Write a suite of loosely connected applications that can be used to create, read, update, and delete a set of csv files. You should be able to use each utility with the other utilities so that you can fully manipulate a spreadsheet document (csv) with just the utilities you have written. Each utility must be capable of manipulating a record based on a specific value contained in a particular row and must be capable of editing more than one row at a time. In addition to displaying particular rows, it must also be able to display or return only particular columns or sets of columns. Each utility program should accept a spreadsheet as input and return a spreadsheet as output. You must also write a menu-based interface to unify each utility into a single program.

To get full credit, you must earn at least 100pts. Note that you need not achieve or implement every milestone for full credit. You can focus on the user interface or the terminal data manipulation utilities exclusively as long as you also write the required documentation in Part C. This strategy would give you full credit. You can also feel free to mix and match from each section. Just be sure to indicate which milestones you implemented when you hand in your project to ensure credit for each feature implementation. Also, make sure your feature implementations add up to or exceed 100pts so that you hand in enough work for full credit.

PART A: The set of command-line utilities:

(15pts) 1. A create utility:
A terminal application that adds records to a csv file.

(15pts) 2. A read utility:
A terminal application that pulls rows from a csv based on search criteria

(15pts) 3. An update utility:
A terminal application that can edit an existing record

(15pts) 4. A delete utility:
A terminal application that can delete an existing record in a csv file.

Bonus:

(5pts) Utilities can parse records on more than just commas (for intstance, tab delimited or semi-colons)

(5pts) Utilities can receive input and output to and from stdin, stdout, literal arguments, and files

(5pts) Utilities adhere to the Unix philosophy (each application does one thing but does it well)

(5pts) Utilities can pipe data to and from each other so that data processing can be chained

(5pts) Write a utility that performs aggregation functions on spreadsheet data and outputs a pivot table

(5pts) Write a utility that can join spreadsheets together on columns with common attributes.

(5pts) Make your utilities function for spreadsheets whose data contain double quotes (“), commas (,), tabs( ), semicolins(;), and all other special characters, meaning that these characters are part of the data fields (content). Make these characters work even if one of these characters is being used as the delimiter. For instance, if the comma is being used to separate values in a row, your program should be able to handle data that also contains commas

(5pts) Make your utilities function with flags in addition to arguments to make your program more versatile

PART B: A menu-based system for interacting with the command-line utilities:

(15pts)1. A menu system where a user can select a file and then choose to add a row, read a row, update a row, or delete a row. This utility does not actually do the creating, reading, updating, or deleting, but calls the utilities built in **PART A** to do the work. Doing the actual spreadsheet is not part of the 15pts for this section.

(15pts)2. The menu selection system is divided up into multiple sections so that the screen refreshes to bring up sub-menus while still remembering the state of your choices from the previous screens until you run a task or quit the program. There is a specific way to exit the program and you can only leave the program using this method (or by overriding the program with a kill command).

(15pts)3. The layout of the screen is beautiful and intuitive. It is not simply a series of textual questions posed to a user but appears visibly as a spatially organized place that communicates possible avenues of navigation, not unlike a well-made web page or GUI application. Feel free to use special characters, colors, and other tricks to achieve this kind of viscerality.

Bonus:

(5pts) The menu system presents as a unified experience such that the user cannot tell they are still in the terminal. For instance, when the screen updates, the terminal screen either does not appear to scroll or actually does not scroll in the traditional way (over-ridden by very technical terminal printing or an added utility whose api you have employed to the task such as tput or ncurses).

(5pts) The layout of the system works well regardless of the size of a user's window or the font size they have chosen to use in their terminal.

(5pts) You can perform multiple spreadsheet manipulations without having to leave your interface. For instance, you can add a row, then modify another row, then delete yet another row, all without exiting the program. You should be able to leave the program and open the spreadsheet in vi or emacs to see the changes you made to the file. Of course, if your project is focusing on the interface, you can write placeholder scripts for the actual spreadsheet manipulation that simply echo "create complete" or "delete complete" to achieve credit for this part.

(5pts) The program should be able to edit multiple different files in one session without exiting the program

PART C: Documentation

(5pts) You must write a man-like file describing how to use your program.

(10pts) Write an additional tutorial with examples that walks a person through how to use your tool.

(5pts) Add your applications permanently to the PATH so that they can be executed just like normal terminal commands. Write a script that will install your programs permanently to another computer's PATH. Be very careful, backup your work, you can make your machine unbootable if this is done incorrectly!