# Assignment 1

## A Supply Chain Management Tool

Due Date: 23 September 2022

***This is an individual assignment.***

In this assignment you will implement a secure supply chain management tool for an e-commerce website. You will be developing a linked-list implementation of a (highly simplified) blockchain for this assignment. The purpose of supply chain management is to be able to track the supply of a product potentially from the raw material to the final customer. In this assignment, we simplify this down to tracking only four stages of an online order, from order placement to order delivery.

### Blockchain Background

You do not need to understand exactly how blockchain technology works; it is enough to know that a blockchain is essentially, as the name suggests, a chain of connected blocks (just like a linked list is a chain of connected nodes) that contain some data. In this assignment we assume that when a new transaction occurs, it is stored on the blockchain as a new block. A blockchain should be tamper-proof, i.e. no one can change or delete any data that has become part of the blockchain. This is accomplished through hashing. Each block contains *hashes* of all previous blocks. A hash is a one-way mathematical function that is applied to some input data to get a fixed-size, usually small output. A block contains the hash of the previous blocks so that if any block is removed or changed, the hash value of all subsequent blocks will not be correct, and it can be detected that something was changed.

### Assignment Details

First you will create an initial .csv file with a list of about twenty transactions that have occurred in the past. Your first task will be to read in these transactions into a Linked List data structure (your blockchain). The file will be formatted as follows:

```
Transaction Number, Order ID, Timestamp, Status, Item, Purchaser, Seller, Price
```

Here is a sample of one line of the csv file:

```
1, abd3re4s, 1603612778, ORDER_RECEIVED, Blk-Headphone-E45, john.doe, appletech, 3400
```

Note that the timestamp is in Unix time, i.e. seconds since 1 January 1970.

Here are the details of each column:

1. Transaction Number - an integer beginning at 1 noting the transaction number.
2. Order ID - a unique 8-character string identifier corresponding to each order that was placed. An order ID could appear multiple times in the file, each with a different status indicating where it is in the supply chain. This order number should have alphabets as well as digits.

3. Status - the status of the order. Valid values are listed below:
   - ORDER_PLACED - indicates that someone placed an order for this item
   - PAYMENT_PROCESSED - indicates that payment was made for the item and is ready for shipment
   - SHIPPED - an item has begun shipping from the warehouse
   - RECEIVED - the item has been arrived at the shipping address and was signed for by the purchaser
4. Item - a string indicating the item name
5. Purchase - a string containing the purchaser username on the website. Could contain white spaces.
6. Seller - a string containing the seller name (like the store/seller name on a website like Amazon or Daraz.pk). Could contain white spaces.
7. Price – an integer representing the purchase price in PKR

**Note:**
All Transaction numbers are unique and should be auto incremented beginning at 1.
Order ID is used to track an item through the order chain process.

Your program should read in this list of transactions (ignoring the header row of the csv), create a Transaction object for each row, then add it to a linked list representing all transactions that have been read in by the file. As the linked list is a blockchain, each block should contain, in addition to the data and the next pointer, a hash value of the previous block. So every time you insert a new block, calculate the hash of the previous block and add it to the new block.

**Commands**
Once you have read in all transactions, your program will prompt the user to enter a command via the console. Your program should check to make sure that a user has entered in a valid command, i.e. only from the set that follows, and included all the required parameters.

ADD_NEW_ORDER [Item, Purchaser ID, Seller ID, Price]
This command should allow inserting an order into the blockchain. A new unique 8 character ID that is a mix of alphabets and digits should be generated for the order, and the status should be ORDER_PLACED. The timestamp should be auto-generated as the current system time. This command results in an insertion into the linked list, so remember to calculate the hash of the previous block and add it to the newly created block.

MODIFY_ORDER_STATUS [Order ID, Status]
This command should allow adding an update about a previously placed order. A new transaction will be created and added in the blockchain. The purchaser, seller and item will be copied from previous occurrences of the order ID, and the new status as entered by the user will be entered under status. As this also represents an insertion in the linked list, note that you should correctly calculate the hash value of the added block.

CHECK_STATUS [order ID]
(Example: CHECK_STATUS becfd046-7361-43e5-97c5-8290e76a639b )

This command should print out each transaction that has occurred for a single order # provided in the command. This should be printed out in a table format, ordered by the timestamp.

REPORT PURCHASER [name]
(Example: REPORT PURCHASER Erik G. )
This command will print out all transactions that have been created for this particular purchaser.

REPORT SELLER [name]
(Example: REPORT SELLER William and Sonoma )
This command will print out all transactions that have been purchased from this seller.

EXPORT
(Example: EXPORT )
This command should print order ID's and details in four different files indicating which orders are in which current status. The current status of an order is determined by finding the ***most recent*** transaction for an order and looking at the status of that particular transaction.
For example, if an order first appears in ORDER_RECEIVED and then later the same ID appears in another transaction with the status PAYMENT_PROCESSED, that order would be printed in the file for report-payment-processed.csv.

The files that are exported should be as follows:

- report-order-placed.csv - all orders that are in the ORDER_PLACED status
- payment-processed.csv - all orders that are in the PAYMENT_PROCESSED status
- shipped.csv - all orders that are in the SHIPPED status
- received.csv - all orders that are in the RECIEVED status

No order should appear in more than one file, and each order should only be printed once (using the last transaction). In other words, multiple transactions should not appear in a report for a single order, just the latest.

DELETE TRANSACTION [Transaction Number]
This command should delete a particular transaction number from the blockchain. You should not recalculate the hashes for the nodes that come after the deleted node.

RUN SECURITY CHECK
This command simply invokes the **security_check** function of the blockchain (described below) and outputs an alert if the blockchain has been invalidated. It also calls the recovery function which recovers any deleted blocks from the backup and inserts them back into the right position in the chain (i.e. ordered by timestamp).

DONE
This command will terminate the program.

**Demonstrate a Security Breach Attempt and Recovery:**
First write a **security_check** function that you can call periodically. The function should iteratively check that the hash of every block in the blockchain is indeed the hash of the previous block.

**Hashing function:**
The hashing is done as follows. First convert every block (i.e. each node of the linked list) to a string by concatenating all the string data members of the class that is representing the block. Use any library of your choice to calculate the md5 OR sha-1 hash value (md5 or sha-1 function usually takes a string as input). The output of the function will be your hash value for that block. Note that the hash value is a data member of the block, but it stores the hash value of the *previous* block, not of the current block itself. Hence, the hash value (of the previous block) is treated as a string in the current block and should be included in the construction of the concatenated string itself that you will use to calculate the next block's hash value. Whenever you insert a block in the blockchain, you must populate the hash field with the hash value of the previous block. This way, the hashes are chained together securely and if someone deletes or alters a node, the hash value stored in the next block is no longer valid and the change can be detected.

Write code to simulate a security breach attempt, and show how your system auto-recovers from the breach. Suppose some malicious actor gains access to your program and attempts to delete a block from the blockchain using the DELETE TRANSACTION command (for example, a malicious seller can try to delete the payment_received transaction for an order and claim that the purchaser never paid him). The block should get deleted, as it is just a linked list, but the hash values in the following blocks will now be invalid, as the hash of the preceding block is now not the same. You should write code to demonstrate a scenario where the security breach is attempted, but the security check function detects the breach, the system recovers the node from a backup copy (implying you should back up the whole chain ***on disk*** (i.e. write it to a file) every time there is any insertion) and inserts it back into the right place.

Overall, write your driver program to demonstrate all the functions, with a menu-driven interface where the user can continue to input various commands until the program exits.

## Marking Rubric

This assignment will be marked as follows:

| Item | Marks |
|---|---|
| Correct structure of starting .csv file | 5 |
| Auto-incrementing transaction numbers | 5 |
| Correct calculation of timestamp | 5 |
| Correct generation of order ID (random and unique, correct format) | 5 |
| Correct design and running of the menu driven interface | 10 |
| Correct running of ADD_NEW_ORDER | 20 |
| Correct running of MODIFY_ORDER_STATUS | 20 |
| Correct running of CHECK_STATUS | 10 |
| Correct running of REPORT PURCHASER | 10 |

| | |
|---|---|
| Correct running of REPORT SELLER | 10 |
| Correct running of EXPORT | 40 |
| Correct running of DELETE TRANSACTION | 10 |
| Correct running of RUN SECURITY CHECK (including correct hash calculation) | 20 |
| Correctly recovering from security breach (backing up and restoring deleted node from file back up) | 30 |
| Correctly and clearly demonstrating a security breach and recovery | 10 |
| Code style: commenting and meaningful variable names | 10 |
| Code modularity (separate classes, .h and .cpp files for each class, and OOP encapsulation principles) | 30 |
| Correct naming and submission | 10 |
| **Total:** | **260 Marks** |