MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG315 Algorithms

Assignment 1: Transaction Data Management

This assignment aims to help you practice file operations, data structures, sorting algorithms (specifically Heap Sort) and algorithm analysis in a practical manner. It will also allow you to practice data processing and analysis.

Overview

In this assignment, your task is to create a **C program** that reads transaction data from a text file, processes it, and organises it into customer structures. You will use the Heap Sort algorithm to sort these customer structures based on four options:

1. The total number of transactions;
2. The total number of items purchased across all transactions;
3. The total amount of money spent (calculated as the sum of cost per item multiplied by the number of items purchased for each transaction);
4. The average amount of money spent for each transaction (calculated by dividing the total amount of spent by the total number of transactions).

Requirements

- **Read Transaction Data:** Your program should read transaction data from a text file. The data is structured in columns, including Name, Transaction ID, Transaction Time, Item Description, Number of Items Purchased, Cost Per Item, and Country, with each line representing a single transaction.

- **Create Customer Structures**: As the program reads the data, it should create a customer structure for each unique customer. These customer structures should be kept in an array which will be dynamically allocated based on the number of unique customers in the transaction data file. A customer structure should store name (max 80 characters), the total number of transactions, the total number of items purchased across all transactions, and the total amount of money spent.

  - **Total Number of Transactions**: Count the number of transactions for each customer by incrementing a counter for each transaction the customer appears in.
  - **Total Number of Items Purchased**: Sum the number of items purchased in each transaction for each customer.
  - **Total Amount of Money Spent**: Calculate the total amount of money spent by multiplying the cost per item by the number of items purchased for each transaction and summing these values for each customer.

- **Sort Customer Structures:** Implement the Heap Sort algorithm to sort the customer structures based on one of four sorting options:

  - **Option 1:** Sort by the total number of transactions for each customer.
  - **Option 2:** Sort by the total number of items purchased across all transactions for each customer.
  - **Option 3:** Sort by the total amount of money spent across all transactions for each customer.

   - **Option 4:** Sort by the average amount of money for each transaction for each customer.

- **Menu Display:** Provide a menu that allows the user to select one of the four sorting options and display the sorted customer structures accordingly.

- **Output Results:** After sorting, print the results with the respective customer information, including name, total number of transactions, total number of items purchased, total amount of money spent, and average money spent per transaction.

Input

The program should take input commands from the user and data in a text file called "transactions.txt" containing transaction data in the following format:

```
Name;TransactionId;TransactionTime;ItemDescription;NumberOfItemsPurchased;C
ostPerItem;Country
Alice;6355745;02/02/2019;LONDON BUS COFFEE MUG;6;11.73;United Kingdom
Bob;6283376;12/26/2018;SET 12 COLOUR PENCILS DOLLY GIRL;3;3.52;France
Charlie;6385599;02/15/2019;PACK OF 12 SUKI TISSUES;72;0.9;Germany
...
```

Please note that the content of the data file can be changed, but the name of the file will not be changed, and the structure should be in the format as shown above.

Internal Processing

As the program reads the data, it should maintain customer structures for each unique customer and update the total number of transactions, total number of items purchased, and total amount of money spent for each customer. These customers structures should be stored in an array.

The Heap Sort algorithm should be used to sort the customer structures based on the selected sorting option (total number of transactions, total number of items purchased, total amount of money spent, average amount of money for each transaction). After sorting, the program should display the results according to the user's choice.

In this assignment, you are given a C file called "template.c" attached. This will be the template code which you need to **strictly** follow. Inside the "template.c" file, all function prototypes are declared, the main function is already written along with the menu function and printCustomers function. You will basically need to fill in the blanks and write the code for the following functions:

- **countCustomers:** This function will take the file pointer of the transactions file as a parameter and read the content, and as it is doing so, it will count the number of unique customers found in the dataset and return it. This number will be used in the main function to dynamically allocate the memory for the customers array. This last bit is already done in the main function, so you do not need to worry about it.
- **readTransactions:** This function will read the transactions file again but this time, you will read and process customer information as you try to populate the customers array. Every time you read a data line, you will check if you have added the customer in that line to the array already, if yes, then you just need to update their info in the array, otherwise you will need to add them in a different spot.
- **heapSort:** This function implements heap sort, it will take in the customers array, its length, and the sorting criteria, and it will sort the array accordingly.
- **heapify:** This function will be used by the heapSort function to perform sorting.

Important Notes for Implementation

- The template code assumes the data file you are going to read is called "transactions.txt" and is stored directly in the code's immediate directory. Note that if you are using an IDE like Clion, you will need to store the file in the "cmake-build-debug" folder.
- You are not allowed to change any code in the template, and you should abide by the prototypes and data structures given. The places where you need to find write your code are marked with the comment: "WRITE YOUR CODE HERE".
- There are data columns in the file that you do not need for your program such as the transaction time and country, so you do not need to process those data columns in your program.
- Please note that scanf and its variants (e.g. fscanf) will read until space when you use %s, one thing you can use when reading the file is %[^;] which would read until semi-colon.
- You need to ensure usage of comments to clarify your codes.

Incremental and Modular Development

You are expected to follow an incremental development approach. Each time you complete a function or module, you are expected to test it to make sure that it works properly. You are also expected to follow **modular programming** which means that you are expected to divide your program into a set of meaningful functions. Specifically, you are expected to implement some helper functions.

Output

You can find the sample output below for a given transaction file:

```
There are 9 unique customers
List of customers:
Name: Alice, number of transactions = 4, number of items = 78, total amount paid
= 299.22, average amount paid per trans
action = 74.80
Name: Bob, number of transactions = 4, number of items = 51, total amount paid
= 113.52, average amount paid per transaction = 28.38
Name: Charlie, number of transactions = 4, number of items = 81, total amount
paid = 181.98, average amount paid per transaction = 45.50
Name: David, number of transactions = 4, number of items = 135, total amount
paid = 524.76, average amount paid per transaction = 131.19
Name: Eve, number of transactions = 4, number of items = 123, total amount paid
= 267.60, average amount paid per transaction = 66.90
Name: Ash, number of transactions = 5, number of items = 11, total amount paid
= 517.89, average amount paid per transaction = 103.58
Name: Aissen, number of transactions = 5, number of items = 20, total amount
paid = 1354.80, average amount paid per transaction = 270.96
Name: Jack, number of transactions = 5, number of items = 16, total amount paid
= 529.84, average amount paid per transaction = 105.97
Name: Kale, number of transactions = 5, number of items = 16, total amount paid
= 854.84, average amount paid per transaction = 170.97
Please choose one of the following options:
(1) Display customers sorted by number of transactions
(2) Display customers sorted by number of items purchased
(3) Display customers sorted by total amount paid
(4) Display customers sorted by average amount paid per transaction
(5) Exit

Command:1
 List of customers:
Name: Bob, number of transactions = 4, number of items = 51, total amount paid
= 113.52, average amount paid per transaction = 28.38
```

```
Name: Eve, number of transactions = 4, number of items = 123, total amount paid
= 267.60, average amount paid per transaction = 66.90
Name: David, number of transactions = 4, number of items = 135, total amount
paid = 524.76, average amount paid per transaction = 131.19
Name: Alice, number of transactions = 4, number of items = 78, total amount paid
= 299.22, average amount paid per trans
action = 74.80
Name: Charlie, number of transactions = 4, number of items = 81, total amount
paid = 181.98, average amount paid per transaction = 45.50
Name: Jack, number of transactions = 5, number of items = 16, total amount paid
= 529.84, average amount paid per transaction = 105.97
Name: Kale, number of transactions = 5, number of items = 16, total amount paid
= 854.84, average amount paid per transaction = 170.97
Name: Ash, number of transactions = 5, number of items = 11, total amount paid
= 517.89, average amount paid per transaction = 103.58
Name: Aissen, number of transactions = 5, number of items = 20, total amount
paid = 1354.80, average amount paid per transaction = 270.96
Please choose one of the following options:
(1) Display customers sorted by number of transactions
(2) Display customers sorted by number of items purchased
(3) Display customers sorted by total amount paid
(4) Display customers sorted by average amount paid per transaction
(5) Exit

Command:2
 List of customers:
Name: Ash, number of transactions = 5, number of items = 11, total amount paid
= 517.89, average amount paid per transaction = 103.58
Name: Kale, number of transactions = 5, number of items = 16, total amount paid
= 854.84, average amount paid per transaction = 170.97
Name: Jack, number of transactions = 5, number of items = 16, total amount paid
= 529.84, average amount paid per transaction = 105.97
Name: Aissen, number of transactions = 5, number of items = 20, total amount
paid = 1354.80, average amount paid per transaction = 270.96
Name: Bob, number of transactions = 4, number of items = 51, total amount paid
= 113.52, average amount paid per transaction = 28.38
Name: Alice, number of transactions = 4, number of items = 78, total amount paid
= 299.22, average amount paid per trans
action = 74.80
Name: Charlie, number of transactions = 4, number of items = 81, total amount
paid = 181.98, average amount paid per transaction = 45.50
Name: Eve, number of transactions = 4, number of items = 123, total amount paid
= 267.60, average amount paid per transaction = 66.90
Name: David, number of transactions = 4, number of items = 135, total amount
paid = 524.76, average amount paid per transaction = 131.19
Please choose one of the following options:
(1) Display customers sorted by number of transactions
(2) Display customers sorted by number of items purchased
(3) Display customers sorted by total amount paid
(4) Display customers sorted by average amount paid per transaction
(5) Exit

Command:3
 List of customers:
Name: Bob, number of transactions = 4, number of items = 51, total amount paid
= 113.52, average amount paid per transaction = 28.38
Name: Charlie, number of transactions = 4, number of items = 81, total amount
paid = 181.98, average amount paid per transaction = 45.50
Name: Eve, number of transactions = 4, number of items = 123, total amount paid
= 267.60, average amount paid per transaction = 66.90
Name: Alice, number of transactions = 4, number of items = 78, total amount paid
= 299.22, average amount paid per trans
action = 74.80
Name: Ash, number of transactions = 5, number of items = 11, total amount paid
= 517.89, average amount paid per transaction = 103.58
```

```
Name: David, number of transactions = 4, number of items = 135, total amount
paid = 524.76, average amount paid per transaction = 131.19
Name: Jack, number of transactions = 5, number of items = 16, total amount paid
= 529.84, average amount paid per transaction = 105.97
Name: Kale, number of transactions = 5, number of items = 16, total amount paid
= 854.84, average amount paid per transaction = 170.97
Name: Aissen, number of transactions = 5, number of items = 20, total amount
paid = 1354.80, average amount paid per transaction = 270.96
Please choose one of the following options:
(1) Display customers sorted by number of transactions
(2) Display customers sorted by number of items purchased
(3) Display customers sorted by total amount paid
(4) Display customers sorted by average amount paid per transaction
(5) Exit

Command:4
 List of customers:
Name: Bob, number of transactions = 4, number of items = 51, total amount paid
= 113.52, average amount paid per transaction = 28.38
Name: Charlie, number of transactions = 4, number of items = 81, total amount
paid = 181.98, average amount paid per transaction = 45.50
Name: Eve, number of transactions = 4, number of items = 123, total amount paid
= 267.60, average amount paid per transaction = 66.90
Name: Alice, number of transactions = 4, number of items = 78, total amount paid
= 299.22, average amount paid per trans
action = 74.80
Name: Ash, number of transactions = 5, number of items = 11, total amount paid
= 517.89, average amount paid per transaction = 103.58
Name: Jack, number of transactions = 5, number of items = 16, total amount paid
= 529.84, average amount paid per transaction = 105.97
Name: David, number of transactions = 4, number of items = 135, total amount
paid = 524.76, average amount paid per transaction = 131.19
Name: Kale, number of transactions = 5, number of items = 16, total amount paid
= 854.84, average amount paid per transaction = 170.97
Name: Aissen, number of transactions = 5, number of items = 20, total amount
paid = 1354.80, average amount paid per transaction = 270.96
Please choose one of the following options:
(1) Display customers sorted by number of transactions
(2) Display customers sorted by number of items purchased
(3) Display customers sorted by total amount paid
(4) Display customers sorted by average amount paid per transaction
(5) Exit

Command:5
 Goodbye!
```

Algorithm Analysis

As part of this assignment, you are also expected to submit the time complexity analysis of the **readTransactions** function, as shown in the example below. In your analysis, you need to assume that you have n records in your transaction data file with m unique customers.

Code	Cost	Times
`i = 1;`	c1	1
`sum = 0;`	c2	1
`while (i<n){`	c3	n+1
`      i = i + 1;`	c4	n
`    sum = sum + i;`	c5	n
`}`		

```
Total cost = c1 + c2 + c3 * (n+1) + c4 * n + c5 *n
The time required for this algorithm is proportional to n
```

You need to submit the time complexity analysis of your **readTransactions** function as a .pdf file.

Submission

You create a folder called CNG315-A1 with a single .c file for your implementation and a single .pdf file for the time complexity analysis of your **readTransactions** function. The name of the .c and .pdf files should be your 7-digit student id number. You need to submit the compressed version of the folder to ODTUCLASS.

Grading Scheme

The assignment will be graded based on the following scheme:

Grading Item	Mark (out of 100)
countCustomers	20
readTransactions	30
heapSort	15
heapify	20
Time complexity analysis of readTransactions	15

The assignment will be graded as follows:

- **countCustomers** and **readTransactions** will be tested with a different transaction data file. If it does not create an expected output, then **<u>you will not receive any mark from these functions</u>.**
- **heapSort** and **heapify** will be tested together for each sorting option, and if a particular sorting option does not work as expected, **<u>you will not receive any mark from that option</u>.**
- The time complexity of readTransactions will be manually checked as follows:
  - 15 marks for a correct solution with no problem
  - 10 marks for a solution with minor problems
  - 5 marks for a solution with major problems
  - 0 marks for an incorrect solution

You need to ensure your code runs and is testable. **If you submit a code which does not compile, you will automatically get zero.** Please note that code quality, modularity, efficiency, maintainability, and appropriate comments will be part of the grading. Therefore, **<u>providing an expected output does not guarantee a full mark</u>.**

Professionalism and Ethics

You are expected to complete the assignments on your own. Sharing your work with others, uploading the assignment to online websites to seek solutions, and/or presenting someone else's work as your own work will be considered as cheating.

Please note that we will use a tool to compute the similarity between your submissions, and there are also tools to detect if the code you submitted is generated by AI tools.