CS 130A
Assignment I - Transaction-chain

Assigned: October 4, 2018
Due Demo Day: October 17, 2018

PLEASE NOTE:

- Solutions have to be your own.
- No collaboration or cooperation among students is permitted.
- 5% of the points will be deducted for each day the assignment was late, with a maximum of 4 days
- Requests for a regrade must be submitted within seven days from the day when we return the assignment.

1 Introduction

Blockchain is a list of blocks which are linked using cryptography. Each block consists of multiple transactions and each transaction has money transfer from one (sender) to another (receiver). In other words, when a (*sender*) sends *money* to a (*receiver*), that interaction is stored as a *transaction*. In blockchain, each block may contain one or more transactions along with the hash value of the previous block.

In this assignment, you will implement a simplified version of blockchain. Every block will represent a single transaction which consists of an amount, receiver and sender details along with the cryptographic hash of the previous block. Transactions will create a transaction-chain which will be represented using a ***linked-list***. The content of each node in the linked-list contains an *integer* amount field and two *strings* for sender and receiver, . Each node in the linked-list also contains a *hash pointer*, which is a *pointer* to the previous node for traversal and the *hash* of the contents of the previous node in the blockchain.

NOTE

- You can either use a *class* or a *struct* to represent a block in the linked-list.

- The very first transaction will not contain a hash field (It could be NULL), because there is no transaction before the first transaction.

Figure 1 shows the structure of the linked-list and contents of each node:

```
class Transaction
{
    public:
        ...
    private:
        Transaction * next; //points previous transaction
        int amount; //amount of money that has been transferred
        string sender; //sender of money
        string receiver; //receiver of money
        string nonce; /*random string that will be used
                        during hashing for the next transaction*/
        string hash; //hash of content of previous transaction
    ...
};
```

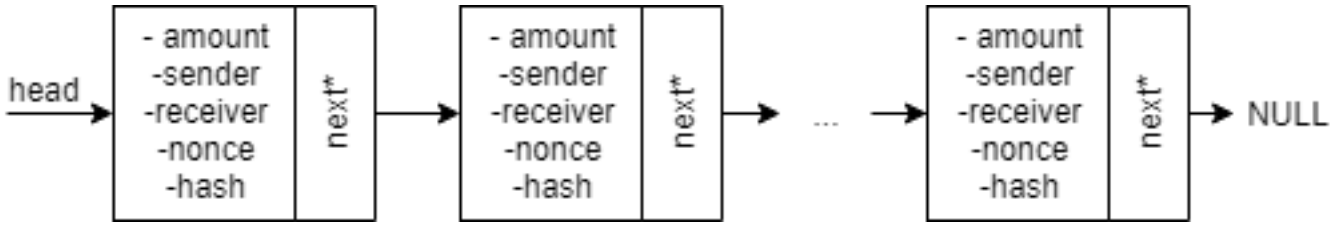

Figure 1: Transaction-chain

In order to generate the hash of the previous transaction, you will be using cryptograhic hash function (SHA256). You are **not** expected to write your own hash function. You can use existing C++ libraries for hashing. OpenSSL is one such library that provides SHA256 hash function. However, you can use other libraries if you want.

Most hash functions take *string* as input. So, you need to hash the previous transaction after changing it to string format as shown in Figure 2. The output of SHA256 is an unsigned char* represented in hexadecimal format. You can convert the output into a string consisting of digits 0 through 9 and letters a through f.

$$T_{n+1}.hash = SHA256(T_n.amount + T_n.sender + T_n.receiver + T_n.nonce)$$

Figure 2: T stands for Transaction

IMPORTANT NOTE: When calculating the hash value, you need to make sure that the last character of the hash is a *digit* (0-4). In order to do that you will create the nonce randomly. *The length of the nonce is up to you.* If the calculated hash value does not end with a digit between 0 and 4 as its last character, you will have to try another nonce again.

In other words, you need to create the nonce randomly until the right most character of the resulting hash value is a digit between (0-4). This is a simplification of the concept of *Proof of Work (PoW)* used in Bitcoin.

The ***code below*** shows you how to create a random character between a-z in C++. You may use either this code or develop your own. Note that the *rand()* function generates a random integer. We are calculating mod26 and adding 97 (ascii code of 'a') to get a character between a and z. You may refer to the ascii codes of the characters here: http://www.asciitable.com/

```
#include <cstdlib> //for rand()
#include <iostream> //for time()

int main(){
    //every time you run the code, rand() returns a different value.
    srand (time(NULL));
    //prints the generated character between a-z
    std::cout << char(rand() % 26 + 97);
    return 0;
}
```

2 Implementation details

As a part of this homework, you will implement 3 functions as explained below:

- *Add Transaction:* This function will take 3 inputs: amount, name of the sender and the receiver. *Amount* should be an integer type so you need to check if it is integer or not. If it is not an integer, you will reject it and ask the input again. After all inputs are correctly given, you will create a transaction from the inputs, add the transaction to the transaction-chain and prompt an informative message to the console. **Don't forget!** You will also need to find the hash value of the previous transaction and add to the newly added transaction with the corresponding *nonce*, which guarantees that the hash ends with a digit. You should also print the hash value you found.

- *Find Transaction:* This function will take only 1 input which is the name of a sender. You will need to traverse the whole linked-list and print all the transactions which have the same sender's name.

- *Verify and print the chain:* This function will not take any inputs. This function will traverse the whole linked-list and for each block b_i (i.e., node) in the list, the function will verify if the hash stored in b_i is the correct hash of the transaction in b_{i-1}. You can use the formula shown in Figure 2 to verify the hashes in the linked-list. If you find a block b_j with hash h_{j-1} that does not match the hash of the transaction in b_{j-1}, then the function will return false and will print the two blocks with the mismatching hashes. If there is no problem with the verification process, your function will return true and print the chain. (You can use your own way to print the chain, but it should be clear and informative).

2.1 Program Flow

NOTE: We do not want any front end UI for this project. Your project will be run on the terminal and the input/output for the demo will use stdio.

In the beginning of the application, you will prompt a welcome message to the user and ask them which of the three functions the user wants to use. Based on the input, you can prompt the user to input further details such as *amount, sender and receiver* details. You can create your own style, but below is an example:

```
Welcome to the transaction-chain application....
1) Add a transaction to the chain.
2) Find a transaction with name of receiver or sender.
3) Verify and print the chain.
Which operation do you want to make? (1,2,3):
```

If the user enters different operation, you will prompt "wrong operation!" message and ask the question again.

3 Demo

We will have a short demo for each project. It will be on **October 17, 2018** in CSIL. Time details will be announced soon. Please be ready with the working program at the time of your demo.