OOP CRYPTO-WALLET

PROJECT OVERVIEW:

In this project you will build a console-based system to manage different types of cryptocurrencies: A cryptocurrency (or crypto currency) is a digital asset designed to work as a medium of exchange that uses strong cryptography to secure financial transactions, control the creation of additional units, and verify the transfer of assets.

OBJECTIVES:

- To maintain accounts of multiple users and different types of currencies.
- To provide a secure medium for transaction of coins
- Apply Object Oriented Programming principles to derive a solution to this problem
- Use of Standard Template Library Containers
- File Handling

IMPLEMENTATION:

Provided is the skeleton code of the Project.

Currency.h contains a class named currency. This class is supposed to be a representation of a cryptocurrency. It has the member amount denoting the quantity of coins it holds.
This class is further inherited by two classes **Ether & Bitcoin**
You are supposed to implement the function **DeductTransaction** to deduct transaction fee from the account.
The transaction fee for ether is 5 ethers.
The transaction fee for Bitcoin is 2 bitcoins

You are also supposed to implement +=,-= operators for this class.
Note: Be careful to make some functions virtual wherever required.

Account.h contains a class Account. This class is a representation of a cryptocurrency bank account. In the normal world a user is known by some unique id and password. The password is secret to the user while the username is available to everyone.
In cryptography an Account has a public key and a private key. Public key is like a username, anyone can use this to transfer coins to an account while a private key is like a password. A person can send coins from an account only if he has the password.
The account also has a pointer of currency to store currency;
Storing Public Key & Private Key: Storing passwords as they are is not safe in today's world. In case of data theft all the username & passwords would be lost. So we never store passwords as they are, instead a hash of passwords is stored.

Hashing: An algorithm to convert string into some unique encoded string to check for data verification or validation is Hashing.

Hash can be used to convert a string into another unique string and store it in the system, since hash is unique, when a user enters a private key or password,hash of that string is calculated and compared with stored hash.
Hash function is exclusive to organization. In this project we'll be implementing two of our own hash Functions(discussed later).
Account(string,string) takes the public key and private key as parameters. The public key is stored as it is. A hash of the private key is calculated and then stored.
getHash() function takes a string and returns the hash of that string.
VerifyAccount() takes a string as parameter,takes its hash and compare it with stored private_key_hash
SendCoins() first verify the account then reduce coins from amount.
AddCoins() add the passed amount of coins in the account.

The Account class is further inherited by two classes **i.e EtherAccount and BitCoinAccount**
EtherAccount.h:
In this class when the constructor is called , the currency pointer in Account is allocated a memory of type **Ether.** An ether account has a default amount of 120 ethers when created.
getHash() function follows a newly developed encoding algorithm **ASCIHash.** In this algorithm we append the ascii of every character in the string into the resulted string
For example: "abc" hashes to "979899" because ascii of a is 97,b is 98 and c is 99.

BitCoinAccount.h:
In this class when the constructor is called , the currency pointer in Account is allocated a memory of type **BitCoin.** A bitcoin account has a default amount of 10 bitcoins when created.

getHash() function follows a newly developed encoding algorithm **RepeatTilSize.** In this algorithm for every character we append the character and the next ascii characters till length of string into the resulted string
For example: “abc” hashes to “abcdbcdecdef” because size of “abc” is 3 and next 3 characters of a are bcd, b are cde, c are def.
If the ascii has no next members e.g 125,126,127 the hashing loops back and starts adding from zero.

Transaction.h has a class named Transaction. It contains records of transfer of coins from one account to another.
TransactionHashed inherits class Transaction. It has only one difference that it stores hash of public keys of sender and receiver. The type of hashing depends upon the account type i.e **ASCIIHash** for **Ether** and **RepeatTIllSize** for bitcoin.

AccountManger.h:
This file contains a class AccountManager which has an array of Accounts and its size and a Vector of Transaction to store all the transactions till now.
If you don’t know about vectors. Here is the link
std::vector

AddAccount() function adds the passed account into the array of Accounts if not already.
GenerateRandomKey() this function returns a random 3 letter string every time. The string must be alpha-numeric e.g only numbers and alphabets.
CreateAccount() function generates random public and private keys and creates an account object (of the passed type, BITCOIN OR ETHER) using the generated keys, adds the account to arr and returns the private key of that account.
MakeTransaction() Send Coins from sender_public_key if private key is correct.Coins must be added to receiver account. If the receiver is not found then coins are lost. Make an object of Transaction and insert it in TransactionLog.
Note: Sending Coins from BitCoinAccount to EtherAccount and vice versa is possible with conversion rate :1 bitcoin = 12 ethers.
WriteToFile() function writes the state “this” to a file whose name is passed as parameter. File must be created if not already.
LoadFromFile() function reads the state from a file whose name is passed as parameter. Should give an error message if file is not present.

HashedAccountManager
This class inherits AccountManager. The only difference is that it contains a vector of HashedTransaction instead of Transactions.

The Final Program:
In the main file of the program you will create an Account object miner with public key “Miner” and private key “123”. Then you’ll create an object of AccountManager and add miner account to it.
You must provide the following options
1.Create Account
2.Make a Transaction
3.See Transaction Log
4.See All Accounts

You can use the already implemented toString() functions to display.
Create Account: This option will ask user to select from type of accounts e.g
BITCOIN or ETHER and create an account of that type using the CreateAccount Function of AccountManager. It would display the public key and private key for the user to note down.
Make a Transaction: This option will ask the user for the public key and private of Sender ,then Public key of receiver and then amount to be transferred and call the MakeTransaction Function of AccountManager.
Note: For every successful transaction you used log 2 transaction in transaction log. One from sender to receiver and second from sender to a user whose public key is “Miner” amount equal to the transaction fee of the account(This is where the transaction fee is transferred).

GUIDELINES:
- You can modify the access label of members from private to protected wherever required.
- You can make functions virtual to implement runtime polymorphism wherever required.
- Do not modify the toString functions in all classes, since these would be used to evaluate your project.
- You need to create classes HashedTransaction and HashedAccountManager yourself.
- Start as early as possible.

SUBMISSION INSTRUCTIONS:
You add code in the provided template wherever required. Create a zip file named you rollNo only (19I-XXXX) and submit.