CS 218 DATA STRUCTURES

ASSIGNMENT 1- Linked List

Fall 2023

DUE: Thursday, Sept 21, 2022

NOTE: Late submissions will not be accepted

TO SUBMIT: Documented and well-written structured code in C++ in the classroom. Undocumented code will be assigned a zero.

Assignment Description:

In this programming exercise, you will implement a small banking system where the bank stores a linked list of accounts. Each account is characterized by an account number, account title, account balance, and a history of account transactions. The history of account transactions is also maintained in the form of a linked list. Each transaction includes information such as transaction ID, transaction time, transaction date, transaction type (credit or debit), the amount transferred, and the account number (if any) through which money is transferred. Whenever a transaction is performed on an account, its history is added to that account's transaction history.

The bank offers a special service where two accounts can be merged. In this situation, a new account is created, and the previous two accounts are removed. The new account is assigned a new account number and title, and the account balance is the sum of both accounts. The history of previous transactions is preserved and sorted according to transaction ID (transactions with less transaction ID must come before the transaction with higher transaction ID).

Requirements:

1. Create a class '**Transaction**' to represent a transaction. This class must have attributes like transaction ID, transaction time, transaction date, transaction type, transferred amount, and the account number through which money is transferred.

2. Create a class `**Account**` to represent individual bank accounts. The class should include attributes such as account number, account title, account balance, and a doubly linked list (***List<Transaction> history***) to maintain the history of transactions.

Implement methods within the `Account` class to:

- Perform credit transactions (money added to the account).
- Perform debit transactions (money deducted from the account if available).
- Display the account details and transaction history.

3. Create a class `**Bank**` to manage multiple accounts using a doubly linked list (***List<Acount> accounts***).

Implement the following functionalities within the `Bank` class:

- Add a new account.
- Remove an existing account.
- Perform a merging of two accounts. When merging, create a new account with a new account number and title, sum the balances of the merged accounts, and merge their transaction histories while maintaining chronological order based on transaction ID. Implement the merging functionality efficiently, taking into account the preservation of transaction history order.
- Perform a transaction on an account. Given the Account number, transaction type, and amount to process, this function must find the respective account from the list of accounts and then perform the transaction if possible. In case the desired amount is not available as an account balance and the transaction type is debit, then just print an appropriate error message on the screen.

4. Use your implementation of List and Iterators to develop the above-mentioned classes.

Important Note:

1. This programming exercise involves creating classes for accounts and a bank, implementing transaction functionalities, and handling account merging while preserving transaction history. It requires the implementation of data structures linked lists, its iterator, and careful sorting of transactions. You might need to utilize friend relationships among classes of bank, account, and transaction to effectively implement this system.
2. Implement all required constructors (including copy constructor), destructor, and overloaded assignment operator.
3. Add two data members in the class bank: acc_Serial and trans_Serial. Initialize both of them to zero. Whenever a new account is created, use the value of acc_serial as the account number of the newly created account and then increment it. Similarly, whenever a new transaction is performed, use the value of trans_serial as the transaction ID and then increment trans_serial by one.