CS 2024 Final Project: How-To Document

Chesley Tan (ct353), Qingjun Wang (qw77), Shaoke Xu (sx77)

For our final project, we aimed to create an expense/debt tracker application that could be used not only to log transactions, but also to simplify debts between a group of individuals so as to minimize the amount of money that has to be exchanged in order to resolve the debts. We decided to create a command-line application since that would be simple to implement and easy-to-use. In terms of the user interface, we settled on a command-based design for user interaction since this would be easy for a user to quickly learn how to use. Since some commands, such as `add` for adding a transaction require the user to provide additional information, like the name of the payer and the name of the debtor, we chose to use a series of prompts asking for each individual required argument. This allowed us to validate the inputs as they were provided by the user and resulted in a streamlined process for adding new transactions.

To achieve this user interface, we needed an input loop and we needed to be able to parse the user's commands, so we created the main application file, `main.cpp`, which contains the code for prompting the user, reading the user's input, validating the user's input, and executing the user's command, which may be one of `add`, `log`, `summary`, `analytics`, `quit`, or `help`.

Since we need to be able to persistently store the transactions, we chose to use SQLite for our database, because it is relatively simple to use, it provides a C/C++ API along with source code, and is well-suited for single-user applications. We created a class `DatabaseHelper` to handle reading and writing to the database. By separating the implementation of the main program from the implementation of the database code, we were able to develop the two in parallel. The transactions added by the user are stored into a `Transactions` table which has columns for storing the name of the payer, name of the debtor, amount of the debt, and an optional description of the transaction. The calculated debts between pairs of individuals are stored in the `Debts` table, which has columns for storing the name of the payer, name of the debtor, and the amount of the debt. When a transaction is added using the add command, a new row is inserted into the Transactions table, and the Debts table is updated to reflect the new debt values between the payer and the debtor; depending on the existing debt value, either a new row is inserted, a row is deleted and a new one is inserted, or an existing row is updated.

The algorithm for simplifying the debts is implemented in `simplify.cpp`, which is used in `main.cpp` to handle the `summary` command. The algorithm works by first calculating the total debts for each individual involved in the transactions and then re-assigning the debts among pairs of individuals so as to minimize the amount of money that has to be exchanged as well as ensuring that all payers are repaid correctly and all debtors owe the correct amount of money to others.

These separate components are all tied together by the `handle_*_cmd()` functions in main.cpp which execute the parsed user command.