CS2710 - Programming and Data Structures Lab

Krishna Sivalingam

Lab 5

Sep. 6, 2023, 2pm – 5.30pm

Instructions

- Submission Instructions: (1) Submit the code on HackerRank; (2) Submit the code on Moodle; (3) The TA will inspect the code on the system and approve; the student can leave the Lab only AFTER gives permission.
- **Help from TAs will be provided only until 4.45PM.**
- After 4.45pm, TAs will be on verifying the submissions are done correctly on Moodle and HackerRank, and give permission to the student to leave.
- You are required to solve ALL the problems using C++.
- The questions are based on your training in programming in CS1111. So no additional inputs are needed, except the changes needed to switch from C to C++.
- If you need assistance, ask your TA, not your classmate.
- Please use g++ compiler.
- You can use online sites such as replit.c or onlinegdb.com for debugging purposes. However, the code must be downloaded to your DCF account and uploaded on **moodle.**

  The code must also be uploaded on the HackerRank page at:

  `https://www.hackerrank.com/cs2710-2023-lab5`
- **Please save your files locally every 2-3 minutes** so as to not lose your progress due to technical/Internet issues. Also, please note that in case you lose your code due to issues in HackerRank, additional time will not be provided.

Problems to be solved in lab

1. (15) Implement the doubly linked list (DLL) implementation, with header node and trailer node, to represent and compute transactions in a bank.

   The bank maintains a set of accounts. Assume that the transactions in the list are numbered as: $T_1, T_2, \ldots, T_n$. Each transaction specifies either a deposit (`AccountNo D y`) or a withdrawal (`AccountNo W x`). In the case of the deposit, the balance in the account is increased by y units, if the account number is present in the list of account numbers. Likewise, the balance is reduced in the case of withdrawals: note that negative balances are permitted.

There is a notion of a "cursor", which denotes the *most* recent transaction processed in the DLL. The cursor can be moved *forward* to process the specified next number of transactions. The cursor can be moved *backward* to **reverse** the specified next number of transactions.

The program will read a list of bank accounts (total of $C \leq 100$) and store it in a *singly linked list (SLL)*, with ONLY *Node *first*, Node *last* and *int len*. Each element in the SLL will store: (i) Account number (unsigned integer with value less than $10,000$); (ii) Current balance (can be positive or negative.) The initial balance of each account is 1000 units.

The program will read the entire set of transactions from input and store it in the same order in the DLL.

The cursor is initially set to the head node in the DLL. Subsequently, the program will read the following commands from input and process them as described below.

- F X // Process the next X transactions after the cursor; if *last* transaction is processed before completing X transactions, then the cursor stays in the *last* node.
- R Y // Undo Y transactions starting from the cursor; if *first* transaction is processed before completing Y transactions, then the cursor stays in the *first* node.
- I T K // Insert a new Transaction T **immediately after** the K^{th} transaction in the list. Insert only if K is a valid value. If inserting **before** cursor position, then process this new transaction as well. The cursor position does not change.
- D A M // Delete (up to) M ($\neq 0$) number of transactions of account A after/before (but excluding) the cursor's transaction; delete in the forward direction if $M > 0$ and reverse direction if $M < 0$. Update the balance of account A accordingly. The cursor stays in its original location.
- C // Process ALL transactions after the cursor.
- S Y // Print all the processed transactions (until the cursor) of Account Number Y, one transaction per line. If Y is invalid, no output is generated.
- G X // Print the count of the accounts whose balance is greater than or equal to X, with respect to the current cursor position.
- M // Print the Account Number with the highest balance, with respect to the current cursor position. If there is more than one account number with the same highest balance, print all numbers on one line separated by a single space, in increasing order.
- V X // Print the balance of Account Number X, with respect to the current cursor position.

Input: The inputs are:

- The first line specifies the number of accounts to be maintained, denoted as C.
- The second line contains C space-separated unsigned integers – each integer denotes an account number.
- The next line specifies the number of transactions to be maintained, denoted as N.
- The next set of N lines: each line either deposit or withdrawal.
  Insert only the transactions of valid account numbers to the list, in the same order as in the input file. Invalid account numbers must be ignored.
- Each following line contains a processing directive (as listed above).
- The last line contains only the word *END*.

Constraints: The constraints are:

- Deposit and Withdrawal amounts will be non-negative integers less than or equal to 50000.

Output: This will be the output of each query as applicable.

Example: An illustrative example is given below:

- *Input:* Output values are shown in comments, and are NOT part of the input file.

```
5
718 270 113 431 536
10
270 D 10
431 W 50
536 D 20
536 W 70
113 D 100
718 W 900
113 W 400
431 D 240
270 W 120
113 D 200
F 3
G 1010 // Output: 2
M // Output: 536
V 113 // Output: 1000
V 536 // Output: 1020
R 2
G 1010 // Output: 1
V 536 // Output: 1000
F 8
V 270 // Output: 890
D 270 -2
V 270 // Output: 1000
C
G 1010 // Output: 1
M // Output: 431
END
```

Output values are shown in comments, and are NOT part of the input file.

Note:
- Solutions that do not adhere to the constraints will not fetch credit.