COMP 348: ASSIGNMENT 1

Important Info:

1. *Feel free to talk to other students about the assignment. That's not a problem. However, when you write the code, you must do this yourself. Source code cannot be shared under any circumstance. This is considered to be plagiarism.*

2. *Assignments must be submitted on time in order to received full value. Appropriate late penalties (< 1 hour = 10%, < 24 hours = 25%) will be applied, as appropriate.*

3. *The graders will be using a the docker command line interpreter to test your code. Make sure that your code is can be compiled and run from the command line.*

DESCRIPTION: In this assignment, you will gain hands-on experience with the C programming language. While you are not required to be a C expert to complete the work, you will certainly have the opportunity to explore most of the things that we have discussed in class (and a few other things as well).

NOTE: This is a long document but the assignment itself is quite easy to describe. Most of the document consists screen images of the running program, plus hints and advice that will make your life a little easier when writing your first C program.

In terms of your task, you will be creating a simple "spreadsheet" application that can be used to calculate and display final grades in COMP 348. Superficially, this is what a program like MS Excel would do. Our application, of course, will be MUCH simpler. It will work as follows.

To begin, you will display a trivial starting menu that will contain just eight basic options. When run, the program will provide the following menu:

```
Spreadsheet Menu
----------------
1. Display Spreadsheet
2. Display Histogram
3. Set sort column
4. Update Last Name
5. Update Exam Grade
6. Update Grade Mapping
7. Delete Student
8. Exit

Selection:
```

Selecting Option 8 will, of course, exit the program and display a message to say "Goodbye and thanks for using our spreadsheet app".

If Options 1 through 7 selected, on the other hand, various actions will be performed.

Before we go any further, note that the screen should always be cleared whenever the menu is (re)displayed. In other words, the text printed by your program should not simply keep scrolling down the screen. Doing this is quite easy – we simply use the `system` function found in `stdlib.h`. Specifically, `system("clear")` will clear the screen on a Linux system (e.g., the docker-based installation that the graders will be using.) On Windows, it's `system("cls")` but this form will fail on docker/Linux.

In terms of the menu options, all will be operating on data read from an input file (i.e., when the app begins). The input file will be located in the same folder as your app and it will be called `students.txt`. As the name implies, it is a simple text file and contains a series of rows, each defining a number of fields. A sample of such a text file is listed below:

```
5640|Blanden|Drona|32|18|14|21|36
5054|Thraves|Lucilia|12|28|31|11|32
5385|Marsay|Gnni|5|28|24|16|26
5184|Bygrove|Lennard|22|21|27|17|31
5459|Close|Alva|25|19|37|18|23
5875|Lockless|Demetra|12|15|16|7|14
5904|Drane|Benjy|39|38|37|24|40
5040|Harty|Conant|33|32|35|23|37
5557|Deetlof|Ada|34|27|34|19|22
5236|Mew|Sylvia|35|16|25|14|24
```

Each line in the file represents one student. The fields in each row are separate by the pipe symbol ('|') and can be interpreted as follows:

<student ID, last name, first name, A1 grade, A2 grade, A3 grade, midterm grade, exam grade>

Note that the maximum grade for each assignment is 40, while the maximum grade for the midterm is 25, and the maximum value for the final exam is 40. The contributions to the final numeric grade will match those of our course – the 3 assignments will be 25% of the final grade, the midterm will be 25% of the final grade, and the final will be 50% of the final grade.

Important: The input data will have NO errors. All fields are valid (correct ranges and no missing entries) and there will be no blank rows. Moreover, there are no spaces between any fields or symbols. A real file might have errors, of course, but it can be quite tedious in C to check all of this. So the data is guaranteed to be clean.

So when the app begins, it will read this data before even displaying the menu. You are free to store the data in memory any way that you like (i.e., data structures of your choice). Just to be

clear, you will not be able to store it as a giant text string, as this will be useless when trying to perform spreadsheet calculations.

Option 1: Okay, so let's select the first option "Display Spreadsheet". The idea here is to provide the user with a display that is vaguely reminiscent of a Excel spreadsheet listing. So, given our sample data above, we would get something like the following:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5040  Harty       Conant      33  32  35  23        37        90.08     A
5054  Thraves     Lucilia     12  28  31  11        32        65.79     C
5184  Bygrove     Lennard     22  21  27  17        31        78.33     B
5236  Mew         Sylvia      35  16  25  14        24        59.83     D
5385  Marsay      Gnni        5   28  24  16        26        60.37     C
5459  Close       Alva        25  19  37  18        23        63.62     C
5557  Deetlof     Ada         34  27  34  19        22        66.28     C
5640  Blanden     Drona       32  18  14  21        36        79.33     B
5875  Lockless    Demetra     12  15  16  7         14        33.45     F
5904  Drane       Benjy       39  38  37  24        40        97.74     A

Press 'c' or 'C' to continue
```

There are a number of things to emphasize here:

1. We obviously have a neatly formatted column-oriented display. While the exact number of spaces between columns is not important, you can see that everything is lined up properly, making the data very easy to read. You must do something similar.
2. All columns have column headers (e.g., "Last", "Exam").
3. The display includes two additional columns: `Total` and `Grade`. These are NOT included in the data file and are calculated dynamically from the input fields.
4. The students are sorted by their student ID number. This is NOT the same order as the rows in the original input file.
5. The `Total` field represents the final numeric grade in the course, based upon (1) the individual graded components and (2) their relative weight, as per the syllabus.
6. The letter grade is based upon the value in the `Total` field. By default, the grade mapping is **A >= 80, B >= 70, C >= 60, D >= 50, F <50**.
7. There is a final prompt below the sheet that allows you to go back to the menu (more on this below). The prompt prevents the menu from immediately re-displaying and gives the user time to actually look at the spreadsheet.

Option 2: Selecting this option will display a "graphical" histogram of the grade distribution. With our sample data – and the default grade mapping - the app would display the following:

```
COMP 348 Grade Distribution

A: **
B: **
C: ****
D: *
F: *


Press 'c' or 'C' to continue
```

Here, we can see that we have something resembling a "bell curve": 2 As, 2 Bs, 4 Cs, 1 D and 1 F.

Option 3: As noted, the default sort order of the table is *ascending* order by student. But that isn't always very useful. So we want to be able to modify the listing to reflect a few other useful sort orders. So when we select Option 3, and provide our input, we will see something like the following:

```
Column Options
--------------
1. Student ID
2. Last name
3. Exam
4. Total

Sort Column: 4

Sort column updated

Press 'c' or 'C' to continue
```

To summarize, the list includes (1) the (default) Student ID, (2) Last name (in *ascending* order), (3) Exam grade (in *descending* order), and the Total numeric grade (in *descending* order). Clearly, `last name` is a lexicographic (alphabetic) sort, while the others are numeric.

If, for example, we select Total as the new sort column, this will be recorded so that the next time we choose Display Spreadsheet from the main menu, we will see the new sort order. With our sample data we would get the following:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     B
5184  Bygrove     Lennard     22  21  27  17        31        70.33     B
5557  Deetlof     Ada         34  27  34  19        22        66.28     C
5054  Thraves     Lucilia     12  28  31  11        32        65.79     C
5459  Close       Alva        25  19  37  18        23        63.62     C
5385  Marsay      Gnni        5   28  24  16        26        60.37     C
5236  Mew         Sylvia      35  16  25  14        24        59.83     D
5875  Lockless    Demetra     12  15  16  7         14        33.45     F

Press 'c' or 'C' to continue
```

It should be obvious that the students have been re-sorted, so that they are now listed in descending order by their final numeric grade in the course.

Option 4: Sometimes we have to make updates. In this case, we want to update/replace the last name for a particular student. So with this option, we would be prompted as follows:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     B
5184  Bygrove     Lennard     22  21  27  17        31        70.33     B
5557  Deetlof     Ada         34  27  34  19        22        66.28     C
5054  Thraves     Lucilia     12  28  31  11        32        65.79     C
5459  Close       Alva        25  19  37  18        23        63.62     C
5385  Marsay      Gnni        5   28  24  16        26        60.37     C
5236  Mew         Sylvia      35  16  25  14        24        59.83     D
5875  Lockless    Demetra     12  15  16  7         14        33.45     F


Enter Student ID: 5236

Enter updated Last Name: Apple
Last name updated

Press 'c' or 'C' to continue
```

Note that before asking the user for the new name, we first display the contents of the sheet. If we don't do this, it will be almost impossible to remember the Student IDs displayed on previous screens.

So in this case we have selected student 5236 (Sylvia Mew) and changed her last name to Apple. When we go back to the menu and display the spreadsheet again, we should see the following:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     B
5184  Bygrove     Lennard     22  21  27  17        31        70.33     B
5557  Deetlof     Ada         34  27  34  19        22        66.28     C
5054  Thraves     Lucilia     12  28  31  11        32        65.79     C
5459  Close       Alva        25  19  37  18        23        63.62     C
5385  Marsay      Gnni        5   28  24  16        26        60.37     C
5236  Apple       Sylvia      35  16  25  14        24        59.83     D
5875  Lockless    Demetra     12  15  16  7         14        33.45     F

Press 'c' or 'C' to continue
```

We can see here that Sylvia Mew is now Sylvia Apple.

Option 5: Here, we make another change. In this case, we update the Exam grade. Our prompt screen will look like this:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     B
5184  Bygrove     Lennard     22  21  27  17        31        70.33     B
5557  Deetlof     Ada         34  27  34  19        22        66.28     C
5054  Thraves     Lucilia     12  28  31  11        32        65.79     C
5459  Close       Alva        25  19  37  18        23        63.62     C
5385  Marsay      Gnni        5   28  24  16        26        60.37     C
5236  Apple       Sylvia      35  16  25  14        24        59.83     D
5875  Lockless    Demetra     12  15  16  7         14        33.45     F


Enter Student ID: 5875

Enter updated exam grade: 39

Exam grade updated

Press 'c' or 'C' to continue
```

In this case, we have decided to update the exam grade for the last student in the list, Demetra Lockless. We have changed her grade from a 14 to a 39.

Let's display the spreadsheet again (with Option 1). We see the image below:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     B
5184  Bygrove     Lennard     22  21  27  17        31        70.33     B
5557  Deetlof     Ada         34  27  34  19        22        66.28     C
5054  Thraves     Lucilia     12  28  31  11        32        65.79     C
5875  Lockless    Demetra     12  15  16  7         39        64.70     C
5459  Close       Alva        25  19  37  18        23        63.62     C
5385  Marsay      Gnni        5   28  24  16        26        60.37     C
5236  Apple       Sylvia      35  16  25  14        24        59.83     D

Press 'c' or 'C' to continue
```

In this case, Demetra has now moved from an F grade to a C. This update will, of course, also change the histogram that we print from Option 2.

Option 6: When creating final grades, we need the option to adjust the grade mappings (i.e., number grade to letter grade). As noted earlier, we have default mapping ranges for the grades. So we want to be able to change those.

The image below shows the basic prompt screen:

```
Current Mapping:
 A: >= 80
 B: >= 70
 C: >= 60
 D: >= 50
 F:  < 50


Enter new A baseline: 70
Enter new B baseline: 50
Enter new C baseline: 40
Enter new D baseline: 30

New Mapping:
 A: >= 70
 B: >= 50
 C: >= 40
 D: >= 30
 F:  < 30

Press 'c' or 'C' to continue
```

The new mapping will be utilized going forward. So when we select Option 1 again, we will now see this:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     A
5184  Bygrove     Lennard     22  21  27  17        31        70.33     A
5557  Deetlof     Ada         34  27  34  19        22        66.28     B
5054  Thraves     Lucilia     12  28  31  11        32        65.79     B
5875  Lockless    Demetra     12  15  16  7         39        64.70     B
5459  Close       Alva        25  19  37  18        23        63.62     B
5385  Marsay      Gnni        5   28  24  16        26        60.37     B
5236  Apple       Sylvia      35  16  25  14        24        59.83     B

Press 'c' or 'C' to continue
```

It should be obvious that this completely changes the set of letter grades for the course. This would also be reflected in the histogram displayed with Option 2.

Option 7: Finally, we have the option to remove a student from the class. The prompt screen for this is straightforward:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5040  Harty       Conant      33  32  35  23        37        90.08     A
5640  Blanden     Drona       32  18  14  21        36        79.33     A
5184  Bygrove     Lennard     22  21  27  17        31        70.33     A
5557  Deetlof     Ada         34  27  34  19        22        66.28     B
5054  Thraves     Lucilia     12  28  31  11        32        65.79     B
5875  Lockless    Demetra     12  15  16  7         39        64.70     B
5459  Close       Alva        25  19  37  18        23        63.62     B
5385  Marsay      Gnni        5   28  24  16        26        60.37     B
5236  Apple       Sylvia      35  16  25  14        24        59.83     B


Enter Student ID: 5040
Student successfully deleted

Press 'c' or 'C' to continue
```

In this case, we are going to remove the second student in the list, Conant Harty. So when we go back to Option 1 and display the class again, we get:

```
COMP 348 GRADE SHEET

ID    Last        First       A1  A2  A3  Midterm   Exam      Total     Grade
--    ----        -----       --  --  --  -------   ----      -----     -----
5904  Drane       Benjy       39  38  37  24        40        97.74     A
5640  Blanden     Drona       32  18  14  21        36        79.33     A
5184  Bygrove     Lennard     22  21  27  17        31        70.33     A
5557  Deetlof     Ada         34  27  34  19        22        66.28     B
5054  Thraves     Lucilia     12  28  31  11        32        65.79     B
5875  Lockless    Demetra     12  15  16  7         39        64.70     B
5459  Close       Alva        25  19  37  18        23        63.62     B
5385  Marsay      Gnni        5   28  24  16        26        60.37     B
5236  Apple       Sylvia      35  16  25  14        24        59.83     B

Press 'c' or 'C' to continue
```

As we can see, our friend Conant Harty is now nowhere to be seen. Just in case it's not obvious, your sorting should continue to work after students are removed.

THINGS TO KEEP IN MIND

If you were writing this application with a language like Python, this would be a very easy assignment, since there is no complex logic and Python has functions to do all of the I/O, sorting, calculations and display in simple, easy to use methods.

C, on the other hand, will make things a little more difficult. Because we want you to focus on the structure of C, and the things that distinguish it from other languages, we do not want you to get lost in obscure issues and common "gotchas". So the following section gives you some advice about what to do and what to avoid. Please read this carefully!

- Start small. Do not try to do everything at once. You might, for example, get a simple menu to display first, then try to read the input file and display the individual fields.

- Compile frequently to see what's working and what is not. Do not continue until you have fixed the current problems (they will not magically go away in the future).

- For info on the functions in the standard C libraries, including clear examples of their use, **cppreference.com** is a very good place to start. Just go to:
  https://en.cppreference.com/w/c

  In particular, follow the *headers* link for the C docs, and then use the links to see the functions in the various libraries (stdio, stdlib, etc). Again, along with most functions, there is sample code that shows exactly how one would use the function in a C program. Feel free to use this code as a starting point – this is NOT plagiarism, it is part of the documentation for the language.

- If you have *segmentation faults* due to pointer problems (and you will), do NOT try to find them with print statements. This is painful and almost never works well. Instead, use the `valgrind` system included with the docker image. An invocation like `valgrind --leak-check=yes my_app_name` will run and analyze your program. It will list problems along with the associated line numbers. To use `valgrind` effectively, you must compile your program with some additional options. On the docker installation, this would look like the following: `gcc -Wall -g -gdwarf-4 file1.c file2.c`

- In the absence of valgrind, a debugger like `gdb` can also help a lot.

- We have kept the GUI as simple as possible. While it's possible to use several mechanisms to read keyboard input, the `scanf` function is probably the simplest since it reads the text and transforms it into the appropriate data type in one step. One non-obvious complication is that when reading the 'c' character to continue processing after the menu actions have been executed, the `scanf` function will actually read the \n (newline) character that remains in the input buffer after the previous `scanf` call. As a result, you will probably want to use code like the following when capturing the c/C character:

```
char proceed;
printf("\nPress 'c' or 'C' to continue ");
do{
      scanf("%c", &proceed);
} while ( (proceed != 'c') && (proceed != 'C'));
```

  This will swallow any *newline* chars and then read the user's `c` character. By the way, you have to hit the Enter key after you press 'c'.

- Reading the input file is fairly straightforward. You will use `fopen` to open the input file(s) in read mode ("r"). You can then use the `fgets` function to read each line as a string (i.e., a `while` loop of `fgets` calls). None of the rows are very long so they can be read into a fixed size char array, something like `char row_buffer[256]`. So a trivial version of the `fgets` loop would look like this:

```
file_ptr = fopen("students.txt", "r");
while (fgets(row_buffer, ROW_LENGTH, file_ptr) != NULL){
      // do something with contents of the row_buffer
}
```

  The one non-obvious part is that `fgets` will include the \n character at the end of the string, which you definitely do not want (e.g., "36\n", instead of "36"). Since you need to convert each row from the char buffer into a set of individual fields, you need to make sure that you do not include the newline character in the final field. There are various ways to do this, but a simple method would be to use a function like `strndup` to create a copy of the original row without the trailing \n.

- You will likely be using pointers in this application. Note that a variable holding a pointer is typically declared like this: `int *foo`. Here, `foo` is a pointer to an int (or possibly an array of ints). So `foo[3]` could be used to access the third int in such an array.

  It is also possible to *dereference* a pointer, which means that we are getting the data that the pointer references. So syntax like `int x = *foo` implies that the int pointed to by `foo` will now be assigned to x. This is a technique that is typically employed in the *comparison* function used when sorting arrays in C.

- In terms of sorting, you will use the `qsort` library function included in the standard libraries. The syntax is sometimes confusing for those not used to C. The basic idea is that `qsort` is a "generic" library function that sorts arrays of elements, based upon a comparison function provided by the programmer. As parameters, `qsort` takes (1) the array to be sorted, (2) the numbers of elements in the array, (3) the size (in bytes) of each element, and (4) the name of the comparison function.

  In turn, the comparison function takes two parameters: `void` pointers to any two array elements that `qsort` will compare during the sort. `qsort` uses "void" pointers since it doesn't know *a priori* what you want to sort. So, typically, the programmer `casts` the pointer to the right type (e.g., void* becomes int*) and then dereferences the pointer to get the element itself. The two elements can then be directly compared for equality.

  A simple `qsort` example can be found on cppreference.com at the following link: https://en.cppreference.com/w/c/algorithm/qsort

- To extract the fields from each input row, you will use the `strtok` function in the standard `string` library. One simply uses a trivial while loop to separate the string on the | delimiter. Again, you can find a `strtok` example on cppreference.com: https://en.cppreference.com/w/c/string/byte/strtok

- When multiple source files are used, header files should be employed in order to provide prototype information for the compiler. Never include variables or function implementations in a header file. NEVER! Header files are primarily used for function prototypes, constants, and things like *struct* definitions.

- Use the #define mechanism for any fixed values (i.e., constants) that should be used throughout the program.

- In terms of dynamically created data, the `malloc` functions are used for this purpose. They always return a pointer/address value.

- When de-allocating memory (i.e., returning memory you no longer need), you will use the `free` function. To be clear, only use `free` with a matching `malloc` call. So a `char` array declared as `char *buffer = (char *)malloc(10)` could later be deallocated

with `free(buffer)`. NEVER try to deallocate a `char` buffer (or any array) defined like this: `char buffer[10]`. malloc was not used to create this and its memory is fully managed by the language environment.

- If a *struct* is defined like `struct foo {int x; inty;};` then we can declare a variable of this type as `struct foo bar`. We can now use `bar.x` or `bar.y` to access the internal values. But if we use `malloc` to dynamically create space for a new `struct foo` value, like `struct foo *bar = (struct foo *)malloc(sizeof(struct foo))`, then we must use the syntax `bar->x` or `bar->y` to access the value since we are now using a pointer to the foo struct.

- Using pointers is crucial in C but it can be confusing at first. A couple of things to keep in mind: If we use `malloc` to create an array (e.g., of ints or chars), we can just use the standard array syntax (e.g., `foo[4]`) to access the elements of the array. The C compiler already knows that the array variable is a pointer. It is also possible to have pointers to pointers. So if we use `malloc` to create an array of strings called `foo`, this would actually consist of a pointer to the main array, which would then contain pointers for the strings. So `foo` would be defined as `char **foo (or char* *foo)`. In other words, `foo` is pointer to an array of string pointers. This can be confusing at first and can lead to lots of unexpected segmentation faults if you are referencing the wrong pointer. Moreover, if you are deallocating this memory, you have to be REALLY careful. Specifically, the strings must be deallocated first, and then the main array. And, just to be clear, free'ing the main array does not automatically free the memory used by the strings.

EVALUATION: Please note that the markers will be using a standard Linux system, likely the same docker installation that most of you should be using. Your code MUST compile and run from the Linux command line. If you are using an IDE (e.g., Eclipse or VS Code), rather than a simple editor, you will want to test execution from the command line, since IDE's often create their own paths, folders, environment variables that might prevent the code from running on the command line.

To evaluate the submissions, the marker will simply create a test folder and add a sample input data file plus your source files. Your code will be compiled and run and each of the menu options will be tested. Every student will be graded the same way. Note that you are expected to deallocate any memory that you have allocated during the running of the application. Point value will be deducted if you have memory leaks.

The markers will be given a simple spreadsheet that lists the various criteria described above. There are no mystery requirements. Please note that it is better to have a working version of a slightly restricted program than a non-working version of something that tried to do everything (e.g., you might be able to display the spreadsheet but you can't sort it, or you can display the existing students but you can't delete any). So make sure that your code actually compiles and runs. It is virtually impossible to grade an assignment that does not run at all.

DELIVERABLES: Your submission should have multiple source files (if you can't get that to work, one large source file will still be graded, just at a reduced value). Ideally, you would have the following source files: `spreadsheet.c` (the `main` function and the basic GUI), `ordering.c` (any code related to sorting the spreadsheet), `data.c` (any code used to read the input file and extract the fields from each row), and `calc.c` (the code used to carry out the logic of the menu options – this will definitely be the biggest file). Each of these may have an associated header file of the same name (with a .h extension). The header files must list one or more publicly accessible functions. These .c and .h files represent the only code that you will submit.

IMPORTANT 1: As noted, no error checking is required on the input data. However, very basic error checking is required for the user entries. For example, if you are updating a name and you use a student ID that doesn't exist, the program should not crash. As a second example, you should not be able to update an exam grade with a value of 456 or -23. This makes no sense.

IMPORTANT 2: All files (.c and .h) must be prepared/saved in the same folder so that they can be compiled from the command line using the following simple gcc command:

```
gcc spreadsheet.c data.c ordering.c calc.c
```

You can add the `-Wall -g -gdwarf-4` options during development so that you can test and profile your code.

Note that you may include a README file if some of the application's functionality is not complete. This will allow the grader to give you value for the parts that are working, instead of assuming that nothing is working properly.

Once you are ready to submit, compress all .c/.h/README files (and ONLY these files) into a zip file. The name of the zip file will consist of "a1" + last name + first name + student ID + ".zip", using the underscore character "_" as the separator. For example, if your name is John Smith and your ID is "123456", then your zip file would be combined into a file called a1_Smith_John_123456.zip". The final zip file will be submitted through the course web site on Moodle. You simply upload the file using the link on the assignment web page.

Please note that it is your responsibility to check that the proper files have been uploaded. No additional or updated files will be accepted after the deadlines. You can not say that you accidentally submitted an "early version" to Moodle. You are graded only on what you upload.

Good Luck