SEng265: Assignment 2
Due: Midnight, Saturday evening, October 11

Overview

The code for handling spreadsheets in CSV format, as at the end of Assignment 1, had some obvious deficiencies. This second assignment requires you to add flexibility to the implementation by removing limits on lengths of input lines and sizes of text items in the spreadsheet cells. Some new operations on spreadsheets will also be added.

Goals

This assignment will provide experience in using dynamically allocated storage and performing text input-output to/from data files.

Changes in the SpreadSheet Data Structure

The `csvfunctions2.h` file specifies a data structure where the number of rows and number of columns are dynamically determined by what is read from an input file (as before). However, the field named `contents` refers to a linked-list. Each item in the linked-list contains a pointer to a vector with one element per column. And each of those elements is a pointer to a `char` array which holds the text value of the spreadsheet cell. Figure 1 shows a diagram of the spreadsheet data structure when there are 4 rows and 3 columns. (Only one sample cell value is shown, and strings are drawn as strings rather than as arrays with `char` elements.)

Figure 1: Spreadsheet Data Structure Example

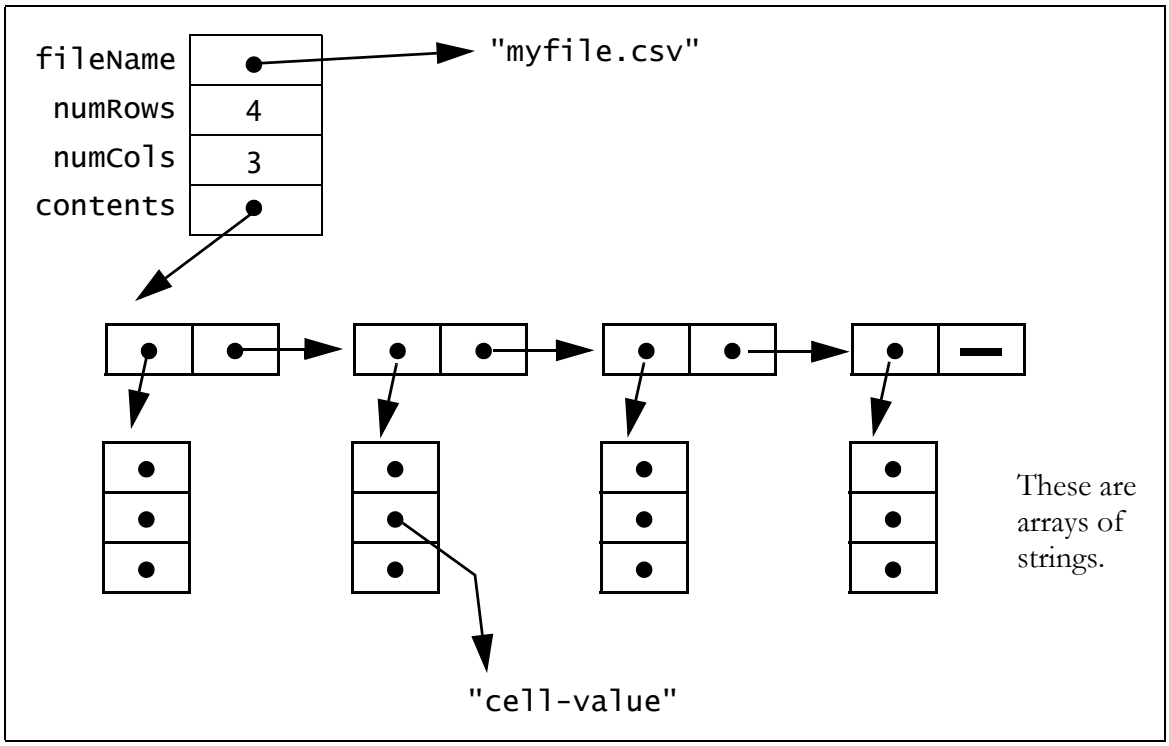

Table 1: Examples of 7 New Subcommands

`load "mydata.csv"`	Read in a spreadsheet from the file named `mydata.csv`. On completion, the number of rows and columns in the spreadsheet are reported. (If another spreadsheet had been previously loaded, that spreadsheet should be unloaded and its storage released.)
`save`	Saves the current spreadsheet back to the hard drive, overwriting the original spreadsheet held there.
`save "otherfile.csv"`	Save the current spreadsheet as a file with a new name (overwriting that file if it already exists).
`merge "moredata.csv"`	Reads in another spreadsheet and appends its rows to the current spreadsheet. (The number of columns in the second spreadsheet should match the number in the current spreadsheet; if they do not match, the current spreadsheet is unchanged and an error reported.)
`stats`	Reports the filename, number of rows and number of columns for the current spreadsheet.
`sort B`	Sorts the rows of the spreadsheet according to the values in column B. String comparison is used (as implemented by the strcmp function), and rows are put into ascending order
`sortnumeric C`	Similar to the above, except that column C should contain entries which can be converted to floating-point, and these floating-point values are sorted numerically.
`deleterow 27`	Deletes row number 27 from the spreadsheet.

The Supplied Files

You are provided with the files listed in Table 2.

Table 2: Supplied Files

`ass2.c`	The driver program. It processes command line options and prompts the user to enter subcommands. All access to the spreadsheet is via the functions implemented in `csvfunctions2.c`.
`Ass1SampleSolution.zip`	Your instructor's solution to Assignment 1. (You are free to use this code to help with completing Assignment 2.)
`csvfunctions2.c`	A starter version with just stubs for most needed functions.
`csvfunctions2.h`	A C header file which specifies the interface between the functions provided in `csvfunctions2.c` and any clients that use the functions, such as `ass2.c`.

Implementation Requirements – The Code You Need to Write

The requirement is that you edit and complete the `csvfunctions2.c` file so that *all the subcommands from Assignment 1 work correctly* and all the new subcommands listed in the table above also work correctly.

For full marks, any command which involves unloading a spreadsheet or deleting spreadsheet rows (i.e. unload, merge and delete) should release any heap storage which is no longer needed. The function `free` in the C library performs exactly that task.

Unlike Assignment 1, you are now free to change the `ass2.c` file if needed to accomplish the assignment or to fix bugs. You are also free to split `csvfunctions2.c` into multiple files (and use any suitable names for the files). However, if you do choose to split the file, then each of the resulting files must contain a `#include` directive for `csvfunctions2.h`. And the `csvfunction2.h` file must NOT be changed. (Your functions *must* use the data structure defined in that header file.)

Tasks 1 through 6 are listed below in the order in which you are advised to attempt them. Task 0 applies throughout. You are advised to commit your files to the Subversion repository whenever you have a new version of `ass2` which compiles and runs some of the subcommands.

Task 0: (Subversion) Weight 10%

Use of Subversion to manage your code and leaving the final result there for grading.

Task 1: (Reimplement Three Existing Commands) Weight 20%

The load and printrow functions have already been reimplemented to use the new data structure. However the findrow, evalsum and evalavg commands also need reimplementing. You are advised to finish these functions one by one, testing after each function has been completed.

Task 2: (Stats) Weight: 5%

The output from the stats command should imitate this example:

```
File: foobar.csv
Rows: 27
Columns: 5
```

Task 3: (Save) Weight: 15%

Implementation of both versions of the save command. Do read the comments in the `csvfunctions2.c` file about how to handle doublequotes.

Task 4: (Sort and SortNumeric) Weight 20%

It is possible but can be quite tricky to sort a linked list directly. If you prefer, you can use the following implementation approach:

- Allocate an array with the same number of elements as the linked list.
- Copy the list elements into the array.
- Use the `qsort` function in the C library to sort the array. (Read the C library documentation for how to pass the array and a comparison function to be used by `qsort`.)

- Copy the elements of the sorted array back into the linked list (overwriting the values that were in the list previously).
- Deallocate the array.

For long lists, use of the `qsort` function in this manner will be more efficient than manipulating the list directly. (It uses the QuickSort algorithm.)

Task 5: (Unload and DeleteRow) Weight 20%

There is no unload command which can be called directly, but there is a `SS_Unload` function which needs to be implemented. If the load command is used twice, the spreadsheet read in by the first use of load should be unloaded. This means that all the heap storage used for the spreadsheet contents should be released by *carefully* making calls to the `free` function. (Making calls in the wrong order will lead to unpredictable results.)

The `SS_ReadCSV` function cannot know in advance how many columns will be found in a spreadsheet row read from a CSV file, and the number of characters in an element's value is unknown too. It is acceptable to use a local `char*` array in the function with a large size (say 128 elements) into which the spreadsheet values are copied, and it is acceptable to use a local `char` array (say 128 characters) into which the characters of one element are copied. Once an entire element has been read, its heap storage can be allocated, and once an entire row has been read, the correct size array for that row can be allocated.[1]

Note that a fairly efficient implementation of unload would work by repeatedly calling `SS_DeleteRow`.

If we detect a memory leak in your program, you will not receive full credit for programming Task 5.

Task 6: (Merge) Weight 10%

A second spreadsheet should be read from the specified file. If it has the same number of columns as the current spreadsheet, the rows of the second spreadsheet should be added as new rows in the current spreadsheet. They should appear at the end, in the same order as they were read in. After the rows have been copied over, any storage which is no longer needed should be released.

If there is a mismatch in the number of columns, an error message should be displayed and all storage for the second spreadsheet should be released.

Running Your Program

If your executable program is named `ass2` then it can be invoked with the command

```
./ass2
```

There is no longer a command-line argument to specify a CSV file. A CSV file should be read only when the load subcommand is entered interactively. Until a file has been successfully loaded, the various commands which operate on a spreadsheet should simply report an error and do nothing.

As before, there is an optional `-d` argument on the command-line to enable debugging output.

1. It is not difficult to write C code so that there are no predefined limits on the number of columns in a row or the number of characters in one spreadsheet value. However, it is not required for this assignment.

Repeat Of An Important Requirement! Your code must compile and run on the Linux machines in the course lab (ELW B215). It is quite possible that the program will run on your home computer or laptop but does not work on the lab machines. You must upload your code to a lab machine and try it out there before you can be sure that you are finished. Do allow extra time for this step.

Submission

If you have not already done so, ensure your Subversion project is checked out from the repository. Within the project create an `assign2` subdirectory. Ensure that this directory and all files you create or edit for this assignment are placed under Subversion control in that subdirectory by using the `svn add` command

Your submission for this assignment will consist of all files in your `assign2` directory in the repository whose names end with the suffix `.c`. We will add the original version of the `csvfunctions2.h` file to your submission and compile using the command

```
gcc -Wall -g *.c -o ass2
```

Be careful that your `assign2` directory in the repository does not contain any extra `.c` files! (Note that it is safe to have extra `.c` files in your Linux directory, just don't use use the `svn add` command on those extra files so that they do not get uploaded to the repository when you perform the `svn commit` operation.)