CONFIDENTIAL

Assignment 5 – November 22, 2022 @ 11:59pm.
This assignment must be done individually.

1 Problem Statement

For this programming assignment, the objective is to deepen your understanding of classes and file I/O. [redacted] You will be starting this programming project with some sample code and data. **You will need to submit your code, a typescript, a report, and the results of your investigation.** For a real security incident response, it isn't enough to simply declare your findings. It's required to be able to explain the steps you followed to get to your answer.

2 Description

After many wild restaurant spending sprees, Donna has fallen on hard times. It's not her fault! Being a food critic, barista, worm and crop farmer just doesn't pay like it used to. As a result, Donna owes Olive Garden an amount close to the annual GDP of a small country. To make ends meet and pay off Olive Garden, she's fallen in with a notorious group of hackers known for hacking financial institutions. [redacted] A recent data heist job has gone south and garnered enough attention from authorities that Donna has been brought in for questioning, but Donna is adamant of her innocence. You play the role of a NSA Cybersecurity Specialist who has been provided data and knowledge of the security incident. It's up to you to analyze the data and present your findings in court. [redacted] Can you crack the case and prove Donna's innocence?

#FreeDonna

The affected banking institution has provided the account and transaction data that was accessed by the hackers. You have been provided an "accounts.txt" and "encryptedTransactions.txt" files. "accounts.txt" contains a list of accounts. Each row contains a first name, last name, PIN, and account number. "encryptedTransactions.txt" contains a list of transactions. Each row contains a from account number, to account number, and amount (but encrypted, of course). The provided data is too large to comb by hand, so you'll have to employ your programming skills to interact with the data.

You have also been provided the header files for both Account and Transaction to help get you started – you **shouldn't** need any additional methods than what has been provided, but you can add some if you feel like you need.

The bank's cybersecurity team has shared that the exploit used is only possible for accounts that have not performed outgoing transactions. "Maybe if we can find a transaction leaving Donna's account, we can clear her name", you think to yourself.

Task 1: Create the Account class

This is a container class. It should only hold information relating to the Account class and the Account class attributes.

Using the provided Account.h, create an Account.cpp file and a main.cpp file. Account.h is a header file; It defines all the method and attribute definitions for the Account class. Account.cpp is a source file; It defines what logic is performed when the defined methods and attributes in Account.h are accessed. To handle the data in the accounts.txt file, the account class must hold: "firstName", "lastName", "PIN", and "accountNumber". Use public attributes for "firstName" and "lastName" and use private attributes for "PIN" and "accountNumber". This way, only methods internal to the Account class can have direct access to the "PIN" and "accountNumber" attributes. Additionally, your Account class should have the normal constructors, getter and setter methods for each attribute, and a print method.

Task 2: Create your Account Read Method in Main

You will handle all file reading and writing in your main method. Create a void returning method: "accountRead". This method should take in two arguments, passing a read-only string for filename and a vector object by reference:

```
const string filename, vector<Account> & accounts
```

The accountRead method should read all rows of the provided file into the passed by reference accounts vector. Test your Account class works by adding some test code to your main.cpp file. Because they are public attributes, you can show that the "firstName" and "lastName" attributes can be accessed directly from an instance of Account:

```
// Printing the first and last name of the 0th element in accounts vector:
cout << accounts[0].firstName << endl;
cout << accounts[0].lastName << endl;
```

Task 3: Create the Transaction class

Using the Transaction.h file, continue working on the Transaction.cpp file. To handle the data in the transactions.txt file, the Transaction class must hold: "fromAccountNumber", "toAccountNumber", and "amount". Use private attributes for all "fromAccountNumber", "toAccountNumber", and "amount". Additionally, your Transaction class should have the normal constructors, getter and setter methods for each attribute, and a print method.

Task 4: Create the Transaction Read and Write Methods in Main

Create two more void returning methods: "transactionRead" and "transactionWrite". The read method should take in two arguments, passing string for filename and a vector object by reference:

```
const string filename, vector<Transaction> & transactions
```

The write method should also use two arguments, passing a read-only string for filename and a vector object:

```
const string filename, vector<Transaction> transactions
```

The read method should real all rows of the provided file into the passed by reference transactions vector. The write method should write all elements of the passed transactions vector into the specified file.
Lastly, test your Transaction class works by adding some test code to your main.cpp file.

Task 5: Process the accounts.txt file

Adding to your existing main.cpp, create a vector of Account objects using the following code:

```
vector<Account> accounts;
```

Next open the accounts.txt file using your accountRead(...) function you created:

```
accountRead("accounts.txt", accounts);
```

And process it line by line, adding each account row to your vector of accounts, until you've processed the entire file. **HINT: Make sure the last element of your accounts vector is the last line from the accounts.txt file.** Lastly, print the contents of the accounts vector to the console.

Task 6: Process the encryptedTransactions.txt file

Adding to your existing main.cpp, create a vector of Transaction objects named "transactions" following the provided code in Task 5, but using <Transaction> in place of <Account> like so:

```
vector<Transaction> transactions;
```

Similarly open the encryptedTransactions.txt file using the readTransactions(...) function you created and process the file into the transactions vector. Lastly, print the contents of the transactions vector to the console (you'll see the encrypted transaction data).

Task 7: Prove that Donna is innocent

Referring to the previous information provided to you in the description section, only accounts whom have not made any outgoing transactions can perform the exploit that was used to pull off the data heist. First, iterate over your accounts vector looking for an element where the "firstName" attribute is equal to Donna and store this account object for later use. With Donna's account information, we can attempt to decrypt the encryptedTransactions.txt file. Iterate over each element of the transactions vector and attempt to decrypt each element. The decrypt method will return true when the decrypt operation was a success and the attributes for that element will be decrypted, otherwise it will return false. If the decrypt method returns true for any of the transactions, we can prove that Donna has an outgoing transaction, could not have performed the attack, and we can conclude that she is innocent.

Task 8: Write the decryptedTransactions.txt file

Using the transactionWrite(...) function you wrote previously, you will create a file called decryptedTransactions.txt. Use the account and encrypted transaction data, and decrypt the encryptedTransactions.txt file into its plaintext form. You will also submit this file with your assignment submission. Also, using the encryptedTransactions.txt file and working knowledge of how this exploit is performed, can you prove who these heinous hackers are? Write about this in your report!

3 Testing

Your program must run by using g++ on Turing. Test your program with a variety of inputs to check that it operates correctly for all of the requirements listed above. You are welcome to develop your code on

OnlineGDB if you would like, but the textfiles provided are too big to be stored on OnlineGDB and you will have difficulty once you get to task 4 and/or 6 (depending on how you test task 4).

4 Typescript, Documentation, and Submission

You must also submit a typescript. See previous assignments for more details about how typescripts are created and turned in.

Documentation

When you have completed your assignment, write a short report using the "Programming Project Report Template" describing what the objectives were, what you did, and the status of the program. Since you have a Typescript, the testing section does not have to be as detailed. Talk about what you tested and include any sample outputs that had surprising results. If there are any cases where your code doesn't work or has known problems be sure to talk about it. Save this project report in a separate document to be submitted electronically.

If you have figured out *who* the hackers are (refer back to task 8 above), be sure to mention it in your report! You must also discuss *how* you determined the nefarious hackers' identities!

Submission

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above go to Blackboard to upload your report (a single .docx or .pdf file), your typescript (a single .txt file), the decryptedTransactions.txt file, and your C++ program (all .cpp files and .h files needed). Do NOT upload an executable version of your program.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late, (Sunday, November 27, 2022)
- 20% off if less than 2 days late, (Monday, November 28, 2022)
- 30% off if less than 3 days late, (Tuesday, November 29, 2022)
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

5 Academic Honesty Statement

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations