Wallet Program Developer Document

Overview of the Solution

This program is made in C to manage financial transactions effectively. It uses dynamic memory to handle different numbers of transactions and categories. It defines two main structures: Transaction and Wallet, to organize the data. The program includes functions that allow users to add, edit, and save transactions.

Data Structures

Transaction: Stores a single transaction's details (date, type, category, amount).

Wallet: Manages a collection of transactions, their capacity, and a list of categories.

Memory Management

The transactions array in the Wallet struct dynamically resizes using realloc as more entries are added.

Categories are stored in dynamically allocated strings within a categories array.

Modules

Initialization and cleanup (e.g., initWallet, freeWallet)

File operations (e.g., loadDatabase, saveDatabase)

Transaction management (e.g., addEntry, editEntry, deleteEntry)

Summarization and reporting (e.g., displayTotals, displayStats)

Dynamic Reallocation

For transactions and categories, memory grows when capacity is exceeded. This ensures scalability without memory wastage.

String Matching

Category existence is verified using strcmp, ensuring no duplicates in the category list.

Date Filtering

Transactions within a date range are selected by comparing strings lexicographically, as dates are stored in YYYY/MM/DD format.

Functions and Interfaces

Initialization and Cleanup

void initWallet(Wallet* wallet)

Initializes a wallet with empty transactions and no categories.

Input: Pointer to a Wallet struct

Return: None

void freeWallet(Wallet* wallet)

Frees allocated memory for transactions and categories.

Input: Pointer to a Wallet struct

Return: None

File Operations

void loadDatabase(Wallet* wallet, const char* filename)

Reads transaction data from a file and populates the wallet.

Input: Wallet pointer, filename as string

Return: None

void saveDatabase(const Wallet* wallet, const char* filename)

Writes transaction data to a file.

Input: Wallet pointer, filename as string

Return: None

Transaction Management

void addEntry(Wallet* wallet, const char* date, const char* type, const char* category, double amount)

Adds a transaction to the wallet.

Input: Wallet pointer, transaction details

Return: None

void editEntry(Wallet* wallet, size_t index, const char* date, const char* type, const char* category, double amount)

Modifies a transaction by index.

Input: Wallet pointer, index, transaction details

Return: None

void deleteEntry(Wallet* wallet, size_t index)

Removes a transaction by index.

Input: Wallet pointer, index

Return: None

Summarization

void displayTotals(const Wallet* wallet)

Displays total income, expenses, balance, and top expense category.

Input: Wallet pointer

Return: None

void displayStats(const Wallet* wallet, const char* startDate, const char* endDate)

Displays financial statistics within a date range.

Input: Wallet pointer, start and end dates

Return: None

Utility

void addCategory(Wallet* wallet, const char* category)

Adds a new category if it doesn't exist.

Input: Wallet pointer, category string

Return: None

void listCategories(const Wallet* wallet)

Lists all categories in the wallet.

Input: Wallet pointer

Return: None

Conclusion

It provides support and directions to users and developers for understanding, using and altering the Wallet program. On the further development side, the modular design allows developers to continue adding extra features.