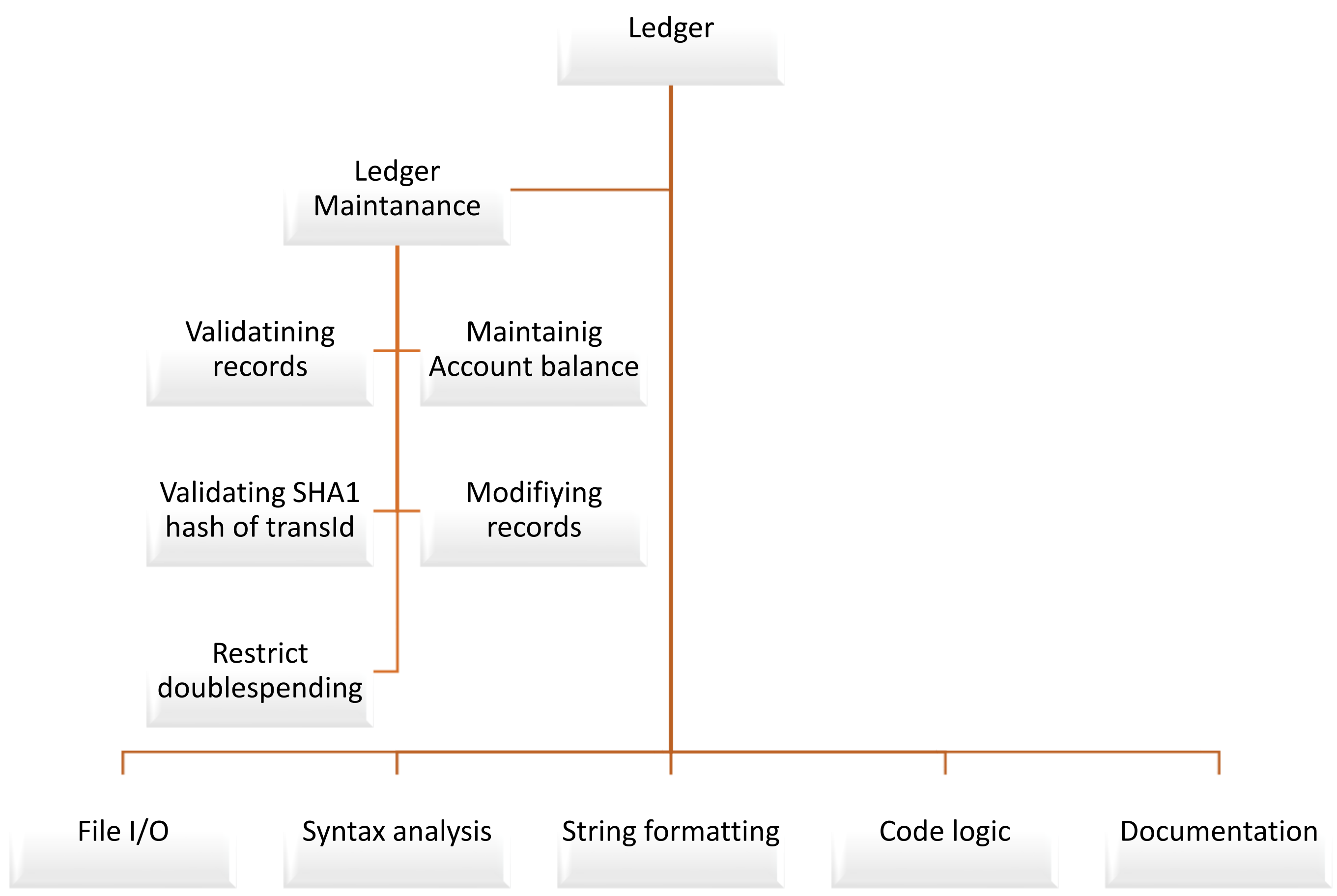
Approaching the Ledger assignment

YUVARAJ SRIPATHI

UFID: 14677313

I followed a top-down modular approach to check the validity of the transaction i.e. several independent modules were created and finally used the solution from these modules to confirm whether the given transaction was legitimate or not. C++ standard 11 programming language is used for assignment



For dealing with the basic file, string formatting and syntax checking operations, the code logic is to create separate functions to deal with each of them and correspondingly communicate them to each other as the bigger picture requires all of them to be working on the same piece of data to get the required solution.

- **File I/O** – file input/output operations are done using "fstream" library provided by C++11 standard. Input is used when a file is read to get the contents of the ledger wherein each of the lines represent an individual record (valid/not valid) which needs to be checked for validity. Output is used for dumping the ledger (validated records) into an output file.
- **Syntax analysis** – Each of the record read from the file or received from the user as part of "[T]ransaction" operation, needs to be checked if the records follows the format rules i.e. placement of delimiters (";") and pairs of tuples conform to the given number or not like checking the number of given "vto" is same as the "M" and if given number of output is same as "N" and check if the parenthesis are balanced and "," is present in right positions. If the syntax is found to be wrong, the record read is rejected (Validation turns out to be false) and not included in the main ledger else if syntax is right, the record is sent to another module to check if enough money exist and whether SHA1 hash (transId) is correct for the record and further decisions are taken in those modules.
- **Code logic** – Each of the valid record read from the file or given by the user is stored in the program in the form of Linked List in the form of pair(transId, Record) where Record is a custom data structure containing information related to that record. The Linked List is chronological (maintained in the order of input) and also combination of hash maps given by C++ standard 11 are used for maintain balance for each individual and to check if a given transaction is feasible or not.
- **Ledger Maintenance**
    - **Validating records** – After syntax analysis, the record needs to be validated based on feasibility of the transaction i.e.
        - If there is enough money to be transferred from debitter (utxo)
        - If the utxo is used as a whole (logic given by Bitcoin ledger)
        - If utxo is used up, indicate them with a flag to restrict others from using them in future
        - Mark the transId and utxo if used to restrict double spending
        - If there is not sufficient balance or if using an already used up utxo, display the corresponding error
    - **Maintaining account balance** – based on outputs for the records, maintain a hash map that contains the balance for each individual against their name and update them on every valid transaction
    - **Validating SHA1 hash** – The transaction id (transId) of every valid record needs to be the first 8 characters of the SHA1 hash of the record without transId. If the hash is found to be wrong from what we calculated using openssl open source library (source and library included in the project folder) then update the record with correct hash and display a message to inform the user about the same. If the transId is changed, then correspondingly modify in all the hash maps that use transId as the key.
    - **Restrict double spending** – For every valid transaction, the utxo being spent needs to be marked against the transId as a set that is used up. This makes sure that the utxo which is already used cannot be used in further transactions. Also, once a transaction Id enter the main ledger, it cannot be used in a new record (this is done by checking the map that contains valid transaction id in the ledger) thereby restricting double spending issue.

The completed assignment gives an executable "ledger", and this was tested for correctness on all the operations that it offers and checked for validity, syntactic analysis and balance queries of made up transaction records that was given in the assignment brief and also discussed with other fellow class mates to confirm the correctness. Also, tested on UF CISE department's storm server and it works fine starting with installation and ledger execution.

To test the correctness of the ledger,

- Checked for syntax-based validation to find out if the records are well formed or malformed.
- Made sure same transaction Id doesn't get into ledger by giving inputs with already used up transId and making sure that corresponding error is displayed.
- Check for SHA1 hash and update. This is checked by using records with wrong transId hash and modifying the transId to the correct hash for the records and then enter the same to the actual ledger and also check if all the corresponding data structures having transId as a member are also updated (for keeping a check on balances and valid records in the ledger). The hash values are calculated using the SHA1 API provided by the opensource openssl library that is include in the same folder as that of the source files and the links to the library are provided to the compilation command in the make file.
- Check if already used up utxo is referenced again. If so, don't enter the record into the ledger and display the necessary information.
- Do all the above mentioned on both "FILE" and "TRANSACTION" operations offered by ledger.
- In the "PRINT" option, check if all the records displayed are the valid ones and are in chronological order and contain the correct transId (correct hash value). Also, check for syntax.
- In "DUMP" option, check if the output to the given file, follows the same format as that for "PRINT" operation.
- Check if all the records and all the data structures maintained are erased and the next set of entries to ledger can start from record #1.
- For "BALANCE" operation, tested the correctness by comparing the displayed result and manually calculating the balances as per the ledger.
- "INTERACTIVE" and "VERBOSE" operations are tested to see if the corresponding messages are displayed accordingly.

As per my knowledge from the testing phase, all the criteria mentioned in the assignment brief have been met

Experience gained from this assignment,

I got an idea as to how the ledger is maintained and how double spending issues can be dealt with and gives an idea about the challenges that can occur while taking it to distributed peer-o-peer architecture and dealing with issues like concurrency, fraudulent activities and stress that can be handled when the number of users is high.