code, even though the code will be the same for the two members of the group. Additionally indicate in the readme.md file your name and email and project partner member’s name and email. Both members of the group will get the same grade.

### Grading rubric

Your grade depends on how correct the code is. While passing tests is a good indicator, I still look at the code and mark on the basis of how correct it is.

**You absolutely HAVE to use std::map as the main data structure, along with all other data structures already provided in class definition.**

**No points for using anything else.**

### Deadline

The project deadline is **Sunday, ~~Nov 29, 2020, 11:59 PM~~ Dec 6, 11:59 PM**

**Your code must compile/build for it to be tested and graded. If you only complete part of the project, make sure that it compiles before submitting.**

### Blurb for your resume

Use your GitHub account as a ready-to-show portfolio of your programming projects to potential employers. You can also add a blurb to your resume after successfully completing this project by describing your project in your own words.