

Lab 4 Specification – Exploring Structs in conjunction with Arrays and Pointers in C
Due (via your git repo) no later than 8 AM, Thursday, 6th September 2022.
50 points

Lab Goals

- Exploring the implementation of Structs in conjunction with Arrays and Pointers.
- Reflect on low end computing including the usage of Pointers.

Learning Assignment

If not done previously, it is strongly recommended to read all of the relevant "GitHub Guides", available at the following website:

<https://guides.github.com/>

that explains how to use many of the features that GitHub provides. This reading assignment is useful to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, it is also recommended to do the reading assignment from the section of the course textbook outlined below:

- **KR chapter 5 - [5.6 - 5.9] and chapter 06 - [6.1 - 6.4], PH Chapter 1.6**

Assignment Details

Now that we have discussed some more of C Programming Concepts, we are ready to explore and do a few more exercises to think logic, use low end computing tools such as Structs and Arrays/Pointers, and implement it in C programs. We will also get an opportunity to implement these in a practical use case, that is in a simple project level requirement that was discussed in class.

At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL(s) to clarify if there is any confusion related to the lab and/or class materials. The Professor proofread the document more than once, if there is an error in the document, it will be much appreciated if you can communicate that to the Professor. The class will be then informed as soon as possible regarding the error in the document. Additionally, it is highly recommended that students will reach out to the Professor in advance of the lab submission with any questions. Waiting till the last minute will minimize the student's chances to get proper assistance from the Professor and the Technical Leader(s).

Students are recommended to get started with this part in the laboratory session, by discussing ideas and clarifying with the Professor and the Technical Leader(s). It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other's program(s) and technical reports is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as Quizzes, exams, etc . . .

Preliminary Steps



It is important that you can set up Docker and GitHub to complete the rest of the lab. Please follow the guidelines below to complete the preliminary steps.

1. **[Docker Setup.]** At this point, I expect the MAC, Linux, and Windows Pro users, to have this step completed based on our previous class discussions. For those who had not completed this step, the documentation below should provide more details regarding the download and installation setup.

- Get Docker setup completed on your laptops:

- Docker Mac Setup:

<https://docs.docker.com/docker-for-mac/install/>

- Docker Ubuntu Setup

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04>

- Docker Windows Setup:

<https://docs.docker.com/docker-for-windows/install/>

- If the setup goes correctly as desired, you should be able to get started and validate the Docker version and run the hello world docker container using the following commands:

```
docker -version
```

```
docker run hello-world
```

- There are some more documentation for Docker get started to test your installation in the link provided below:

<https://docs.docker.com/docker-for-mac/>

<https://docs.docker.com/docker-for-windows/>

2. **[Loading Docker Container.]** There are two steps in loading the container, namely:

- Build the container
- Connect and Run the container

Build the container: So to build the container. the following steps should be performed.

(a) First, accept the lab URL provided in Slack. After downloading the lab folder from the GitHub classroom, navigate to the `cmpsc200-fall-22-lab4` directory using terminal (Mac/Ubuntu) or Command Prompt/Docker quick start terminal (windows). It is recommended to use Command Prompt to get better results.

(b) Build the docker image using the following command:

```
docker build -t cs200lab4 .
```

Please note: There is a dot towards the end of the command. It is required to include the dot at the end of the command.

(c) Note: In the command above, `cs200lab4` is the user-provided image name. This could be random. But it is recommended to use the same name to easily follow the rest of this document. Additionally, it is required to be inside the `cmpsc200-fall-22-lab4` directory to run the build command. If you are not inside the `cmpsc200-fall-22-lab4` directory, you may receive an error message.

(d) Upon successful build, it is recommended to verify the correctness of image creation by using the following command:

```
docker image ls
```

(e) The image named "`cs200lab4`" should be listed as one of the outputs from the command above.

Connect and Run the container: So to create and run the container. the following steps should be performed.

(a) Run the docker container based on the image created in the previous steps using the following command:

Mac/Ubuntu:

```
docker run --rm -v $(pwd)/src:/root -it cs200lab4
```

Windows:

```
docker run --rm -v "%cd%/src":/root -it cs200lab4
```

(b) To run the above command, it is required to be inside the `cmpsc200-fall-22-lab4` directory. And, please note, you will log in to the container after entering the above command.

(c) **Please note:** If you are windows user, you will get better results by running this command using the Command Prompt instead of Powershell.

(d) After creating the container, the run command above creates a mount between the host machine and the container with a shared folder space. So, any files placed inside the host mount directory can be easily accessible inside the container mount directory and vice versa.

(e) After connecting to the container, we can compile C Programs using the command below within the container:

```
gcc hello.c -o hello.out
```

(f) After compiler the program, we can execute C Programs using the command within the container:

```
./hello.out
```

(g) **[GitHub Setup.]** Take a look at the detailed documentation for getting started with GitHub, which is available at: <https://docs.github.com/en/get-started>

You are required to know the procedure to git clone, git pull, git add, git commit, and git push to access the lab specification folder and to submit your lab for grading purposes. If there is an issue with your GitHub setup please discuss it with your Technical Leader(s) and/or the Professor.

Section 1: Reflection on Pointers



VectorStock® VectorStock.com/27232034

This section is worth 20 points. Each task is 10 points.

In computer science, a pointer is an object in many programming languages that stores a memory address. This is a value located in computer memory, or in some cases, that of memory-mapped computer hardware. As an analogy, a page number in a book's index could be considered a pointer to the corresponding page; a pointer is then used by flipping to the page with the given page number and reading the text found on that page. The actual format and content of a pointer variable is dependent on the underlying computer architecture. (text adapted from wiki).

Professor David Brailsford is one of the most famous and oldest computer scientists who had seen the evolution of computers from olden programming languages to modern programming languages. In this section, we will watch two videos and reflect on the points discussed in the clips. This reflection is instrumental to further advance our understanding of C programming, and in general to foster the learning from our recent discussions on Performance Assessment and understand Why C is so influential from the man who is so popular in the CS community. To complete this part, it is required to do the following:

Task 1 → Overview

Watch this 10-minute video, by using the link below:

<https://www.youtube.com/watch?v=cilPJexnfNE>

After watching the video, edit the video-reflection-1 file. In this file, provide a detailed description of the questions presented below:

Questions:

1. Why is C so powerful? and what features are provided in the language to help provide better performance according to the speaker?
2. What is Moore's law and Why did the speaker mention this popular (law in the field of computer science) in this video?
3. What is the popular phrase "If ain't broken, then don't fix it" has to do with this video?
4. What is an assembler and an assembly language? (Research into this online), and what is the speaker's view on high-level languages compared to Assemblers? Note: We will discuss in detail about assembly language in this course.
5. What did the speaker say about Java? and what programming features were discussed as an example, to compare Java with C?

Task 2 → Overview

Watch this 10-minute video, by using the link below:

https://www.youtube.com/watch?v=h-HBipu_1P0

After watching the video, edit the video-reflection-2 file. In this file, provide a detailed description of the questions presented below:

Questions:

1. What is a pointer?
2. What is dereferencing? How is it related to indirection discussed in class?
3. How is a pointer variable different from a regular variable?
4. The video discussed the byte allocation of int, char, and float data types. How much is the memory allocated for (int, char, and float) in terms of bits respectively?
5. How do computers differentiate memory address from the actual value stored in a variable? For example, explain what happens when we execute the following line

`int alpha = 100;` in a C program?

Section 2: Structs Program

This section is worth 30 points. Each task is 15 points.

We discussed Struct implementation in C, during our last two classes. Please refer to the Week4 slides and source code. We defined a struct, declared a variable of the struct type and initialized the members of the struct, and processed the struct. We did this for both a single struct element and an array of struct elements. The goal of this program is to create two arrays of structs for all the patients in floor 1 and 2 respectively. After creating this array of structs, the program should be able to swap the patients in the two floors by using a separate method. To complete this section, implement the `patient.c` program in the starter code by adding code to handle the following requirements. The starter code is incomplete and we need you to complete the code by carefully implementing the requirements listed below:

Task 3 → Patients Database Program Overview

- **R1:** Define a struct named **patient** with six members, that is, Patient ID (**int**), Patient Full Name (**char ***), Weight (**float**), Height (**float**), Disease (**char ***), and Floor (**int**). This part is completed.
- **R2:** Declare two struct arrays of type **patient** with the name floor1 and floor2 respectively of size 5 each. We may assume that the total number of patients in both floors combined is 10. Refer students.c in Week4 folder and products.c in Week5 folder for creating an array of structs. This part is completed.
- **R3:** Next assign the values for each element in the struct array with the corresponding patient details provided in the patient table given above. This declaration should be done in the main method. You may load the struct members directly by hard coding the values in the code or by creating a separate file to store the data, and generating the array of structs by reading from that file. This part is open ended and you may select an approach that works well for you. Read through the comments to know how to do this hardcoded. This part is incomplete. You need to complete this to make the program functional.

Patient ID	Patient Full Name	Weight	Height	Disease	Floor
10011001	Alex Crowe	210.5	5.6	Tuberculosis	1
10011002	Amelia Kaur	161.6	6.2	Asthma	1
10011003	Amanda Daya	171.3	5.8	Heart Attack	1
10011004	Ben Krish	151.3	5.3	Stroke	1
10011005	Brian Miller	181.5	5.9	Urinary Tract Infection	1
10011006	Chris Miller	251.3	6.4	Asthma	2
10011007	Drew Millan	178.4	5.7	Hypertension	2
10011008	Frank Derrick	191.9	6.0	Sarcoidosis	2
10011009	Robin Meade	271.3	6.8	Lung Cancer	2
10011010	Rosy David	131.3	5.3	Diabetes	2

Table 1: Patient Database Table

- **R4:** Add your implementation to the **Swap** method. The first parameter is a struct array named floor1, and the second parameter a struct array named floor2. Add your logic to the method for swapping the patients in floor1 with floor2 and vice versa. Please note, the changes are simply the modified content. Refer the comments in this method for more information.
- **R5:** The **Display** method is completed. The method iterate and display the contents of the struct arrays. Please note, the caller, that is main method, call this method to display both the initial and modified values of the two structs.

Validation:

Due to space limitations the output is shown in two screenshots below:

A sample screenshot (see next page)

```

amohan@amohanmacpro src % gcc patients.c -o patients.out
amohan@amohanmacpro src % ./patients.out
ID      Full Name      Weight Height Disease      Floor
Floor 1:
-----
10011001    Alex Crowe      210.5   5.6   Tuberculosis    1
10011002    Amelia Kaur     161.6   6.2   Asthma          1
10011003    Amanda Daya     171.3   5.8   Heart Attack    1
10011004    Ben Krish       151.3   5.3   Stroke          1
10011005    Brian Miller    181.5   5.9   Urinary Tract Infection 1
-----
Floor 2:
-----
10011006    Chris Miller    251.3   6.4   Asthma          2
10011007    Drew Millan     178.4   5.7   Hypertension    2
10011008    Frank Derrick   191.9   6.0   Sarcoidosis     2
10011009    Robin Meade     271.3   6.8   Lung Cancer     2
10011010    Rosy David      131.3   5.3   Diabetes        2
-----
ID      Full Name      Weight Height Disease      Floor
Floor 1:
-----
10011006    Chris Miller    251.3   6.4   Asthma          1
10011007    Drew Millan     178.4   5.7   Hypertension    1
10011008    Frank Derrick   191.9   6.0   Sarcoidosis     1
10011009    Robin Meade     271.3   6.8   Lung Cancer     1
10011010    Rosy David      131.3   5.3   Diabetes        1
-----
Floor 2:
-----
10011001    Alex Crowe      210.5   5.6   Tuberculosis    2
10011002    Amelia Kaur     161.6   6.2   Asthma          2
10011003    Amanda Daya     171.3   5.8   Heart Attack    2
10011004    Ben Krish       151.3   5.3   Stroke          2
10011005    Brian Miller    181.5   5.9   Urinary Tract Infection 2
-----

```

Task 4 → Songs Statistics Program Overview

A program named `songs.c` is provided in the starter code. This program includes the implementation to generate a text file using randomized data points generated for each song. Each song, has a number (ID), count (number of times heard), rank (a number between 0 to 5), and duration (a number between 0 to 6.5). These randomized data points are generated and written to a file named `data.txt` file. Next the program initializes a struct array and store the data points into the array by segregating the members of the struct. The program requires some additional implementation to be complete. That is, a new set of logic should be added to the `report` method to display some statistics on the songs dataset. Please make sure to complete the following requirements:

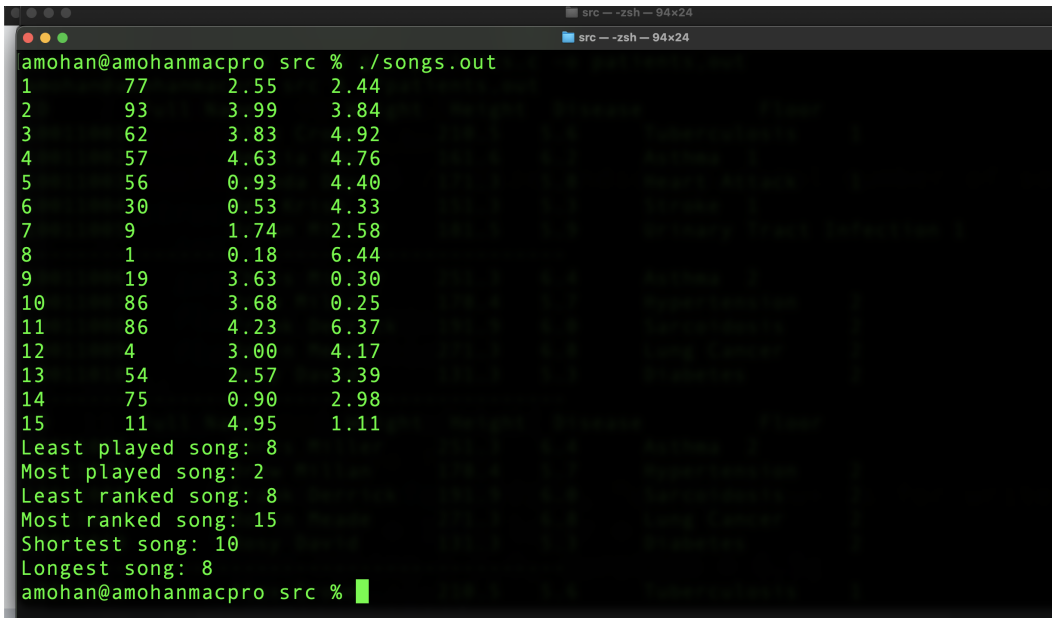
- **R1:** Complete the implementation in the `report` method to answer the top 6 questions on the dataset. Add the required logic to figure out the answers for these 6 questions. Once you added the logic, and bind the results to the corresponding variables, then the output will be displayed by the starter code. For more information on the variable bindings, refer to the comments in the source code. Also, please refer to your previous labs and lecture notes, for finding the minimum and maximum element in an array. The 6 questions are provided below for your reference:

1. Identify the least played song.
2. Identify the most played song
3. Identify the highly ranked song
4. Identify the least ranked song

5. Identify the shortest song
6. Identify the longest song

Validation:

A sample screenshot



```
amohan@amohanmacpro src % ./songs.out
1      77      2.55      2.44
2      93      3.99      3.84
3      62      3.83      4.92
4      57      4.63      4.76
5      56      0.93      4.40
6      30      0.53      4.33
7      9       1.74      2.58
8      1       0.18      6.44
9      19      3.63      0.30
10     86      3.68      0.25
11     86      4.23      6.37
12     4       3.00      4.17
13     54      2.57      3.39
14     75      0.90      2.98
15     11      4.95      1.11
Least played song: 8
Most played song: 2
Least ranked song: 8
Most ranked song: 15
Shortest song: 10
Longest song: 8
amohan@amohanmacpro src %
```

Section 3 - Honor Code

Make sure to **Sign** the following statement in the `honor-code.txt` file in your repository. To sign your name, simply replace Student Name with your name. The lab work will not be graded unless the honor code file is signed by you.

This work is mine unless otherwise cited - Student Name

Section 4 - Reflection

Add a Reflection to the repository by modifying the `reflection` file in the lab repository. List out the biggest learning points and any challenges that you have encountered during this lab.

Submission Details

For this assignment, please submit the following to your GitHub lab repository.

1. updated, completed version of **songs.c** file.
2. updated, completed version of **patients.c** file.
3. updated, completed version of **video-reflection-1** and **video-reflection-2** files.
4. Add a Reflection to the repository by modifying the `reflection` file in the lab repository. List out the biggest learning points and any challenges that you have encountered during this lab.
5. It is highly important, for you to meet the honor code standards provided by the college and to ensure that the submission is completed before the deadline. The honor code policy can be accessed through the course syllabus. Make sure to sign the honor-code file.

Grading Rubric

1. Details including the points breakdown are provided in the individual sections above.
2. If a student needs any clarification on their lab credits, it is strongly recommended to talk to the Professor. The lab credits may be changed if deemed appropriate.

