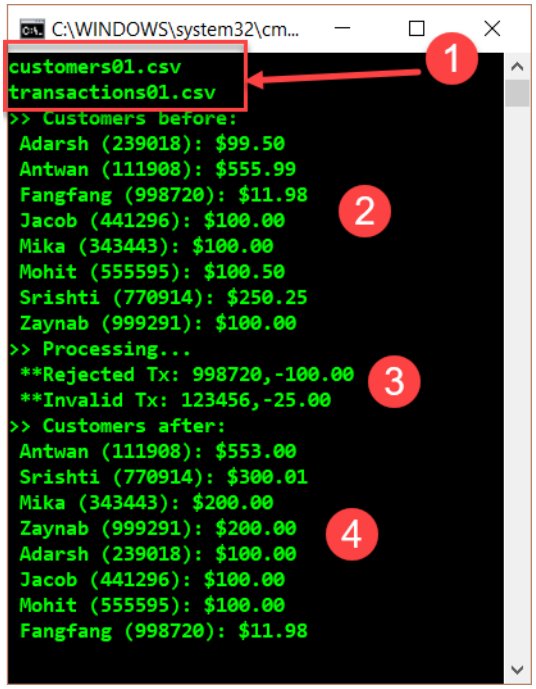
Project #01: Banking App

Complete By: Friday, January 26th @ 11:59pm
Assignment: C++ program to perform Banking Transactions
Policy: Individual work only, late work *is* accepted (see "Policy" section on last page for more details)
Submission: online via repl.it (exercise **P01 Banking App**)

Assignment

The assignment is to input customer data and banking transactions --- deposits and withdrawals --- and update the customer data accordingly. The program inputs the name of two files, the first containing customer data and the second containing banking transactions. The program outputs the customer data in ascending order by name, processes the transactions, and then outputs the updated customer data in descending order by customer balance --- if two customers have the same balance, the output is ordered by name (ascending).



The screenshot to the right shows the program output for the input files "customers01.csv" and "transactions01.csv". In step 1, the program inputs the 2 filenames and inputs the customer data. In step 2, the customers are output in order by name --- assume that customer names are unique. In step 3 the transactions are processed, with invalid transactions flagged and output as such. Finally, in step 4, the updated customer data is output.

Since most of you are relatively new to modern C++, approach this like a CS 1 assignment: one baby step at a time. Do not try to solve this assignment all at once. Also, since the program will be auto-graded using the repl.it system, your program must match this output exactly.

Programming Environment

To facilitate testing and program submission, we'll be using the repl.it system; see the exercise titled "**P01 Banking App**". You can work entirely within repl.it, or in the C++ environment of your choice; Visual Studio is not required for this assignment. Any environment with a modern (C++11 or newer) compiler will suffice; note that **bert** and **ernie** do not support modern C++, while **bertvm** does. If using g++, you may need to supply

the compiler option “-std=c++11” or “-std=c++14”.

Input Files

The program starts by inputting the name of two input files: a file containing customer data, and a file containing banking transactions. The customer data file is CSV file containing 1 or more customers, one per line. Each line contains 3 values: the customer’s name, ID (integer), and account balance (double). For example, here is the provided input file “customers01.csv”:

```
Mohit,555595,100.50
Adarsh,239018,99.50
Srishti,770914,250.25
Zaynab,999291,100.00
Mika,343443,100.00
Fangfang,998720,11.98
Antwan,111908,555.99
Jacob,441296,100.00
```

The banking transactions file is a CSV file containing 0 or more transactions, one per line. Each transaction consists of 2 values: a customer ID (integer), and an amount (double). For example, here is the provided input file “transactions01.csv”:

```
770914,+49.76
999291,+100.00
239018,+0.25
555595,+0.50
343443,+75.00
343443,-25.00
111908,+1.00
111908,+1.00
999291,-50.00
998720,-100.00
111908,+1.00
123456,-25.00
555595,-1.00
343443,+50.00
111908,-5.99
999291,+100.00
999291,-50.00
239018,+0.25
```

A transaction is “invalid” if the ID number does not match a bank customer; the input file shown above contains an invalid transaction: **123456,-25.00**. A transaction is “rejected” if the customer’s account balance does not contain sufficient funds to support the transaction. For example, the transaction **998720,-100.00** is rejected because the customer only has $11.98 in their account.

These sample input files are available on the course web page under “Projects”, then “project01-files”. The files are being made available for each platform (Linux, Mac, and PC) in case you want to work outside of the repl.it environment. NOTE: the best way to download the files is *not* to click on them directly in dropbox, but to instead use the “download” button in the upper-right corner of the dropbox page.

Requirements

As is the case with all assignments in CS 341, how you solve the problem is just as important as simply getting the correct answers. You are required to solve the problem the "proper" way, which in this case means using modern C++ techniques: classes, objects, built-in algorithms and data structures, lambda expressions, etc. It's too easy to fall back on existing habits to just "get the assignment done."

In this assignment, your solution is required to do the following:

- Use **ifstream** to open / close the input file; see lecture #01 for an example of using ifstream, getline(file, line) to read input line by line, and **stringstream** to parse each line.
- Use one or more classes to store the input — at a minimum you must have a **Customer** class that contains data about each customer: their name, ID, and account balance. The design of that class is up to you, but at the very least (1) all data members must be private, (2) the class contains a constructor to initialize the data members, and (3) getter/setter functions are used to access/update the data members.
- Use **std::vector<T>** to store your objects. While other containers might be more efficient, you'll have a chance to use those in later assignments. In this project, we want everyone to use std::vector<T>.
- Whenever possible, use **range-based for** (aka "foreach") instead of index-based for loops.
- Use **std::sort** to sort the customers.
- No explicit pointers of any kind — no char *, no NULL, no C-style pointers
- No global variables whatsoever; use functions and parameter passing.

Writing code that is not modern C++ will result in a score of 0. Using a container other than std::vector? 0. No classes? 0. Using global variables? 0.

For online documentation, I typically use the site http://www.cplusplus.com/reference/. Another way is to google with the prefix **std::**, which refers to the C++ standard library. Example: for information about the C++ string class, google "std::string". For the vector class, google "std::vector".

Have a question? Use Piazza, not email

Do not email the staff with questions — use Piazza. Remember the guidelines for using Piazza:

1. *Look before you post* — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post a question. Posts are categorized to help you search, e.g. "Pre-class" or "HW".
2. Post publicly — only post privately when asked by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means…".

4. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, so don't hesitate to take a screenshot and post; see http://www.take-a-screenshot.org/ .

5. Don't post your entire answer — if you do, you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question --- in that case post only the fragment that denotes your question, and omit whatever details you can. If you must post the entire code, then do so privately --- there's an option to create a private post ("visible to staff only").

Submission

The program is to be submitted via http://repl.it: see the exercise "**P01 Banking App**". The grading scheme at this point is 100% correctness --- you must pass all test cases to obtain a score of 100. We do, however, expect basic commenting, indentation, and readability. You should also be using functions, parameter passing, and good naming conventions. Add your name to the header comment at the top of each .cpp and .h file.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is described here:

http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml .