ECE 209
Fall 2019

Program 3: `market`

Due Friday, Oct 23 @ 11:59pm

This programming assignment must be completed individually. Do not share your code with or receive code from any other student. Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of 0 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

Assignment Project Exam Help

For this assignment, you will be reading and analyzing data about purchases at a grocery store.

The learning objectives of the program are:

https://powcoder.com

- Write C functions according to specifications.
- Use standard library functions for text file I/O.
- Sorting an array of struct values.

Add WeChat powcoder

- Using struct values and pointers with functions.
- Writing a multi-file C program.

Background

Data mining is the practice of analyzing data to discover patterns and extract information. For examples, retail stores examine the buying patterns of their customers to learn what kinds of items are typically bought together. This information can be used for pricing, product placement, promotions, and so forth.

A particular kind of data mining is to learn *association rules* about purchases. There are a few statistics that are useful in associating the purchase of one item with another in the same transaction. These statistics are:

- Support(X) = fraction of transactions that contain item X
- Confidence(X,Y) = fraction of transactions containing X that also contain Y
- Lift(X,Y) = increase in probability of purchasing Y when X is purchased, compared to the probability of purchasing Y with no knowledge of X. If lift is greater than 1, there is a high association of Y with X -- in other words, if X is purchased, it's more likely that Y is also purchased.

These statistics are mathematically defined below, but the idea for this program is to read a text file with purchase information, and to calculate the statistics for certain items and pairs of items.

The data used for this program was downloaded from Kaggle, a site with numerous data sets and competitions related to data science. https://www.kaggle.com/heeraldedhia/groceries-dataset

Your program must be written to work with any file that matches the format described below. The Kaggle data site has more than 38,000 data items. Your program must work correctly for files that have fewer or more data items.

Main Program

You are provided with a program in **main.c**. It doesn't do much, except make calls to some of the required functions. The primary goal of this assignment is to implement the functions, which will be tested and graded using ZyLab. You are encouraged to implement your own **main** function to test your program during development.

Program Structure

The main program is implemented in **main.c**. This is provided for you. You are also given two header files: **groceryFile.h** and **stats.h**. You must create two source code files that define the functions declared in the header files:

groceryFile.c -- function for reading purchase data from a CSV file.

stats.c -- functions to extract statistics from the file data.

Assignment Project Exam Help

All of these files are compiled and linked together to create a single executable program. See the Appendix for information on how to build a project with multiple files. You should start by creating dummy versions of all the functions, so that you can have a working program as you work incrementally.

https://powcoder.com

Add WeChat powcoder

File Format: CSV

For this program, we use a more-or-less-standard CSV (comma-separated values) format that is a common way to store a spreadsheet in a text-only format. Microsoft Excel can read and write CSV files. Typically, an extension of **.csv** is used in the name of a CSV file.

Details about CSV can be found here: https://en.wikipedia.org/wiki/Comma-separated_values

The idea is that values are stored in text format. A line of text corresponds to one row of a spreadsheet, or one record of related data. Data items on a single row are separated by a comma. Spaces are not ignored -- if there's a space, it is considered to be part of the value. A linefeed character indicates the end of the last data item on a row. Each row should have the same number of data items, but that's probably not enforced. (In our files, it will be true.)

Example: Here are two rows of integer data in CSV format, with four data items in each row.

```
10,100,35,19
-97,62,800,-1234
```

Example: Here are two rows of string data in CSV format, with four data items in each row.

```
fee,fie,foe,fum
tweedle-dee,tweedle-dum,hansel and gretel,Snow White & 7 dwarves
```

Things get a little tricky when the data contains a comma. In that case, the data item uses quotation marks to mark the beginning and end of the item. The quotation marks are NOT part of the data. This program will not have any CSV data with commas, so that part of the format is not relevant.

Grocery File Format

The data for this program has three fields in each row of the CSV file, in the following order:

- Member (integer)
- Date (string)
- Item (string)

The first row of the table contains column header strings. You must read this line and ignore it. Each CSV row represents a ***purchase*** of a single item. Items purchased by the same member on the same date are considered to be part of one ***transaction***.

For this program, you can assume that there will be at most 40,000 rows containing data (not counting the row of header strings). The date string will be no more than 10 characters, following the format described below. The item string may include spaces, and will be no longer than 25 characters.

There are two different formats used for dates. (I didn't do this -- the file came this way!)

dd-mm-yyyy
mm/dd/yyyy

Assignment Project Exam Help

In all cases, "mm" is the number of the month, "dd" is the number of the day, and "yyyy" is the year. The year is always four digits. In the top format, mm and dd are always two digits, but that's not true in the bottom format.

https://powcoder.com

Recommendation: Read the information as a string (or as multiple strings), and use the **atoi** function to convert from a string to a decimal integer. See Appendix.

Add WeChat powcoder

NOTE: Your functions will be tested with files other than **groceries.csv**. There may be other files provided for you for testing, and there may be "mystery" files that we use for tests that are not provided to you. Your code must work with ANY file that follows the format described above. Your code may assume that there are no formatting errors -- the data will match the specified format; there will be no missing data or malformed data.

readFile function

The **readFile** function takes two arguments. The first is a string, which specifies the name of the file. The second is a pointer to a **struct fileInfo** value. Information about the file and its data will be written to the value through this pointer.

Two data structures are defined in **groceryFile.h** that will be needed for this function, and for other functions in the program: **struct purchase** holds information about a single purchase (i.e., a single row in the CSV file), and **struct fileInfo** holds information extracted from the entire file, including an array of struct purchase values. There are `typedef` statements that allow these types to be called **Purchase** and **FileInfo**, respectively.

A Purchase value contains the following fields:

- *transaction id* -- (integer) This is a unique ID that identifies purchases that are part of the same transaction. This is not read directly from the file. Your **readFile** function must identify purchase

by the same member on the same date, and assign each of those purchases the same transaction id.

- *member* -- (integer) A unique identifier for each shopper. This is read directly from the CSV file.
- *date* -- (integer) This is an integer representation of the date of the purchase. We use an integer because it takes up less space and is faster to compare than a string. The encoding of the integer must be: (month × 10^6) + (day × 10^4) + year. This format makes the integer easy for a human to interpret it -- for example, the date 3/6/2020 would result in the integer 3062020.
- *item* -- (char array) The string describing the item will be read directly from the CSV file and stored in this array as a string. (It MUST have a null terminator.) The string may include spaces, but the linefeed at the end of the line is NOT part of the string.

Item strings are *case-insensitive*. In other words, for the purposes of comparing or sorting items, case must be ignored. (Use **strcasecmp**.)

NOTE: It's a bit wasteful to allocate space for the item string in each purchase, since there will likely be multiple copies stored in memory. However, we do this for convenience and to make the programming a little easier. (*Thought experiment* -- How would you redesign this program so that only one copy of each item string is stored?)

A FileInfo value contains the following fields:

Assignment Project Exam Help

- *purchases* -- (integer) The number of purchases read from the file.
- *transactions* -- (integer) The number of unique transactions in the purchase data.

https://powcoder.com

- *members* -- (integer) The number of unique member IDs in the purchase data.
- *items* -- (integer) The number of unique item strings in the purchase data. (Remember: item strings are case-insensitive.)

Add WeChat powcoder

- *data* -- (pointer to Purchase) This field is an array of Purchase values. The array must be dynamically allocated, and its size must be greater than or equal to the value stored in the *purchases* field. Each purchase read from the file must be stored in this array. There is no specified order in which the purchases are stored. (In other words, none of your functions should assume any particular ordering of purchase data.[1])

NOTE: You may preallocate the *data* array according to the maximum number of purchases in a file (40,000). This is the recommended approach, even though it may allocate more memory than needed. Any user of the *data* array will know how many purchases are actually stored by looking at the *purchases* field. (*Thought experiment* -- How would you implement **readFile** to make sure that the data array is exactly the right size?)

NOTE: The data file provided has some duplicates: multiple purchases of the same item on the same day by the same member. Do not remove these purchases. It is allowable for a single item to appear multiple times in a single transaction.

Hint: Implement the readFile function first. This will give you the data you need to implement the statistics functions.

Hint: You may want to sort the data in various orders, to make counting the number of items, assigning IDs to transactions, etc. (For example, sorting by item makes it easier to count the number of unique

[1] Remember: We are writing tests for your functions. The data passed in from our test function may not be the same as the data your functions see in your code. You must not assume anything about the ordering of purchase data in the array.

items.) Use the sorting algorithms discussed in class. Implement these as separate functions within **groceryFile.c**.

stats functions and ItemCount

The **stats.h** file defines one additional data structure type, and also declares several functions that must be implemented.

The **struct itemCount** (also known as **ItemCount**) provides a convenient way to keep track of the number of items for various purposes. It contains an item string (const char *) and a count (integer). The item string is intended to point to a string in a purchase -- this is why it's a pointer, and not an array. There is no reason to keep a separate copy of the string in this data structure.

The functions that must be implemented are:

countItem -- return the number of times an item was purchased.

countPair -- return the number of times two specific items were purchased in the same transaction.

topItems -- produce an sorted array of ItemCount for the items that were most purchased. The array must be sorted from highest count to lowest count, and the number of desired items (and the size of the array) is provided by the caller. The return value is the number of items stored in the array. (This is because the caller might ask for the top 1000 items, when the file only contains 50 items.)

support -- return the support value (double) for a specified item.

confidence -- return the confidence value (double) for a specified item pair.

lift -- return the lift value (double) for a specified item pair.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

For all of these functions, a **const FileInfo** pointer is passed in, which provides the file data. This is a const pointer because these functions will only read the data, and will not change it. (NOTE: The *data* array pointed to by the struct can actually be changed, so you are allowed to reorder the purchases and, in theory, to change the purchase data through this pointer. But you definitely should NOT change the purchase data.)

The mathematical definitions of the last three are as follows:

$$\text{support}(X) = \frac{\text{number of times X is purchased}}{\text{number of transactions}}$$

$$\text{confidence}(X, Y) = \frac{\text{number of times X and Y are purchased in the same transaction}}{\text{number of times X is purchased}}$$

$$\text{lift}(X, Y) = \frac{\text{confidence}(X, Y)}{\text{support}(Y)}$$

If the denominator of any of these equations is zero, the function must return 0. Otherwise, you will get a divide-by-zero error and your program may crash.

Reminder: Dividing integer values uses integer division. To perform floating-point division, you must cast one or both of the integers into floating point values, e.g.: `(double) x`.

Hints and Suggestions

- Don't overcomplicate the program. Do not use anything that we have not covered in class.
- For compiler errors, look at the source code statement mentioned in the error. Try to figure out what the error is telling you. Try to fix the first error first, and then recompile. Sometimes, fixing the first error will make all the other errors go away. (Because the compiler got confused after the first error.)
- Use a source-level debugger to step through the program if the program behavior is not correct. If you are using CLion on your own computer, the debugger is integrated with the editor and compiler, so there's no excuse for not using it.
- For general questions or clarifications, use the Piazza, so that other students can see your question and the answer. For code-specific questions, post a private message to Piazza and attach your code as a file. (Do not copy and paste!)

Administrative Info

Updates or clarifications on Piazza:

Any corrections or clarifications to this program spec will be posted on Piazza. It is important that you read these postings, so that your program will match the updated specification.

Assignment Project Exam Help

What to turn in:

- Submit your **groceryFile.c** and **stats.c** files to the zyBook assignment "Fall 2020 - Program 3 - market".

https://powcoder.com

Grading criteria:

Add WeChat powcoder

10 points: Proper coding style, comments, and headers. No global variables, except as needed and justified by your comments. (There is no good reason to use a global variable for this program.) No goto. See the Programming Assignments section on Moodle for more style guidelines. (You will not get these points if you only submit trivial code.)

30 points: readFile

15 points: countItems

15 points: countPairs

10 points: support (depends on countItems)

5 points: confidence (depends on countItems and countPairs)

5 points: lift (depends on countItems and countPairs)

10 points: topItems

NOTE: Points may be deducted for errors, even if all of the zyBook tests pass. This can happen if your dummy function "accidentally" passes a test, even though it doesn't do anything. It may also happen if it is obvious to the grader that the program is written specifically to pass the specific tests, and would not pass other similar tests.

Appendix A: Setting up the CLion project

1. Create a new project for a C executable with the C11 standard.
2. Download the **main.c** file (overwriting the main.c created by your project).
3. Download all the **.h** files into the same directory.
4. Download the **.csv** files into the same directory.
5. For each additional file (groceryFile.c, stats.c), do the following:
    a. Right-click on the project name.
    b. Select New > C/C++ Source File.
    c. In the menu, set the file name (e.g., stats) and select the .c file. Check the box next to "add to targets". Then click OK.

If you don't add to the target, then you will need to edit the CMakeLists.txt file to include the .c files in the list of files to be compiled for the target executable:

```
add_executable(XXX main.c groceryFile.c stats.c)
```

The XXX can be anything, but is generally the name of your project.

Each file is compiled separately. Each file should at least include the corresponding .h file. For example, in **stats.c**, you will need:

```
#include "stats.h"
```

You may need to include other header files. Include the header files for the functions you need to use -- do NOT copy the declarations of those functions into your file.

You will need to set the Working Directory, so that the program can file the .csv files.

Also, set the C compiler flags to be consistent with the zyBook:

```
set(CMAKE_C_FLAGS "-Wall -Werror")
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Appendix B: Some useful library functions

To read the first row of the .csv file, you can use **fgets** (from `stdio`). Create an array large enough to hold the entire line, and use **fgets** to read the first line into that string.

You can use **fgets** to read the remaining lines and then interpret the data. This may or may not be simpler than using **fscanf** to read each element. It's up to you.

https://en.cppreference.com/w/c/io/fgets

The **atoi** function (from `stdlib`) is also useful. It takes a string parameter, and tries to interpret that string as a decimal integer. This will be helpful when converting the date string to an integer value. And if you use **fgets** to read the entire line, this can be used for the member number.

https://en.cppreference.com/w/c/string/byte/atoi