CPSC-131 Data Structures

Project 4: Gate Control

Prof. Allen D. Holliday
aholliday@fullerton.edu

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 class 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.

GateControl has 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 arguments are a card number, a cardholder name, and a time range (a start time / end time pair). The format of these times is the same as described below in the Data and Time section. It returns a boolean success/failure status; failure (false) means the item was already present and couldn't be added.
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.
4. ChangeAuthorization, whose arguments are a card number, a cardholder name, and a time range. It changes the name and time range of an existing card. It returns a boolean success/failure status; failure (false) means the card was not found and couldn't be changed.
5. GetAllAuthorizations, which has one output argument: the address of a vector to receive authorization records for the specified card. It doesn't return anything; the vector will be cleared if there are no authorization records. The format of an authorization record is defined in GateControl.hpp.

6. GetOneAuthorization, whose input argument is a card number. It has one output argument: the address of an authorization record. It returns a boolean success/failure status; failure (false) means the card was not found. The format of an authorization record is defined in GateControl.hpp.
7. GetAllTransactions, which has one output 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 input argument is a card number. It has one output 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. It returns a boolean success/failure status; failure (false) means the card was not found and no transactions could be found.

Date and Time

Date and time is 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.

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.

Source Code Files

You are given "skeleton" code files with declarations that may be incomplete and don't have any implementation. Implement the code and ensure that all the tests in Main.cpp pass successfully.

- GateControl.cpp: This is to be completed. Your code should go into this file, not the .h file.
- GateControl.hpp: This is already completed. It contains declarations of the Authorization and Transaction structures, including the associated map and vector. You may add data and functions if desired to help your implementation but you're not required to do so.
- Main.cpp: This is already completed. It's provided for you to use to test your software while you're writing it. It has several functions whose names begin with Test…; you can comment out the calls to tests of functions that you haven't implemented yet. You may change this file if you wish to add helpful functions for your own testing.

  When we test your project, we'll discard your version of Main.cpp and replace it with a different version than the one you were given. Our version will use different test data to prevent you from writing code that handles only specific test cases.

- README.md: You must edit this file to include your name and CSUF email. This information will be used so that we can enter your grades into Titanium.

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. You can edit your copy of Main.cpp to enable only the tests you need to run. Don't wait until the very end to test your code.

Here's a suggested order:

1. AddAuthorization and GetOneAuthorization
2. GetAllAuthorizations
3. DeleteAuthorization
4. ChangeAuthorization
5. AccessAllowed
6. GetAllTransactions and GetCardTransactions

Important Guidelines

Here are some guidelines for submitting projects that will help your instructors deal with the large volume of projects to grade. Failing to follow these guidelines will affect project grades.

README file

Put your name into the README.md file so you will get a grade for your project. When this file doesn't contain a student name, there's no way to know who did the work.

Ada Lovelace died in 1852 and Charles Babbage died in 1871, so they can't possibly have contributed to any project.

Obtaining and Submitting Code

We will be using GitHub Classroom to distribute the skeleton code and collect your submissions. This requires you to have an account on github.com. If you are new to GitHub, do the following to get started:

1. Create an account at github.com. You may want to use this account to show a portfolio of your work to prospective employers in the future, so choose something professional.
2. Read Understanding the GitHub Flow and Hello World at GitHub Guides.
3. Read the instructions below for instructions on how to test.

Once you understand the basic operation of git, click the assignment link to fork your own copy of the skeleton code to your PC.

Do not fork your repository to your personal github account (instructors have admin access to private repositories under https://github.com/CSUF-CPSC-131-Spring2019/). Your code should have a URL like https://github.com/CSUF-CPSC-131- Spring2019/project1-brians, NOT https://github.com/brian/project1-brians.

https://classroom.github.com/a/Ci2GcHVy

Then edit your code locally as you develop it. As you make progress, commit and push your changes to your repository regularly. This ensures that your work is backed up, and that you will receive credit for making a submission. Don't wait until the deadline to learn how to push code!

Testing

Unless otherwise directed, use the following command to compile your program:
clang++ -g -std=c++14 main.cpp -o test

To attempt to run the compiled test program, use the following command:
./test

Grading rubric

Your grade will have two parts, *Form* and *Function*.

- *Function* refers to whether your code works properly as tested by the main function (80%).
- *Form* refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is April 15th at 11:59pm.
You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.

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.