

The Register Application

Purpose:

Until now, our application has been run by test cases in a main. We are not yet achieving the *end-user's* aims for our register capability.

This exercise is intended to give you experience with reaching the user-level requirements with a basic application. The use of a command-line/console interface prepares the application for one style of control, in a later exercise, by a web browser.

Since each of these user threads are independent, it is important that you step through, implement, and test each requirement individually, gradually building to the entire application. See if you can keep your application ready to compile and run—albeit with limited functionality—at all times.

This program is clumsier for a human to use than one with a nice human-machine interface—but it is *very* scriptable for machine-to-machine interaction, or, as we'll see, for a browser interface.

Problems:

Using the RegisterEntry hierarchy, and the Register container, create an application that will support the following requests:

1. Initiate a new Register, given a bank name, account identifier, and an initial balance
2. Add a check to the Register, given a check number, a payee, and an amount.
Optional: include an optional date, defaulting to today's date
3. Add a deposit to the Register, given an amount and a date
Optional: allow the date to default to today's date if not provided
4. Add a withdrawal to the Register, given an amount and a date
Optional: allow the date to default to today's date if not provided withdraw amount [date]
5. Mark a check with a given number as “cashed” on a statement
Optional: if the amount is provided by the user, double-check it against the register.
6. Mark a deposit of a particular amount on a given date as “deposited” on a statement
7. Mark a deposit of a particular amount on a given date as “withdrawn” on a statement
8. Retrieve the current balance of the account (see notes)
9. Retrieve the current sum of all unreconciled items (see notes)
10. Attempt a reconciliation, given a statement balance (see notes)
11. Print the entire register

Optional enhancement: Allow all of the above to specify a specific register (instead of a default).

Optional enhancement: Add additional commands to handle the other types in the register.

Notes:

It is OK to implement the entire exercise by depending on a default register. **Do Not** implement optional portions before completing the basic assignment.

A sample program that parses command line input is provided in the Sample Solution from the previous exercise. This may be a helpful guide to your efforts.

Use the persistent version of the register (or the sample solution version) to run this exercise, for best effect. With that perspective, the exercise can be seen as a modification of the test program from the previous exercise.

If your own Register and RegisterEntry implementations were limited, or for any other reason you wish, you may start the problem using the provided implementations in the Sample Solutions.

Business rules require that checks and withdrawals always *reduce* the balance (that is, they are a debit on the account), deposits always *increase* the balance (that is, they are a credit to the account). You may wish to implement rules on the user input that enforce that.

Balances, sum of unreconciled items, and reconciliations are the only portions of the problem with a mathematical algorithm, so they are described in detail below

Balances

Your register class may already cache the balance, in which case you just return it. The balance should agree with the sum of all the amounts from the register, including the initial balance.

Sum of Unreconciled Items

This is a sum of the amounts of all register entries which have not been marked as “on statement” yet.

Reconciliations

A reconciliation is *opened* (tried) whenever a user requests it. The reconciliation is *closed* (successfully completed) when

```
statement balance + total of unreconciled checks - total of
unreconciled deposits == register balance.
```

The sum “total of unreconciled checks - total of unreconciled deposits” is the *Sum of Unreconciled Items* described above.

Note that the user must provide only the statement balance; all other information is in the register.

Thus, the user should be notified of the *difference* between these two amounts so he or she can look for errors

A reconciliation entry is (automatically) created whenever a reconciliation is closed.

Evaluation criteria:

Each of the requested capabilities is weighted equally (5 points each), with 3 point emphasis on its structure and implementation, and 2 point emphasis on testing it.

Application structure and implementation (.cpp file)	45%
Specific capabilities and test output	55%