COMP 2401 B/D

Assignment #1: Ghost Hunter Data Entry

Getting Started with C

Goals:

In this assignment you will be writing a program in C using the Ubuntu Linux environment which is run on the course virtual machine (VM). You will get started with arrays and basic code elements in C.

Learning Outcomes:

If you are keeping up with the lectures, completing tutorials, and read the course notes as needed, then by the end of this assignment you will have demonstrated that you can:

- Interact with the course VM running the Linux operating system
- Write a small program in C that is designed with modular functions and documented correctly
- Utilize basic user input and output in C programs
- Perform basic manipulation of primitive arrays in C
- Write functions which communicate by passing parameters and using return values

Assignment Context (Optional Read):

You took the job! Excellent. Welcome to the Carleton University Ghost Hunters Society!

I know our systems are a bit dated, but that's why we've hired you for these next few contracts. You see, we go into houses, farm houses, schools, prisons, and campsites, and we use our array of sensors to detect whether ghosts are in the rooms. By combining sensor data, we can determine what kinds of ghosts and ghouls are infesting the locations. Over the next few months, you will be in charge of developing the software for our core detection and analysis systems!

What? I've never even *heard* of the video game *Phasmaphobia* - this is **CUGHS**!

For your first job: We want to test your skills. We are using something called an *EMF Reader* - it reads electromagnetic fields from things like lights, cameras, wiring in the walls, and ghostly entities. It gives each room of the building a unique identifier (a UUID), and reports back a reading between 0.0 and 5.0 for each room. At the end, the device has a list of UUIDs and EMF readings. Right now, if we survive the ghost hunt, we scroll through the device and write the numbers out by hand. Can you help us do better? Maybe instead of writing it down, **we can type it!** And maybe even **sort** it?

Instructions:

In this assignment, you will be creating a simple command line interface in C which prompts the user to enter pairwise data, sorting that data, and printing it with some nice formatting. The user will be entering a UUID (an 8 digit number) and an EMF value (a floating point number between 0 and 5) on the same line (eg. "32850021 4.2214"). The data will be sorted in descending (decreasing) order.

Pay close attention to the constraints and requirements of this specification, as failure to follow the constraints can lead to **large** penalties.

Provided Code:

You have been provided with some code, *a1-posted.c* which contains:

- Definitions for important error codes and the maximum array size
- Forward declarations for all of the functions you will need
    - **Remember:** Forward declarations tell C, "There will be a function with this signature". By forward declaring, we can order our actual function definitions any way we'd like. In the future, we will learn about forward declaring in separate files from our main code!
- An empty main() function, which you can use for testing and eventually the main control flow

Approach for Completing:

This specification will cover each section more carefully, but here is the overall plan for approaching this assignment:

1. Begin by writing empty functions for all of the forward declarations, and make sure the program runs and compiles. Document each function appropriately.
2. Set up your main() function to define the variables we need for functions (our arrays and counters).
3. Consider setting up some test code in your main() function that lets you run each function you write and make sure it is working correctly. **Note: ALL TEST CODE MUST BE REMOVED before submitting.** If you separate your tests into one or multiple functions and call *these* test functions in main, it will save some cleanup at the end.
4. Write your "`validateUUID`" and "`validateEMF`" functions, and test them out in main with a few values to make sure they're working.
5. Write your "`getEmfData`" function, and make sure it loops and terminates correctly. You can print the return value (number of entries) to test this.
6. Write your "`printEmfData`" function, and visually inspect that all of your data is printing out correctly. Consider testing your error checking here by sending invalid values.
7. Write your sorting function, making sure that it follows the algorithm described in this specification. Test it with some different values to make sure that it works as intended.
8. Clean up the code and verify that it matches the assignment requirements.
9. Write your README file, package everything in a tar or zip file, submit, download, and compile & run on the course VM *one more time* to make sure everything is working.

Carefully read the following instructions for each major component of the assignment. Make sure to follow the constraints detailed here as well as in the more detailed constraints section at the end of the specification. Your program must follow the conventions that we have covered in the course.

Programs should be documented appropriately, code should be separated into modular functions, perform basic error checking with nice error messages, and reuse modular functions where possible.

1. **The Main Function (Part 1)**
   - 1.1. Review the base code. **Do not make any unspecified changes**, but specifically pay attention to not change function prototypes or definition values.
   - 1.2. Define two arrays: One for the EMF Data and one for the corresponding UUIDs. The two arrays represent two parts of the same data; eg. `emfData[3]` contains the data for the room defined by `uuidData[3]`. **They should always remain synchronized.**
2. **Data Inputs and Validations**
   - 2.1. Implement the `int validateEMF(float emf)` function
     - 2.1.1. The function takes in an EMF value (emf) and checks that it is within the appropriate range (between 0.0 and 5.0 inclusively with arbitrary precision)
     - 2.1.2. The function returns either a success flag (from the provided code definitions) OR returns the corresponding error flag, using the predefined constants.
   - 2.2. Implement the `int validateUUID(int uuid)` function
     - 2.2.1. The function takes in a UUID value (uuid) and checks that it is within the appropriate range (**between 32850000 and 32859999 inclusively**)
     - 2.2.2. The function returns either a success flag OR returns the correct error flag
   - 2.3. Implement the `int getEmfData(int *uuid, float *emf)` input function which takes data pairs in on a loop until the user enters `-1`. You **can not** ask the user how many data pairs they will enter in advance.
     - 2.3.1. Prompt the user for a UUID and EMF value pair, separated by a space
     - 2.3.2. Continue prompting for pairs until the `-1` value is submitted
     - **2.3.3. This is C:** Keep track of how many valid data pairs have been inputted
     - 2.3.4. Each time a prompt is submitted, try to store the data pairs in the arrays you created in main; however, before adding, make sure:
       - 2.3.4.1. The arrays still have space; if they are full, return the appropriate error flag
       - 2.3.4.2. The inputs are valid EMF and UUID values, using the validation functions that you wrote earlier. If either input fails, return the appropriate error flag
     - 2.3.5. Finally, if everything was successful, **return the number of data pairs that were entered by the user**; this should represent the number of elements currently contained in our arrays.

Comments: As mentioned in class, arrays in C are *actually pointers* to the first element in the array. That's why we can make changes to them, and why the function parameters in `getEmfData` are pointers. Refresh yourself by reviewing our in-class examples or Section 1.6 of the course notes.

3. Printing Functionality

3.1. Implement the `void printEmfData(int *uuid, float *emf, int num)` function which takes in three input parameters: the array of Room UUIDs (*`uuid`), the array of EMF data (*`emf`), and the number of entries in the arrays (which should match, as their data should be aligned with each other).

3.1.1. Each Room UUID and EMF pairing must be printed on a single line

3.1.2. Each field of the data pairing must be aligned as columns

3.1.3. The EMF value must be printed as one digit followed by a decimal point, followed by only **one** digit after the decimal point, using the format options of `printf()`

3.1.4. After all pairs are printed out, print the total number of pairs.

3.2. Make sure that you only print the data in the arrays (eg. if there are 5 entries, your loop should only print 5 items).

4. Implement Data Sorting

4.1. First, implement the helper function `int findMaxIndex(float *arr, int num)` which takes in an array (`arr`), and the number of valid elements in that list (`num`).

4.1.1. Return the index of the highest value in the array, or an appropriate error flag

4.2. Next, implement the `int orderEmfData(int *uuid, float *emf, int num)` function which takes in an array of Room UUIDs, an array of EMF data, and the number of valid entries.

4.2.1. This function should reorder both arrays so that they are sorted by the EMF data. Remember: The arrays represent the same data pairs (a room ID and sensor data from that room) and the UUID and EMF data for that room should always share an index.

4.2.2. The data should be sorted in descending (decreasing) order (the rooms with the highest EMF reading are at the front, the lowest readings are at the end)

4.2.3. The function **can not** use any kind of built-in sorting functions

4.2.4. Use the following algorithm:

4.2.4.1. Create a **deep copy** of each array; that is, do not simply copy the array variable, but create a new array, loop over it, and add each element one-by-one into the new array. **Why?** A **shallow** copy would only copy the address to the first element! Making changes would affect the original array. Call them anything you like, but let's assume *localEMF, localUUID*

4.2.4.2. Using a loop, at each index:

4.2.4.2.1. Find the current highest EMF value in the local array

4.2.4.2.2. Update the in/out arrays so that the current highest value is at the current index we're looping over (i.e. the correct position)

4.2.4.2.3. Override the highest value in the copied EMF array with -1, so that it isn't considered again on the next iteration

4.2.5. Return a success flag if the operation was successful, or an error flag otherwise

5. **Main Function (Part 2)**
   5.1. Now it's time to put it all together to test the overall control flow.
      5.1.1. Declare two arrays as local variables in the main() functions which will be populated with EMF sensor data and Room UUIDs, with a maximum size defined by the definitions at the top of the provided code
      5.1.2. Read data pairs in from the user and store them in the local arrays
      5.1.3. Print the data pairs to the screen
      5.1.4. Reorder the data pairs
      5.1.5. Print the data pairs to verify that the data is sorted and still synchronized
   5.2. **The main function is where friendly and useful error messages are displayed**. The other functions return error codes, they do not print error messages.

Requirements:

Make sure that you follow these requirements for the whole program to make sure that you get full marks.

1. Document your program as demonstrated in class. A formatting guide *may* be posted to Brightspace to assist with this, and you will be expected to follow this if it is. Minimally but not all encompassing, your documentation should include:
   a. A description of the purpose and process of the function
   b. The role for each parameter (in, out, in/out)
   c. Possible return values
2. Include a README file; a plaintext file named README or README.txt or README.md which contains the following information:
   a. A preamble: The author, student ID, description of the program, and a list of files
   b. Instructions for compilation and launching, including command line arguments
3. Submit all files in a single .tar or .zip file
4. Ensure the code compiles and executes on the course VM following the directions that you provide in the README
5. Functions will only receive full marks if they are implemented, called in the program, and not commented out (this will be assumed to be rough, discarded work)
6. All data should be printed to the screen as required to demonstrate functioning code
7. The code should follow good coding practices:
   a. Functions are modular, reusable, and reused where possible
   b. There are no global variables
   c. Constants (defines) are used where applicable
   d. Return values indicate error state, unless otherwise specified
   e. Helper functions are used when it's consistent with the program design
   f. Functions are documented

Grading:

While specifics of the grading scheme may change, the following is an approximation of the weights of each section:

- 15 marks: correct design and implementation of the main() function
- 6 marks: correct design and implementation of the data validation functions
- 25 marks: correct design and implementation of the data input function
- 8 marks: correct design and implementation of the data printing function
- 8 marks: correct design and implementation of the max index helper function
- 20 marks: correct design and implementation of the reorder function
- 4 marks: correct packaging (file names, README, etc.)
- 4 marks: correct documentation (in the code)
- 10 marks: correct execution (runs and compiles correctly)

Please recall that all course materials are copyright and you may not share or distribute the materials to anyone outside of the class for any purpose.

Collaboration of any kind outside of general material discussion is strictly prohibited.

Late submissions will not be accepted, unless an announcement is made which overrides this policy.

Submit early, submit often. Technical issues at the deadline do not warrant an extension. You are expected to submit your work periodically to ensure at least part marks for work done prior to the deadlines.

For support or clarifications, attend TA office hours or post to the course Brightspace discussion forums.

*You have **one week** after you receive your grades back to contact the TA that graded your assignment to dispute your grades, after which it will not be considered. Disputes are only valid if the TA clearly graded incorrectly; disagreeing with the grading scheme is not a valid dispute.*