Credit Card System

Rupin Mittal
UML Pseudocode
Aug 8 2020

There is a program that randomly generates a record with many transactions (date, amount, location). A structure is defined that represents each transaction. An array called the record is filled with all the transactions in the order that they are generated. Functions are created to be able to sort this record, to print it in various formats, and to add transactions to it.

CreditCardSystem.h (Header file)

A. Defines the struct that represents a transaction
B. Declares but does not define the record array that holds the transactions
C. Declares but does not define variable that holds number of transactions

struct transaction{};	//definition of transaction
extern struct transaction record[]	//extern variable of the record array
extern int numTransactions	//number of transactions in record

RecordHeader.h (Header file)

A. Has function declarations from all the other files that RecordKeeper needs to be able to access.

RecordKeeper.c

A. Contains that main() function
B. Calls the function to randomly generate the credit card data
C. Prints the user's menu options
D. Takes the user's input for their option
E. Starts user loop with user's menu options and handles the requests by calling the appropriate functions
F. Has function for adding transactions
G. Holds number of transactions in the record

int menu()	//menu runs the user's option loop
void printMenu()	//holds the menu options
void addTransaction()	//protocol for adding a transaction to record

RecordGenerator.c

A. Define the record array
B. Random will be used to create a random number, that will be converted to a date

C. Random will be used to pick one of the many locations in an multidimensional array (Chipotle, Target, and so on)
D. Random will be used to pick the transaction amount
E. All these will be loaded into the record array

void randomDate(char *date)
void randomLocation(char *location)
int randomAmount(int lower, int higher)
void generateRecord()
	//for date
	//for location
	//for amount
	//will create a struct for each transaction (by getting random member values) and load them into record array

RecordSorter.c

A. Has quickSort function written from scratch to be able to sort the record
B. Has swap and partition helped functions to aid with quickSort
C. Has compareDate and compareAmount callback functions whose pointers will be passed to quickSort function so that 4 different sort orders can be met with only 1 sorting function (avoid duplicate code)
D. The int direction parameter in the compareDate and compareAmount functions determines the top/down and down/top orders so that each function handles 2 orders

int compareDate(int firstIndex, int secondIndex, int direction)	//compare transactions by date (1 and - 1 for direction)
int compareAmount(int firstIndex, int secondIndex, int direction)	//compare transactions by amount (1 and - 1 for direction)
void quickSort(int (*fPtr)(int, int, int), int direction, int left, int right)	//sort the record
void swap(int index1, int index2)	//helper function for quickSort
int partition(int (*fPtr)(int, int, int), int direction, int left, int right)	//helper function for quickSort
void sortRecord(int type, int direction)	//calls quicksort and passes it correct compare callback function with correct direction

RecordPrinter.c

A. Can print record to the system screen
B. Can print record to the “Record.txt” file

void printRecordToFile() void printRecordToSystem()	//print to Record.txt file //print to system screen