**Assignment 6 (Team of 1-5 people)**

**Done by team of: Kurbanaliyev Omar**

**Temirzhanov Meiram-Sopy**

**Appazov Yerzhan**

**Khalamagamedova Aminat**

**Temirgaliyev Damir**

**Part 1**

Create simple blockchain by creating 3 Classes (TransactionData, Block, Blockchain)

1. Explain in 1-3 sentences what is Blockchain?

Blockchain is a system of recording information in a way that makes it almost impossible to change or cheat the system. Looking for its structure, we can imagine them as just a linked list of blocks that are connected together. It consists of blocks that stores specific data that can be used for number of purposes. These days they are used as cryptocurrency because of their hardness to change the data inside and security measures.

What is Blocks?

Block is part of blockchain that stores hash of their own data, hash of previous block and some data. The type of data stored inside depends on for which purpose the blockchain is being used.

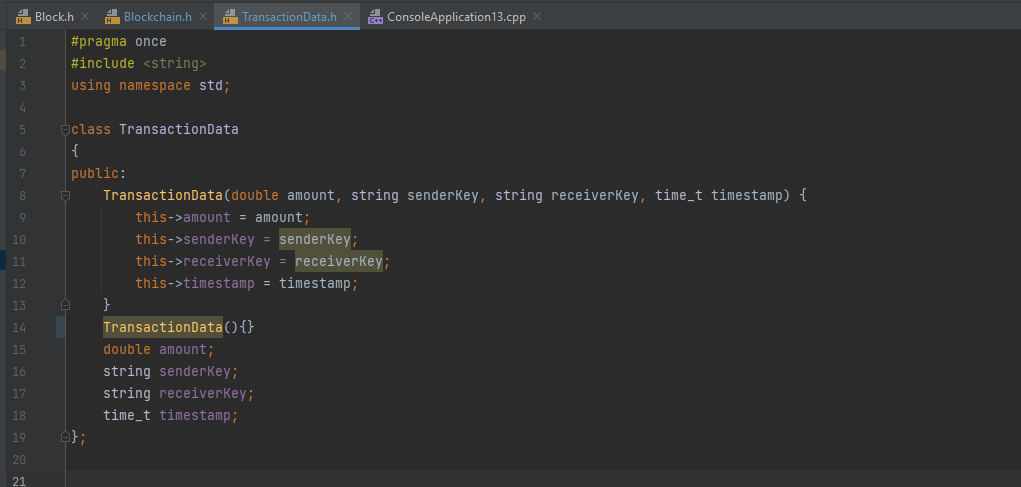
What is transaction on blockchain?

Transaction on blockchain is a way of sending cryptocurrency using blockchain system. For example: in Bitcoin block, the block data stores info about ‘Sender’, ‘Receiver’, ‘Amount of Cash’ and unique ‘Hash’ that identifies exact block and ‘Hash of Previous Block’.

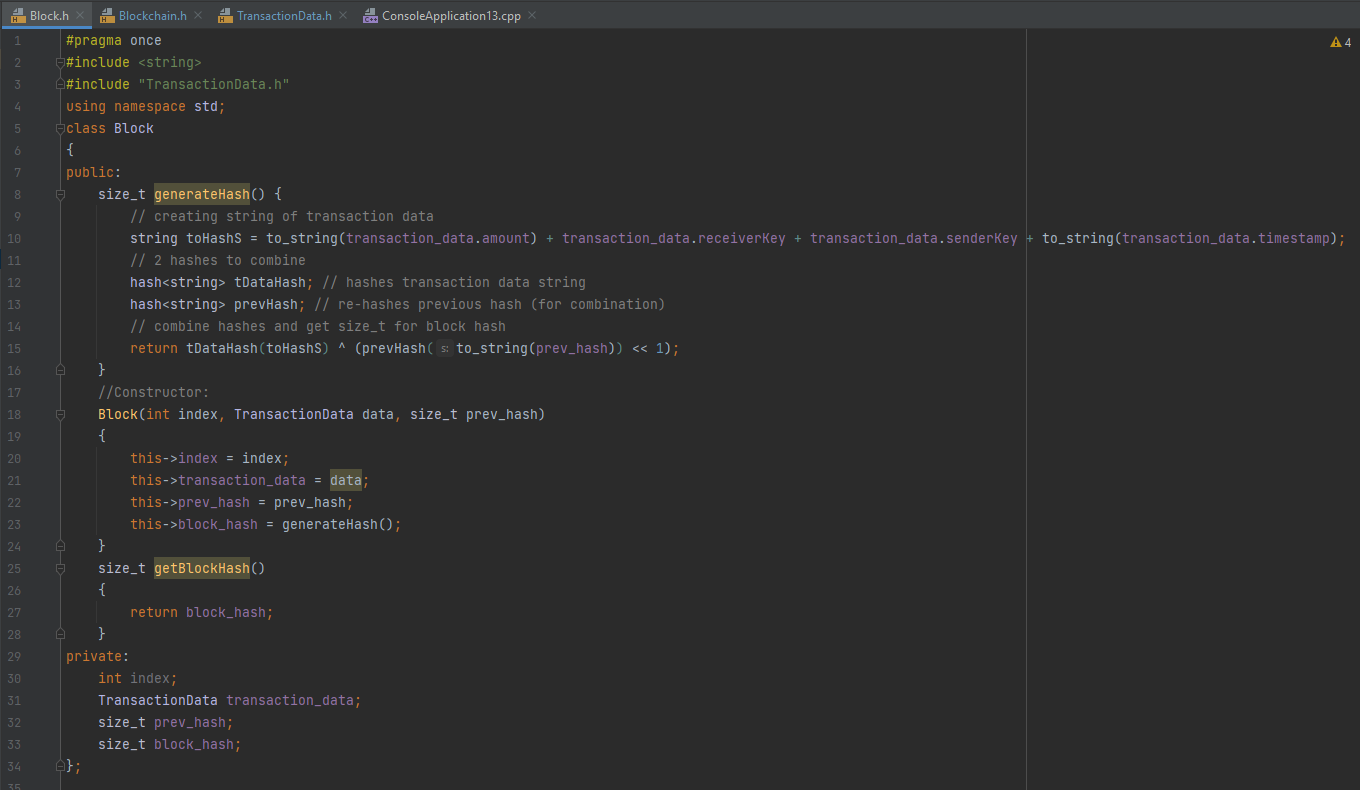
How hashing is happening on blockchain?

Hashing is similar to encryption, but the difference is that in hashing there is no way to get input data back from hash code like decryption. Hash input is any data of any length, however the output is hashed and fixed to specific size. Every hashed output is unique for every data and changing even one letter in input data changes the hash code enormously. In theory, it is possible to get input data from hash code by brute force, but it takes number of decades to implement such idea, which is meaningless for hackers. This feature makes hashing in blockchain so useful in payments, making them almost impossible to cheat.

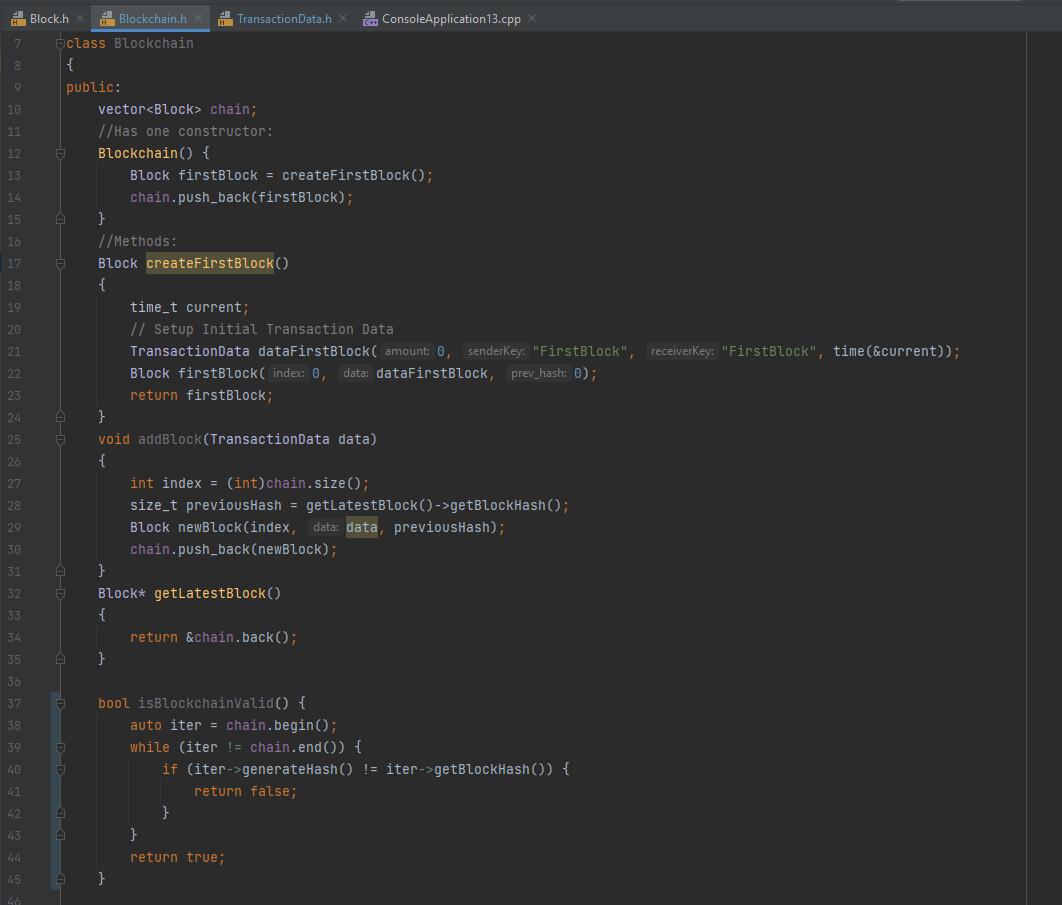
1. Create struct TransactionData double amount, string senderKey, string receiverKey, time\_t timestamp with standard constructor:



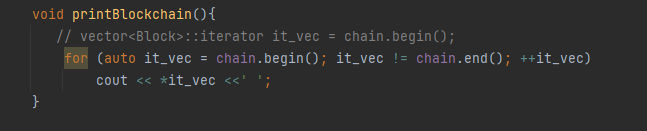
1. Create classes Block (int index, TransactionData transaction\_data, size block\_hash, size\_t prev\_hash), that has methods: standard getters, and methods:



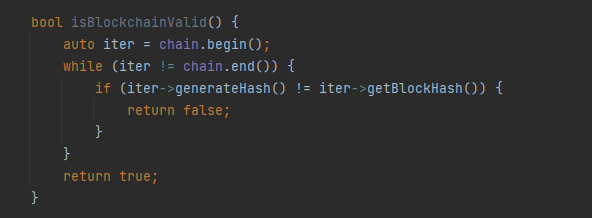
1. Create class Blockchain (vector<Block> chain)



1. Create method printBlockchain() that will print all details about each block in the blockchain starting from the first one (use iterators: vector<Block>::iterator **it**;)



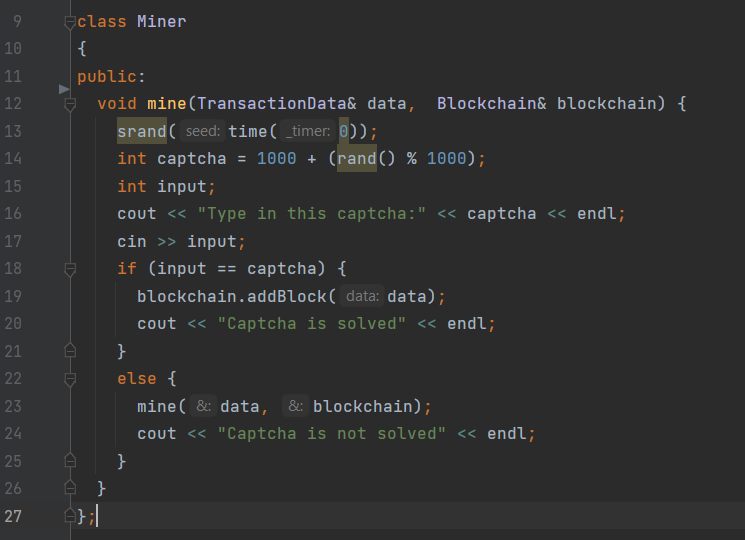
1. Create method isBlockchainValid() that returns bool (also use iterator). You can tell that blockchain is valid if hashes of each block in blockchain is valid. You can check if hashes of each block is valid if **generateHash()** method of the block is equal to **getBlockHash()** method of the block.



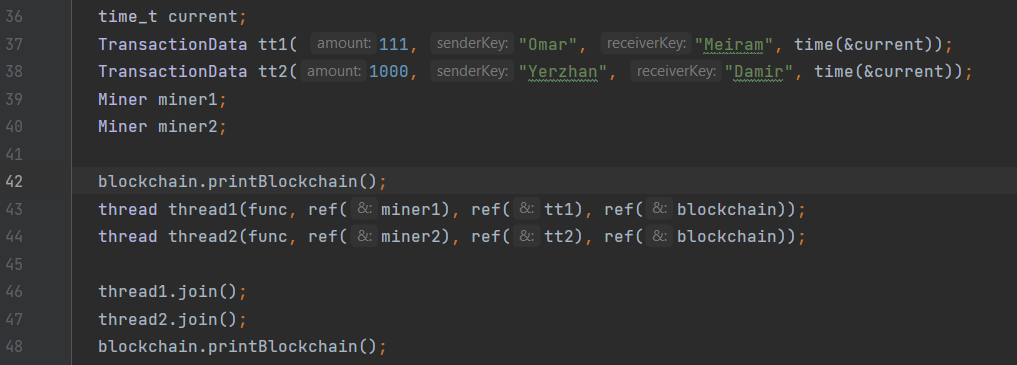
Part 2.

Test your simple blockchain

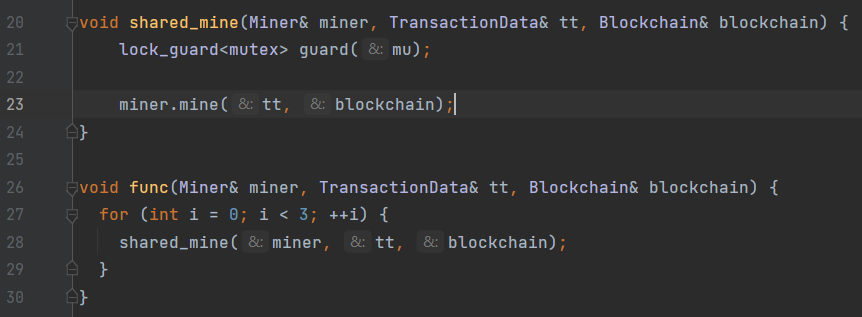
1. Create class Miner with single method mine(Blockchain **blockchain**, TransactionData **data**) that will require to solve some captcha (your design), if captcha solved successfully the **data** can be added to **blockchain**.

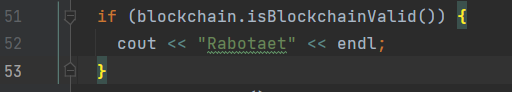


1. In main() create an instance of Blockchain. Start new\_thread1 and new\_thread2, create instances of Miner in each new\_threads and call method **mine** from Miner class in new\_thread1 and new\_thread2.



1. **mine** 3 transactions in new\_thread1 and 3 transactions in new\_thread2, then printBlockchain() on main thread. Also check isBlockchainValid() in main thread.

****

****

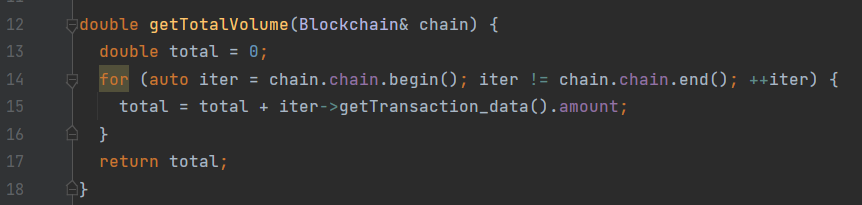
1. Explain where you could use mutex in the Part 2? Where you might use unique\_lock? And why?

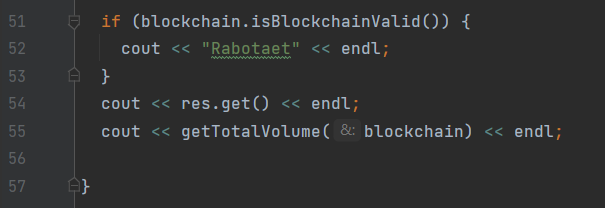
We used mutex in shared mine method and we used unique\_lock in the same place. We used mutex to make sure that one thread cannot access to the blockchain while the other is using it. We used unique\_lock to make sure that mutex will be unlocked even if some exception is thrown.

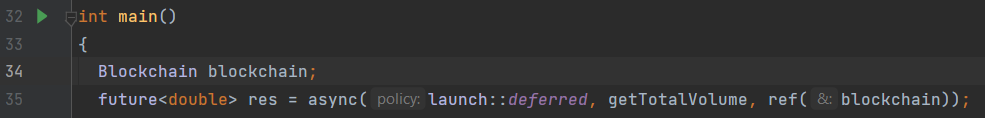
1. Apply mutex and some type of lock where you feel necessary.

Mutex and lock was already applied.

1. In main Create function **double getTotalVolume(Blockchain &chain)** that accepts Blockchain by reference and returns total **amount** transacted on the blockchain. Create async() in main thread and call **getTotalVolume** function, store results in **future<double> res**, print the results.







=====

Bonus: work with git and upload files to Github, have 3 commits min. Provide link (+3%)

GitHub: <https://github.com/ok1203/ConsoleApplication13>