Queen Mary
University of London

Individual Assignment: Banking System

Student Name: Luca J. Ricagni

Student Number: 200894968

Module Name: Functional Programming

Module Code: ECS713P/ECS713U

Due Date: 20/01/2021

Instructions to compile and run code

To run this code first open the project folder using VS Code, then execute the command *'stack build'* into the PowerShell terminal within VS Code to compile the program. Next navigate to the *App* directory using *'cd App'*. The program can now be run in one of two ways: concurrently or in parallel, using the commands *'./Main concurrent +RTS -N10'* and *'./Main parallel +RTS -N10'* respectively. Attempting to run the program using another argument will result in an error.

The reader should note the use of *'N10'* which specifies the number of threads allocated to the program. Previously *'N6'* was used, however when stress testing the program it crashed. This was found to be because each account was not guaranteed its own thread. Therefore using *'N10'*, while not strictly necessary for this project, was used to ensure it would work with more transactions.

Design choice justification

It was decided that each major element of the code base, e.g. the *Customer* datatype or *Transaction* functions, would each be contained within their own module. This was done to keep the code base simple and easy to maintain. Each of these modules would then only export functions which were needed in other modules, any internal functions would not be exported. This was again done to keep the code base as simplistic as possible.

Several methods of storing a customer's balance were considered, but it was concluded that storing it using an *MVar Rational* would be the best way, as it enabled each person's balance to be altered rather than having to create a new one. This also has the advantage of only allowing one thread to act on a given account at once, due to the way the *MVar* datatype works. Similarly, it was also decided to store the number of completed transactions in an *MVar*. This would mean that only one transaction across all threads could take place at once, ensuring that they could not interfere with each other, as well as guaranteeing that exactly the required amount would take place.

When randomising the recipient customer, it quickly became clear that a method of checking whether the randomised customer was the same as the current customer was needed. As such the *Customer* datatype was made an instance of the *Eq* class, enabling direct comparison.

When adding the parallelism feature it was decided to use '*Control.Concurrent.ParallelIO'* rather than *'Control.Parallel'*. This was because it was found that the former was simpler to use and provided greater control than the latter.

Issues faced

When attempting to run the program it seemed to end very quickly, after some testing it was realised that when using *forkIO* to put the bank accounts on their own threads, if the *main* thread ended then the program would too. Two possible solutions were found for this, either a delay could be placed in *main* or an empty *MVar* could be used to prevent *main* from running until it was filled. After some consideration it was decided to use the second option. This was because the time delay used would have to be approximated, which may cause problems on machines of different speeds, or if more transactions were required. The *MVar* option on the other hand would ensure that the program would run until all transactions had been completed, regardless of the number required.

Another issue identified was printing a customer's information in a formatted manner at the end of the program. The first method attempted was to simply make the *Customer* datatype a user defined instance of the *Show* class. However, this proved unsuccessful as in order to do this the *show* function must be called on a customer's balance, which cannot be done within a user defined instance of a class. It was therefore decided to create a bespoke function to print a customer's information in a nicely formatted way.