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

**Part 1**

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

1. Explain in 1-3 sentences what is Blockchain? What is Blocks? What is transaction on blockchain? How hashing is happening on blockchain?
2. Create struct TransactionData double amount, string senderKey, string receiverKey, time\_t timestamp with standard constructor:

TransactionData(double amount, string senderKey, string receiverKey, time\_t timestamp){ this.amount=amount, this.senderKey= senderKey, this.receiverKey= receiverKey, this.timestamp = timestamp }

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

generateHash(){

// creating string of transaction data  
string toHashS = to\_string(transaction\_data.amount) + transaction\_data.receiverKey + transaction\_data.senderKey + to\_string(transaction\_data.timestamp);  
// 2 hashes to combine  
hash<string> tDataHash; // hashes transaction data string  
hash<string> prevHash; // re-hashes previous hash (for combination)  
// combine hashes and get size\_t for block hash  
return tDataHash(toHashS) ^ (prevHash(to\_string(prev\_hash)) << 1);

}

//Constructor:

Block(int index, TransactionData data, size\_t prev\_hash)  
{  
 this.index = index;  
 this.data = data;  
 this.prev\_hash = prev\_hash;  
 this.block\_hash = generateHash();  
}

getBlockHash()  
{  
 return block\_hash;  
}

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

//Has one constructor:

Blockchain(){

Block firstBlock = createFirstBlock();

chain.push\_back(firstBlock);

}

//Methods:

Block createFirstBlock()

{

time\_t current;

// Setup Initial Transaction Data

TransactionData dataFirstBlock(0, "FirstBlock", "FirstBlock", time(&current));

Block firstBlock (0, dataFirstBlock, 0);

return firstBlock;

}

addBlock(TransactionData data)

{

int index = (int)chain.size();

size\_t previousHash = getLatestBlock()->getBlockHash();

Block newBlock(index, data, previousHash);

chain.push\_back(newBlock);

}

Block \*Blockchain::getLatestBlock()

{

return &chain.back();

}

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**;)
2. 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**.
2. 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.
3. **mine** 3 transactions in new\_thread1 and 3 transactions in new\_thread2, then printBlockchain() on main thread. Also check isBlockchainValid() in main thread.
4. Explain where you could use mutex in the Part 2? Where you might use unique\_lock? And why?
5. Apply mutex and some type of lock where you feel necessary.
6. 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.

=====

1. Bonus: work with git and upload files to Github, have 3 commits min. Provide link (+3%)
2. Penalty: similarity >70% (-100%)
3. Penalty: late submission (-5% for each hour)
4. Penalty: missing defense (-30%)