Hack 14.0

Computer Science I
Data Processing Part I

Department of Computer Science & Engineering

University of Nebraska–Lincoln

Introduction

Hack session activities are small weekly programming assignments intended to get you started on full programming assignments. Collaboration is allowed and, in fact, *highly encouraged*. You may start on the activity before your hack session, but during the hack session you must either be actively working on this activity or *helping others* work on the activity. You are graded using the same rubric as assignments so documentation, style, design and correctness are all important.

Problem Statement

Data processing is fundamental to Computer Science and many other disciplines. In this hack you will start a mini-project in which you will process a large amount of transaction data from a financial institution. The data represents transactions that transfer funds from or to an account.[1]

The Data

Transaction data is represented as a flat file in CSV format. The first line indicates how many records are contained in the file. Each subsequent line represents a single transaction. A transaction includes the following pieces of data:

- A universally unique identifier (UUID), an alphanumeric designation that uniquely identifies the transaction

[1] The data we use is derived from financial data generated by PaySim, a transaction simulator, see https://www.kaggle.com/ntnu-testimon/paysim1

- The type of transaction, which may be one of the following: `PAYMENT`, `TRANSFER`, `WITHDRAWAL`, `DEBIT`, `DEPOSIT`
- The amount of the transaction
- The customer account number
- The customer account balance before the transaction
- The customer account balance after the transaction
- The transfer account number or merchant designation identifying where the funds are sent

A small example is provided in Figure 1.

```
10
BC377639-37CC-4824-81AF-60177418B46D,PAYMENT,14535.18,C1906093041,83310.00,68774.82,M95867054
A35686EA-BFE7-455E-A027-54D73968E6D3,PAYMENT,11367.98,C371579810,77199.32,65831.34,M1932650331
7721D4C1-5A2E-4AA0-8AC3-351DFC4FE84A,DEPOSIT,159266.75,C1387043512,7777736.40,7937003.15,C553899299
F8A44740-DE85-4477-9E5E-C090AABE0BF2,WITHDRAWAL,79628.70,C739752704,203042.00,123413.30,C579345824
043FD663-236E-4C02-B042-E2642AA6471E,DEBIT,3225.27,C750937249,99375.00,96149.73,C1232504595
E0EB18EC-4EE2-4206-8834-9465992D1B28,PAYMENT,11268.73,C385148617,167267.40,155998.67,M1084205719
C306CA7E-80A0-499E-8CDE-13DFB50C6753,PAYMENT,17526.96,C1474376627,332543.08,315016.12,M1944361847
003C45B2-1356-4C7F-8D8A-EBB7BF026F16,DEPOSIT,149504.20,C386278926,56082.00,205586.20,C1113941243
B7ECC2F3-E455-4EF8-B4DF-55150A9F3B1F,PAYMENT,7676.11,C2018933315,542758.39,535082.28,M1018630839
76947698-1302-47D9-9CAF-66F21A2A6A52,PAYMENT,3240.99,C208944046,4778.00,1537.01,M962275185
```

Figure 1: Example data file.

Modeling & Loading the Data

To start, you'll need to write a C program that reads and processes a data file containing financial transactions in the format described above. You should design and implement a good model for this data as you'll use this code as the basis for producing several reports.

In addition, you should design at least 1 non-trivial test input file that has at least 2 examples of each type of transaction. You should name your file `transactions.csv` and hand it in.

Reporting the Data

To ensure that your data parsing works, you'll need to produce a report that, for each type of transaction, produces a total number of transactions as well as a total of all amounts of those transactions. For example, your report may look something like the output in Figure 2.

Your program should take the name of the input file as a command line argument and output the report to the standard output.

```
Totals
=======================================
Type            Count             Total
=======================================
Payment          4324  $    10764506.42
Transfer         3212  $    88261299.67
Withdraw         2213  $   274581826.82
Debit            2989  $     1213810.98
Deposit          4885  $    34425549.07
=======================================
Total           13299  $   409246992.96
```

Figure 2: Simple aggregate report output example

Instructions

- Place all of your function definitions in a source file named `transaction.c` and hand it in with your header file, `transaction.h`. Your main function should be placed in a file named `transactionReport.c`. This function will produce the report above.
- Turn in your test case file, `transactions.csv` to the webhandin as well.
- You are encouraged to collaborate with any number of students before, during, and after your scheduled hack session.
- Include the name(s) of everyone who worked together on this activity in your source file's header.
- Turn in all of your files via webhandin, making sure that it runs and executes correctly in the webgrader. Each individual student will need to hand in their own copy and will receive their own individual grade.