University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Mathematics I

Program 6: The Jolly Banker

Purpose

This lab will serve a few purposes. First, it will provide hands-on experience using both queues and binary search trees. It will also provide an opportunity for further program/class design as the project does not specifically delineate class structure or design.

There are aspects of the spec below which allow interpretation—please read/design and ask questions early to clarify any ambiguity.

A **peer design review will be required** to help with the latter aspect. *This design review will require a deliverable to canvas and will be part of the final grade for the lab.*

Problem Overview:

You will build a banking application which processes transactions. This banking application consists of three phases.

1) The program will read in a string of transactions from a file into an in-memory queue. These transactions can open accounts, withdraw funds, deposit funds, transfer funds, or ask for the transactional history to be printed.

2) The program will next read from the queue and process the transactions in order.

3) When the queue has been depleted the program will print out all open accounts and balances in those accounts.

Details:

Input:

To test your program a file will be passed in by an argument to the program. The file will contain a list of transactions that need to be executed. Transactions of the format described below (see section on transactions) will be contained in this file. There will be one transaction per line.

Assume that the input is well-formed in the file—that is, there are no syntax errors. That said, there may be errors in the transactions themselves. For instance, a transaction may try to withdraw more money than there is a fund or try to withdraw for a non-existent fund. See the section below entitled errors for details.

Client Accounts and Funds:

Each client account contains assets held in up to ten funds. A client account is represented by a first and last name (two strings) and a unique 4 digit identifier. A fifth digit can be added to the digit id to represent the individual fund within the account as enumerated below:

```
0: Money Market           5: Capital Value Fund
1: Prime Money Market     6: Growth Equity Fund
2: Long-Term Bond         7: Growth Index Fund
3: Short-Term Bond        8: Value Fund
4: 500 Index Fund         9: Value Stock Index
```

For instance, 23452 represents the Long-Term Bond fund for client account 2345. A client account is opened as a transaction (see below).

Transactions:

There are five types of transactions and each are identified by a character beginning the line.

O: Open a client account with the appropriate funds

D: Deposit assets into a fund

W: Withdraw assets from a fund

T: Transfer assets between funds (can be funds owned by a single client or transfers between clients)

H: Display the history of all transactions for a client account or for a single fund. Include errors in the output where appropriate.

After this character key, the account-fund number is given followed by relevant information for the transaction.

Examples:

D 12341 100	Deposit $100 into the prime money market account of client ID 1234
W 12340 500	Withdraw $500 from the money market of client ID 1234.
T 12340 1000 12341	Transfer $1000 from client 1234's money market to the prime money market.
T 12340 1000 56780	Transfer $1000 from 1234's money market to 5678's money market.
H 1234	Display the history of all transactions of all accounts for client 1234.
H 12344	Display the history for all transactions on the 500 Index Fund for client 1234
O Bowden Charles 6537	Open an account for client Charles Bowden. Use account id 6537.

Errors:
As noted above, assume the format (number and types of input items) are correct. However, the program should deal with all other types of errors. For instance, there could be a bad account number (for instance, one already used) or an amount which is too much to withdraw. Also, assume that each line will begin with an appropriate letter: O, W, D, H, or T.

Examples of errors which may occur:

W 65439 10000 (when the Value Stock Index of client 6543 has only $20)
D 765435 76

A transaction that would cause a balance to become negative should not occur and is an error (A withdrawal or transfer of $0 is fine). **There is one exception to this rule:** if you are withdrawing from a money market fund with insufficient dollars, and it can be covered with dollars from the other money market fund owned by the client account, the desired amount (only the partial amount needed to cover the withdrawal) is moved to that money market account. The two Bond accounts are handled similarly. If one is transferring between two linked funds (say, 0 and 1) then the transaction should only succeed if the fund being withdrawn from has enough funds. No other types of accounts are handled in this manner.

Appropriate error messages should be printed out to `cout`. No other messages should be printed out during phase 1 or phase 2.

Output:

In Phase 3, each client account will be printed out with the amount held in each account. Please make sure to create an intuitive and readable output for this aspect of the program. This should be part of the design.

Data Structures:

One key aspect of this Lab exercise is the right class design for handling the problem. So in general there is no pre-defined classes or signature structure for the classes required. The suggestion is to keep the balances as **ints** (**doubles** or **floats** are not required). This is up to you to define. However, there are two data structures which are required.

First, you need to use a queue to read the transactions into during phase 1. All transactions should be read before processing starts. The queue can be the STL queue.

Second, the client accounts should be stored in a binary search tree (BSTree). Please note that for the design phase of the assignment you can assume the BSTree is just a container with functions like Retrieve, Insert, and Empty.

For this assignment, your binary search tree class only needs to implement the functions shown below (and supporting recursive helpers).

```
class BSTree
```

```
{
public:
        BSTree();
        ~BSTree();

        bool Insert(Account *);

        // retrieve object, first parameter is the ID of the account
        // second parameter holds pointer to found object, NULL if not found
        bool Retrieve(const int &, Account * &) const;

        // displays the contents of a tree to cout
        void Display() const;
        void Empty();
        bool isEmpty() const;

private:
        struct Node
        {
                Account *pAcct;
                Node *right;
                Node *left;
        };
        Node *root;
};
```

What to bring to Checkpoint peer design review

We will use class time to do peer design reviews. The class will break into groups of three or four and present the program design to each other. In order to prep for this aspect you will need to create a number of things to clearly articulate your design. Note that the intent is to have this inform your project as well. Your design will likely change after the presentation.

This peer design review is required and will be graded making up a certain % of the final project grade.

What is required to present your design?

- A brief (<1page) written document describing the design and how it all fits together
- Class diagrams
    - important Function Signatures
    - .h files are good reps so should be included

- Visuals of key data structures
- Explanation of how classes interact; that is describe how the design all fits together.

- Program flow (high level pseudo-code)

These will be turned through Canvas.

Sample Input/Output will be placed on Canvas.

See files: BankTransIn.txt and BankTransOut.txt.

What to turn in final project

As everyone will have different factoring please make sure to follow very closely what needs to be turned in. This is critical to get the correct grade for your project.

Here is what should be turned in:

- All .cpp and .h files required to make your project work
  - Including stdafx.h, stdafx.cpp, targetver.h if you use them.
- An output file called: BankTransOut.txt. This is Output of your program running the sample input found on Canvas. Note that the output from your program will not look exactly like the output from my program.
- The graders and I will build your code and run against new test cases copying the .h and .cpp files into a directory and running :
  - g++ *cpp or g++ -std=c++11 *cpp (to create an executable)

You should test your code this way. Assume the BankTransIn.txt is in that directory as well. If your code will not compile we cannot grade it.