CS 100 Project Four – Spring 2019

Project Overview: In this project, you will implement a set of functions that can be used to retrieve information from a list of real estate transactions stored in a CSV (comma-separated values) file. The CSV file consists of 12 columns with the following column names and types.

- street (a string).
- city (a string).
- zip (a string).
- state (a string).
- beds (a number).
- baths (a number).
- sq_ft (square feet, a number).
- type (a string).
- sale_date (a string).
- price (a number).
- latitude (a number).
- longitude (a number).

For simplicity, assume the order of these 12 columns is fixed exactly as shown above. The first row (or line) of the CSV file is the header row that consists of column names. Each following row represents a real estate transaction. Some values of a row/transaction could be blank (unknown). A row will be excluded from a calculation if the calculation involves its blank value or a test condition involves its blank value (to be explained later). The maximum number of transactions or the order of transactions in the file is unknown in advance.

You are asked to implement the following six functions in the **`functions.c`** file. Every function to be implemented has two parameters, `testc` and `testv`, that can be used to test whether a row should be included or excluded in a calculation. `testc` and `testv` are similar to `argc` and `argv` in command line arguments of the `main` function. The difference is `testv` will contain `3*testc` strings. `testc` is the number of test conditions. `testv` contains three strings for each test condition. The first string will always be one of 12 column names in the header, as listed above. The second string is one of the six relational operators (`"=="`, `"!="`, `">="`, `"<="`, `">"`, and `"<"`). The third string is the reference value to be compared with. For example, assume `testc=2`, and `testv` contains six strings: `"beds" ">=" "3" "baths" ">=" "2"`. If a property has at least three bedrooms and at least two bathrooms, the property will be evaluated to be true for both conditions and it will be included in the computation. If a property does not have a value (blank) for the number of bedrooms or the number of bathrooms, the property will automatically be evaluated to be false and it will be excluded in the computation. If `testc=0`, it means every row/property will be automatically included in the computation.

- `void printAddr(char csvfile[], int testc, char *testv[]);` Given a CSV file, print out the addresses (street, city, state and zip) of the properties that satisfy all the specified conditions.
- `void printCoor(char csvfile[], int testc, char *testv[]);` Given a CSV file, print out the coordinates (latitude and longitude) of the properties that satisfy all the specified conditions.
- `int getCount(char csvfile[], int testc, char *testv[]);` Given a CSV file, return the number of the properties that satisfy all the specified conditions.
- `void getMin(char csvfile[], char column[], int testc, char *testv[], double *pMin, int *pCount);` Given a CSV file, return the minimum of the specified column from the properties that satisfy all the specified conditions through the `pMin` pointer. A property that has a blank value for the specified column will also be excluded from the calculation. The number of the properties that are included in the computation will be returned through the `pCount` pointer.
- `void getMax(char csvfile[], char column[], int testc, char *testv[], double *pMin, int *pCount);` Given a CSV file, return the maximum of the specified column from the properties that satisfy all the specified conditions through the `pMax` pointer. A property that has a

blank value for the specified column will also be excluded from the calculation. The number of the properties that are included in the computation will be returned through the `pCount` pointer.

- `void getAvg(char csvfile[], char column[], int testc, char *testv[], double *pAvg, int *pCount);` Given a CSV file, return the average of the specified column from the properties that satisfy all the specified conditions through the `pAvg` pointer. A property that has a blank value for the specified column will also be excluded from the calculation. The number of the properties that are included in the computation will be returned through the `pCount` pointer.

When implementing the above functions, you can assume `csvfile` is a valid CSV file as described above and it can always be opened for reading. When implementing the `getMin`, `getMax`, and `getAvg` functions, you can assume `column` is a valid column name and it is of numerical type. You can also assume the test conditions specified by `testc` and `testv` will also be valid and in correct form, as specified above. In addition, you are allowed to add helper functions in the **`functions.c`** file.

We recommend you use the `fgets` function to read a line from the CSV file, and you use the `strsep` function instead of the `strtok` function to extract each field from the line because a CSV file could contain blank cells. You can assume that each line does not exceed 300 characters in length and each field does not exceed 100 characters in length. `strsep` is destructive, please make a copy of the line before calling `strsep` if you need to reuse the line. The `strsep` function is a bit hard to use. Please search the web to find out how to use the `strsep` function properly, and it is probably a good idea to write a small program to test out the `strsep` function first.

For consistency, a numeric value is also stored as a string in a cell or in a test condition, you need to use `atoi` or `atof` to convert a numeric string to a number before comparison or computation. For a non-numeric string type, you shall use the `strcmp` function to perform case-sensitive comparison between two strings.

You can download **`main.c`** from Blackboard and you shall not modify anything in **`main.c`**. To compile this project, use the following command.

```
gcc -Wall -std=gnu99 main.c functions.c
```

Testing: a CSV file named **`Sacramentorealestatetransactions.csv`** can be downloaded from Blackboard for testing. You can test the program using the following command, and a sample execution of the program is shown at the end of this document.

```
./a.out Sacramentorealestatetransactions.csv
```

To verify whether you have implemented a function correctly, you can post the corresponding test commands and their results to Piazza and ask whether others agree with your results. **However, posting any part of C code from the project on Piazza is prohibited.**

What You Need To Do

- Create a directory named **`project4`** on your machine. Download **`main.c`** and **`Sacramentorealestatetransactions.csv`** to that directory, and create a file named **`functions.c`** under that directory.
- In **`functions.c`**, implement the six functions as specified above, and make sure to have a header block of comments that includes your name and a brief overview of your task.
- When you are ready to submit your project, compress your **`project4`** directory into a single (compressed) zip file, **`project4.zip`**.
- Once you have a compressed zip file named **`project4.zip`**, submit that file to Blackboard.

Project 4 is due at 5:00pm on Friday, March 8. Late projects are not accepted.

**This document including its associated files is for your own personal use only.
You may not post this document or a portion of this document to a site
such as chegg.com without prior written authorization.**

**A project shall be completed individually, with no sharing of code or solutions.
All submissions will go through MOSS (Measure Of Software Similarity) for similarity check.
The University of Alabama's Code of Academic Conduct will be rigorously enforced.**

A sample execution of the program

```
./a.out Sacramentorealestatetransactions.csv

Enter a command: count

There are 985 records

Enter a command: count1
Enter test condition #1: zip == 95632

There are 21 records

Enter a command: count1
Enter test condition #1: type == Condo

There are 54 records

Enter a command: addr2
Enter test condition #1: beds == 5
Enter test condition #2: zip == 95757

1: 7105 DANBERG WAY, ELK GROVE, CA 95757
2: 6945 RIO TEJO WAY, ELK GROVE, CA 95757
3: 6503 RIO DE ONAR WAY, ELK GROVE, CA 95757
4: 9688 NATURE TRAIL WAY, ELK GROVE, CA 95757
5: 9629 CEDAR OAK WAY, ELK GROVE, CA 95757
6: 5908 MCLEAN DR, ELK GROVE, CA 95757
7: 4821 HUTSON WAY, ELK GROVE, CA 95757
8: 9677 PILLITERI CT, ELK GROVE, CA 95757

Enter a command: coor1
Enter test condition #1: city == RIO LINDA

1: (38.700909, -121.442979)
2: (38.689591, -121.452239)
3: (38.689999, -121.463220)
4: (38.693818, -121.441153)
5: (38.702893, -121.454949)
6: (38.700553, -121.452223)
7: (38.691104, -121.451832)
8: (38.687659, -121.463300)
9: (38.687021, -121.463151)
10: (38.683674, -121.435204)
11: (38.687172, -121.463933)
12: (38.682790, -121.453509)
13: (38.695589, -121.444133)

Enter a command: min beds

Among 877 eligible properties, min(beds)=1
```

```
Enter a command: min1 beds
Enter test condition #1: price >= 300000

Among 214 eligible properties, min(beds)=2

Enter a command: max baths

Among 877 eligible properties, max(baths)=5

Enter a command: max1 baths
Enter test condition #1: city == ELK GROVE

Among 107 eligible properties, max(baths)=4

Enter a command: avg price

Among 985 eligible properties, avg(price)=234144

Enter a command: avg1 price
Enter test condition #1: price >= 400000

Among 113 eligible properties, avg(price)=514073

Enter a command: avg sq_ft

Among 814 eligible properties, avg(sq_ft)=1591.15

Enter a command: avg2 sq_ft
Enter test condition #1: price >= 300000
Enter test condition #2: price <= 500000

Among 145 eligible properties, avg(sq_ft)=2252.85

Enter a command: avg1 sq_ft
Enter test condition #1: street == 7105 DANBERG WAY

Among 1 eligible properties, avg(sq_ft)=3164

Enter a command: quit
```