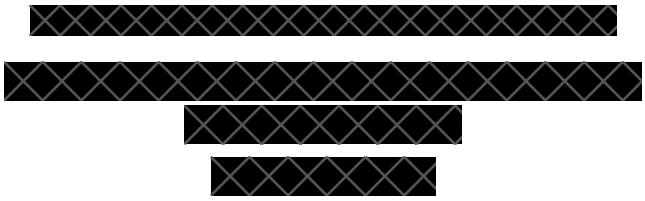

Learning Outcomes

In this assignment you will demonstrate your understanding of arrays, strings, functions, and the `typedef` facility. You must *not* make any use of `malloc()` (Chapter 10) or file operations (Chapter 11) in this project, and should probably also stay away from `struct` types too (Chapter 8).

Files of Numbers

Vast quantities of scientific and engineering data are stored in *comma separated values*-format (CSV) files. In such a file the first line usually describes the columns, and then all the other rows contain numeric data. For example, the first few lines of the test file `data0.txt` consist of:

```
year,month,day,location,mintemp,maxtemp
2020,8,28,18,6.7,12.9
2020,8,28,22,12.7,19.1
2020,8,29,18,7.6,15.3
```

Assignment Project Exam Help

and show three rows of data for temperatures in August 2020, with a code for a location (maybe "18" is Melbourne and "22" is Sydney), a minimum temperature, and a maximum temperature recorded.

https://tutorcs.com

Your task in this assignment is to develop a kind of "CSV tabulator". You will use a two-dimensional C array to store a matrix of numbers, and will write functions that carry out operations on the stored data, including generating reports, graphing it, and sorting it. All of the numbers in all of the input data should be treated as being `double`, even if they do not include decimal points; and two numbers should be regarded as being equal if they differ by less 10^{-6}. Apart from the first row, all data will be strictly numeric.

WeChat: cstutorcs

Before doing anything else, you should copy the skeleton program `ass1-skel.c` and sample data file `data0.txt` from the FAQ page[1], and spend an hour (or two!) to read through the code, understand how it fits together, and check that you can compile it via either `grok` or a terminal shell and `gcc`. Note that if you plan to use `grok`, you will also need to create test files as part of your project, and will need to learn how to execute programs in `grok` via the "terminal" interface that it provides. In other words, now might be a good time to step away from the comfortable environment provided by `grok` and commit to genuine "shell"-mode C programming on your computer.

The skeleton program provides a main program, and two further functions that are somewhat tedious to implement. In particular, the CSV data file is read and processed into internal format (the type `csv_t`) by the function `get_csv_data()`; and the function `get_command()` is provided, together with a controlling loop in the main program, to help you with the interactive input. You do *not* need to understand the way in `get_csv_data()` works, but should be able to by the end of the semester (the relevant techniques are described in Chapter 11). The function `get_command()` should make sense to you by the end of the Week 6 lecture videos. You are to use these two functions and the `main()` function without making any modifications to them.

Once you have `ass1-skel.c` compiled, try this sequence:

```
mac: ./ass1-skel data0.txt
    csv data loaded from data0.txt (12 rows by 6 cols)
```

[1] `http://people.eng.unimelb.edu.au/ammoffat/teaching/10002/ass1/`

```
> i
    column  0: year
    column  1: month
    column  2: day
    column  3: location
    column  4: mintemp
    column  5: maxtemp
> a 4 5
command 'a' is not recognized or not implemented yet
> ^D
Ta daa!!!
mac:
```

Note that `data0.txt` is provided as an *argument* to the program. That file is opened and read as soon as the program commences (magic!) and then the first "`>`" is printed by the program, as a prompt to say "ready for instructions".

The program maintains the numeric CSV data internally in a two-dimensional array `D[][]` of type `csv_t` and buddy variables `dr` and `dc` (the number of active rows and columns respectively), with the row header strings stored in a separate array `H[]` of type `head_t` (and which also uses `dc` as buddy variable). Those are the primary data structures that you need to manipulate in the following stages.

The "`i`" that got typed to the prompt stands for "index", and is a *command*; you can trace what happened though the flow of functions `get_command()`, then `handle_command()`, and then `do_index()`. All commands are single lower-case characters, see `O_IND` and so on. Each command can be followed by a list of integers, specifying column numbers to be selected. If no integers are specified, then all columns are selected, from `0` to `(dr-1)`.

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Stage 1 – Averaging and Displaying (12/20 marks)

Ok, now time for you to add some new commands, starting with `'a'`. Write and incorporate a function `do_analyze()` that takes the standard set of arguments (see `do_index()`) and for each column that is listed in `ccols[]` (buddy variable `nccols`), provides some overall stats about that column of data:

```
> a 1 4
             month (sorted)
    max =      9.0
    min =      8.0
    avg =      8.3
    med =      8.0

           mintemp
    max =    16.1
    min =     6.7
    avg =    10.8
```

If any of the selected are sorted, then that fact is noted, and the median is also computed. Do not report the median if that column is not already sorted. More examples of the required output can be found linked from the FAQ page. Note that throughout this project all CSV-data values are printed as `%7.1f`, with one space in front of them. Output column headings are right-aligned over the numbers they refer to.

The *display* command (`'d'`) processes the rows of the CSV file *in their current ordering*, printing out values from the specified columns, and indicating how many consecutive rows have those values. For example:

```
> d 2 0 1
                    month
            year
     day
    28.0  2020.0     8.0    ( 2 instances)
    29.0  2020.0     8.0    ( 2 instances)
    30.0  2020.0     8.0    ( 2 instances)
    31.0  2020.0     8.0    ( 2 instances)
     1.0  2020.0     9.0    ( 2 instances)
     2.0  2020.0     9.0    ( 2 instances)
```

Note how the columns in the output presentation follow the order of the arguments (and that a column can be shown twice if that is what the user specifies in their command). Note also the way the column headings are layered. There are more output examples available at the FAQ page, illustrating a range of subtleties that you need to make sure are handled by your program.

Stage 2 – Sorting (16/20 marks)

You knew it was coming, well, here it is. The `'s'` command sorts the CSV matrix *according to the specified columns*. That is, the first-listed column is the primary key, with ties in that column broken according to the value in the second column, and so on. In cases where two CSV rows have all of their relevant column values tied, then the ordering that was present in the original array should be retained (that is, the sort should be *stable*).

Assignment Project Exam Help

```
> s 3 0 1 2
    sorted by: location, year, month, day
> d 3 0 1
                    month
            year
location
    18.0  2020.0     8.0    ( 4 instances)
    18.0  2020.0     9.0    ( 2 instances)
    22.0  2020.0     8.0    ( 4 instances)
    22.0  2020.0     9.0    ( 2 instances)
```

https://tutorcs.com

WeChat: cstutorcs

You may (and probably should, so that you can ensure stability) use insertion sort to do this. (I won't tell anyone if you don't tell anyone, ok?)

Stage 3 – Plotting (20/20 marks)

The `'p'` command creates a frequency histogram of all data in the selected columns as a "sideways" bar chart. Ten bands are to be used, computed by dividing the range $[min - 10^{-6}, max + 10^{-6}]$ in to ten equal-width intervals, with *min* the smallest value in any of the selected columns, and *max* the largest value across the selected columns. An integer scaling factor greater than one should be used to ensure that no bar is more than 60 elements wide, with "rounding up" used to determine the number of elements shown in each bar. Examples of input commands and the required output plots are available at the FAQ page.

General Tips...

You will probably find it helpful to include a `DEBUG` mode in your program that prints out intermediate data and variable values. Use `#if (DEBUG)` and `#endif` around such blocks of code, and then `#define DEBUG 1` or `#define DEBUG 0` at the top. Turn off the debug mode when making your final submission, but leave the debug code in place. The FAQ page has more information about this.