

Lab 2 Specification – Exploring Arrays in C
Due (via your git repo) no later than 8 AM, Thursday, 22nd September 2022.
50 points

Lab Goals

- Exploring the implementation of Arrays.
- Exploring the implementation of Pointers and Strings.
- Exploring File System and use of Pointers to access Files.

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 chapters 5 and 6**

Assignment Details

Now that we have discussed some more of C Programming, we are ready to explore and do a few more exercises to think logic, use low end computing tools such as Pointers, and implement it in C programs. We will also get an opportunity to implement Strings and explore a new territory of Files in C programming that was not directly 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-lab2` 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 cs200lab2 .
```

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, `cs200lab2` 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-lab2` directory to run the build command. If you are not inside the `cmpsc200-fall-22-lab2` 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 "`cs200lab2`" 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 cs200lab2
```

Windows:

```
docker run --rm -v "%cd%/src":/root -it cs200lab2
```

(b) To run the above command, it is required to be inside the `cmpsc200-fall-22-lab2` 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
```

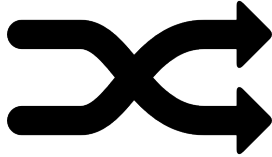
(f) After compiler the program, we can execute C Programs using the command within the container:

```
./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: Shuffling Program using Pointers



This section is worth 10 points.

We implemented a simple shuffling program, that swap the values stored inside two input variables. In this task, we will practice how to swap three input variables. To practice method invocation using Pointers, it is required to complete the tasks listed below:

Task 1 → Overview

The starter code is provided in the src/swap.c file. You may use as many variables as additional storage in the swap and prompt methods. However, the lesser you declare the more efficient your program is implemented. In general, the efficient implementation has fewer lines of code.

Assumptions:

1. Assume that the input provided by the user to be placed in each box is always an integer value.

Core Logic:

1. The prompt method is declared with three input parameters that are Pointers. The Pointers reference the address of the variables b1, b2, and b3 respectively that are declared in the main method.
2. Complete the prompt method by adding the appropriate logic to prompt the user to provide the number to be placed in each of the three boxes. The text to be used for the prompt message is shown in the sample output included in the Validation section. Next scanf the user input on the console to the respective pointer variable. Now you may ask a question. Do you need to use the ampersand sign in the scanf while you read into a pointer? Think - A pointer is already an address.
3. Assuming that the previous step is correctly completed the values provided by the user that are captured in the prompt method will now be available in the main method.
4. The swap method is declared with three input parameters that are Pointers. The Pointers reference the address of the variables b1, b2, and b3 respectively that are declared in the main method.
5. Complete the swap method by adding the appropriate logic to swap the numbers in the three boxes. The number in the First Box (b1) should be swapped with the number in the Second Box (b2). The number in the Second Box (b2) should be swapped with the number in the Third Box (b3). Finally, the number in the Third Box (b3) should be swapped with the number in the First Box (b1).

Let us suppose the initial values in the boxes are [10,20,30].

The values in the boxes after first round of swapping is [20,10,30]

The values in the boxes after second round of swapping is [20,30,10]

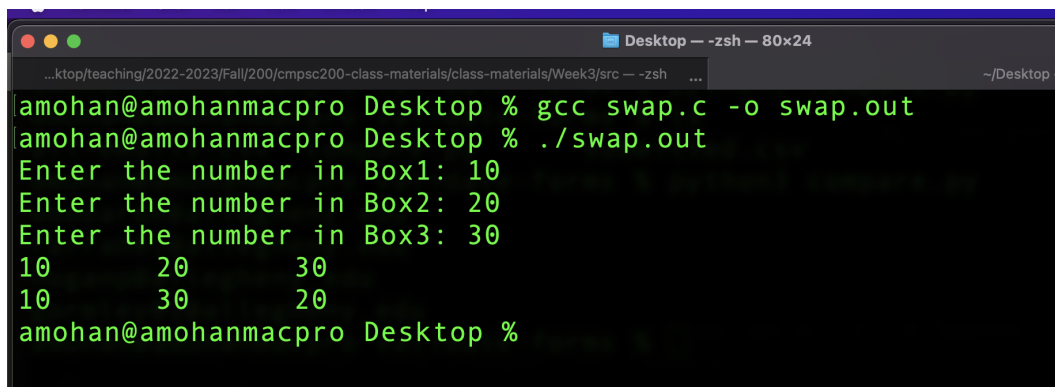
The values in the boxes after second round of swapping is [10,30,20] which is the Final Result.

6. It is not required to display the intermediate swapping results after every round of swapping.

Validation:

You are welcome to use your own test case for the numbers provided by the user. It is important to make sure the following test case is validated. The output from the swap program is provided below:

A sample screenshot for output



```
amohan@amohanmacpro Desktop % gcc swap.c -o swap.out
amohan@amohanmacpro Desktop % ./swap.out
Enter the number in Box1: 10
Enter the number in Box2: 20
Enter the number in Box3: 30
10      20      30
10      30      20
amohan@amohanmacpro Desktop %
```

Section 2: Meal Rating Program



This section is worth 10 points.

We implemented a simple rating program, to prompt, store, and process/print a homogeneous storage platform in Arrays, during our class discussions. In this section, we will practice more on Arrays that includes both one-dimensional and two-dimensional storage, which is driven by an understanding of how storage and contiguous processing works internally. To practice this array storage and processing problem, it is required to complete the tasks listed below:

SEE NEXT PAGE

Task 2 → Overview

This task should be implemented using one-dimensional data storage. The starter code is provided in the `src/meal.c` directory. You may use as many variables and intermediate arrays as additional storage in the `ratings` and `stats` method. However, the lesser you declare the more efficient your program is implemented. In general, the efficient implementation has fewer lines of code.

Assumptions:

1. Assume that there are 5 dining halls on campus.
2. Assume that there is an N number of students on campus. The number of students N is provided to the program through user input.

Core Logic:

1. Declare 5 individual one-dimensional arrays of integer data type, namely `h1`, `h2`, `h3`, `h4`, and `h5`. These arrays declared in the main method are representative of the storage used for each of the dining halls. This step is already complete in the starter code.
2. The `ratings` method is invoked from the main method by passing the 5 individual arrays and the number of students as a parameter. This step is already complete in the starter code.
3. Implement the **ratings method** to prompt each student to provide their ratings for all 5 dining halls and assign the corresponding ratings to the respective array storage. It is expected that the ratings method will fill out the empty arrays `h1` to `h5` for each dining hall. The prompts should be done one student at a time. That is, ask the first student their ratings for all 5 dining halls and store the ratings in the `h1` array. Next, repeat this for all the students. This step is not complete and is required to be completed.
4. The `stats` method is invoked from the main method by passing the 5 individual arrays and the number of students as a parameter. This step is already complete in the starter code.
5. Implement the **stats method** to iterate through the ratings in each array associated with the 5 dining halls. Find the average rating for each dining hall. The average rating is computed by dividing the sum of all ratings for a particular dining hall by the student count. Display all the dining halls and their respective average using `.2f`. That is, display the fraction value with two digits after the decimal point. To complete this, it may be required to cast the expression that you implemented to float by using the keyword `(float)`. This step is not complete and is required to be completed. Refer to the validation section for a sample output.
6. After displaying the average for all the dining halls, find the best dining hall. The best dining hall will be the one with the highest average rating. If there is a tie between two dining halls then just display one of them. You must represent the dining halls as `H1`, `H2`, `H3`, `H4`, and `H5` respectively in your output print statements.

Validation:

While you are welcome to use your own test case for the list of ratings provided by the user. It is important to make sure the following test case is validated. The stats output for the rating scores used in this test case is shown in the next page.

Let us suppose that the number of students is 5.

1st student input ratings are **5,6,7,8,9** for the five dining halls.

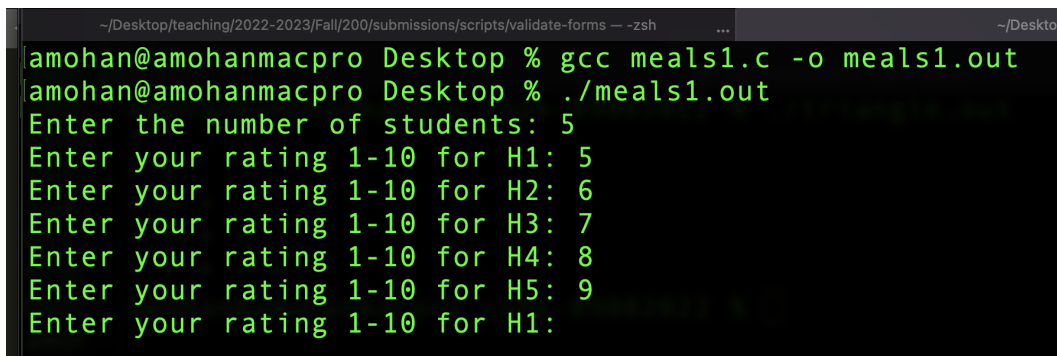
2nd student input ratings are **6,7,8,9,10** for the five dining halls.

3rd student input ratings are **3,4,5,6,7** for the five dining halls.

4th student input ratings are **10,9,8,2,3** for the five dining halls.

5th student input ratings are **2,3,4,7,8** for the five dining halls.

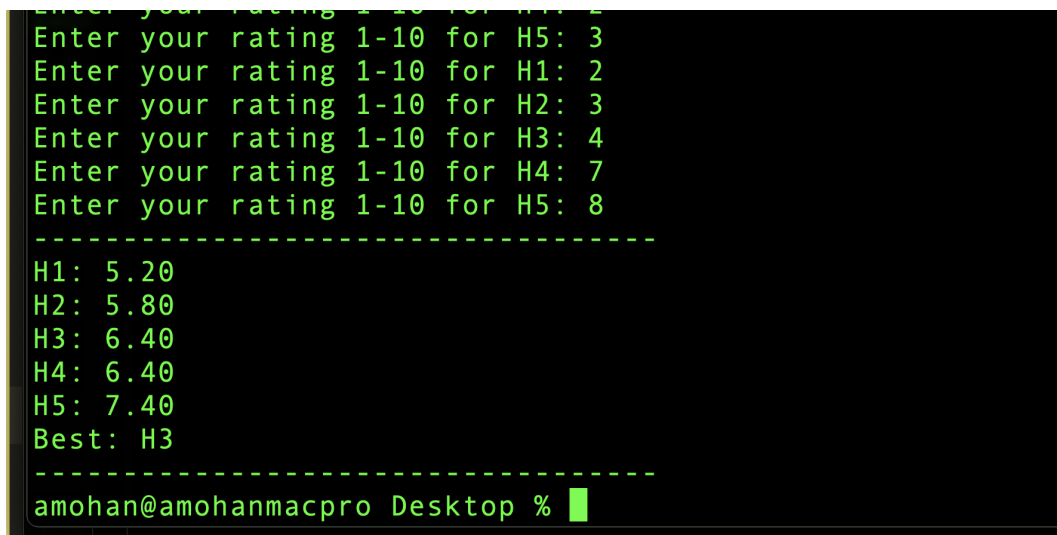
A sample screenshot for user input



```
~/Desktop/teaching/2022-2023/Fall/200/submissions/scripts/validate-forms — zsh ...
amohan@amohanmacpro Desktop % gcc meals1.c -o meals1.out
amohan@amohanmacpro Desktop % ./meals1.out
Enter the number of students: 5
Enter your rating 1-10 for H1: 5
Enter your rating 1-10 for H2: 6
Enter your rating 1-10 for H3: 7
Enter your rating 1-10 for H4: 8
Enter your rating 1-10 for H5: 9
Enter your rating 1-10 for H1:
```

Due to length sake, the screenshot is truncated and only shown the user input provided by the first student.

A sample screenshot for output



```
Enter your rating 1-10 for H1: 2
Enter your rating 1-10 for H5: 3
Enter your rating 1-10 for H1: 2
Enter your rating 1-10 for H2: 3
Enter your rating 1-10 for H3: 4
Enter your rating 1-10 for H4: 7
Enter your rating 1-10 for H5: 8
-----
H1: 5.20
H2: 5.80
H3: 6.40
H4: 6.40
H5: 7.40
Best: H3
-----
amohan@amohanmacpro Desktop %
```

Section 3: File Parsing Program.



This section is worth 20 points. The points breakdown is provided below:

- Task 3 = 10 points
- Task 4 = 10 points

It is worth making a note, that the file system is just the output component referenced in the Von Neumann architecture. So far, we primarily used the Display monitor screen, as the output component to process our programs. In this section, we will experience an unexplored territory of developing C programs using the File System. To complete this part, it is required to have a solid understanding of the basic concepts of C Programming from class discussions and the previous lab. It is important to review the slides, reason through and understand the logical tasks (binary using modulo 2) discussed, and also complete the reading assignments as required. To practice file parsing and integrating logic while parsing a file, it is required to complete the tasks listed below:

Task 3 & 4 → Overview

There is a C program provided in the starter-code repository, to help stay focused on the implementation details, and to get started with the development of this part. Review the starter-code in a file named `parser.c` and `ritchie.txt` in the "src" directory. The code is modified in the `parser` program so that it prints out only every other character (other than the newline character, which will always be printed), i.e., the first, third, fifth, seventh, ... This implementation is done by adding a counter and only printing when the counter has an even value. The text file named "ritchie.txt" can be used for test data, which is also available in the "src" directory. Read the comments to make sure you understand how the program works and the prompts to get the user input.

Assumptions:

First, you should read through the program on page 17 of K&R. This program is used to copy a file. To run it on a file named "myfile.txt", you first create a new file called `copy.c` with the code from the book. Next, you compile and execute the program by typing the following commands. If you receive syntax errors, read the errors carefully and try to fix it. Remember that only with practice and by dealing with errors we get better in programming.

```
gcc copy.c -o copy.out
./copy.out < myfile.txt
```

It is not required to submit the `copy.c` program. This is a preparation for the subsequent tasks listed in this section.

Core Logic:

1. **Task 3:** Write a C program modeled after the ones in sections 1.5.3 and 1.5.4 of K&R that do the following:
Given an input file, print it with leading line numbers, starting with line 1. Assume there are no more than 999

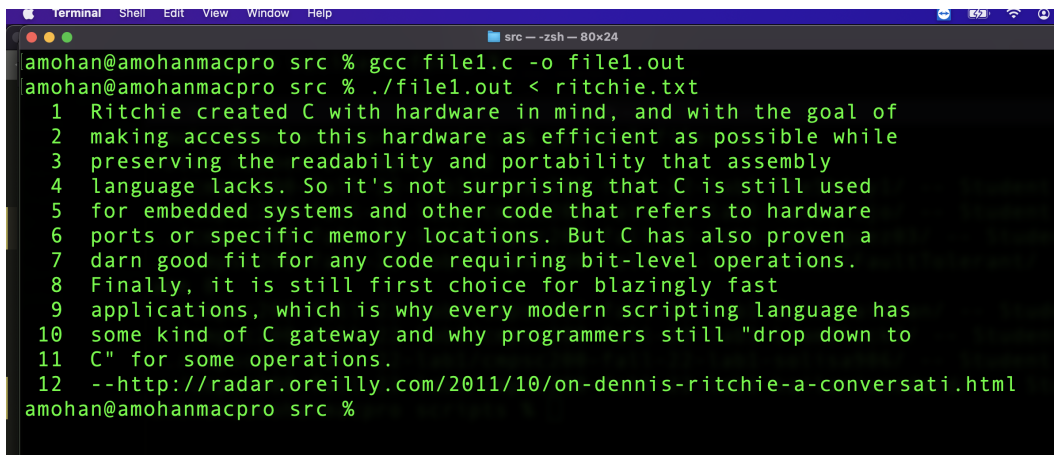
lines in the file. Line numbers should be right-justified in the first **3 columns**. Submit your work, using the file named `file1.c`.

2. **Task 4:** Write a C program modeled after the ones in sections 1.5.3 and 1.5.4 that does the following: Given an input file, count the number of vowels and consonants and print out these counts, appropriately labeled. The vowels are a, e, i, o, and u (both upper and lower case); all other letters are consonants. You may not use any built-in C functions for checking for upper-case, etc. Submit your work, using the file named `file2.c`.

Validation:

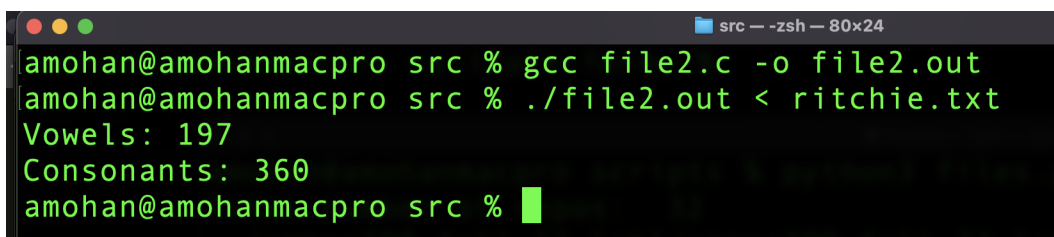
While you are welcome to use your own file as a test case for testing the correctness of your implementation, the "richie.txt" file is provided in the "src" directory to run your test case. It is important to make sure the following test case is validated. The Task-3 output from the "file1.c" and Task-4 output from the "file2.c" programs are shown below for reference:

A sample screenshot from executing Task-3



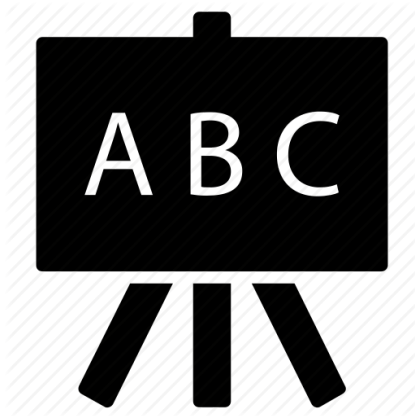
```
amohan@amohanmacpro src % gcc file1.c -o file1.out
amohan@amohanmacpro src % ./file1.out < richie.txt
 1 Ritchie created C with hardware in mind, and with the goal of
 2 making access to this hardware as efficient as possible while
 3 preserving the readability and portability that assembly
 4 language lacks. So it's not surprising that C is still used
 5 for embedded systems and other code that refers to hardware
 6 ports or specific memory locations. But C has also proven a
 7 darn good fit for any code requiring bit-level operations.
 8 Finally, it is still first choice for blazingly fast
 9 applications, which is why every modern scripting language has
10 some kind of C gateway and why programmers still "drop down to
11 C" for some operations.
12 --http://radar.oreilly.com/2011/10/on-dennis-ritchie-a-conversati.html
amohan@amohanmacpro src %
```

A sample screenshot from executing Task-4



```
amohan@amohanmacpro src % gcc file2.c -o file2.out
amohan@amohanmacpro src % ./file2.out < richie.txt
Vowels: 197
Consonants: 360
amohan@amohanmacpro src %
```

Section 4: Strings Program



This section is worth 10 points.

We implemented a simple program involving strings to display a string in forward and reverse order. We discussed Strings are just an array of characters and we can iterate through strings like any other array. In this task, we will practice how to compare strings and their characters in the forward and reverse order. To practice Strings and Character array usage in Strings, it is required to complete the tasks listed below:

Task 5 → Overview

The starter code is provided in the `src/palindrome.c` file. You may use as many variables as additional storage in the `palindrome` method. However, the lesser you declare the more efficient your program is implemented. In general, the efficient implementation has fewer lines of code.

Assumptions:

1. Assume that the input provided by the user is always lesser than 20 characters.

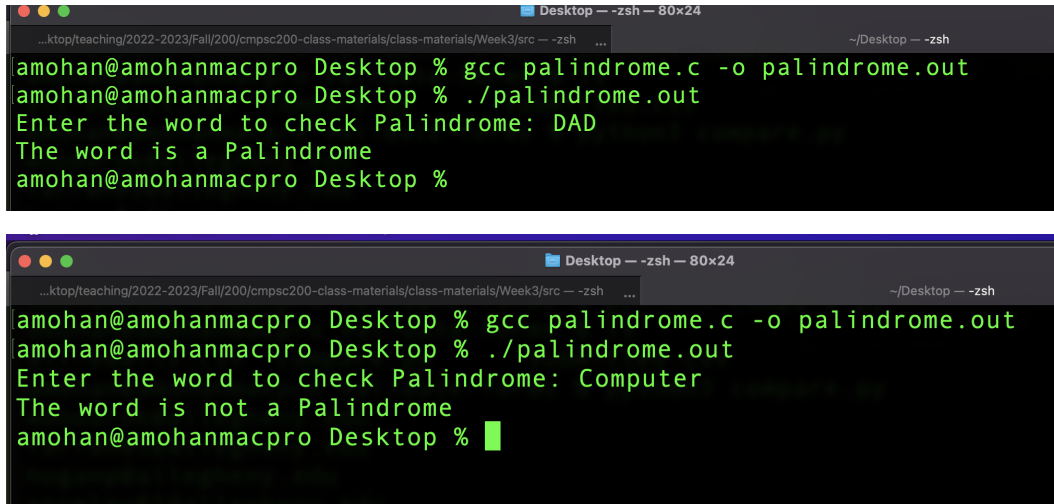
Core Logic:

1. The main method prompt the user to type in an input string and store the string into a char pointer array of predefined size 20. That is, we assume that the words typed are of length 20 or less.
2. Implement the logic in the `palindrome` method to validate if they are equal. If they are equal then print "The word is a Palindrome", else print "The word is not a Palindrome". For example "DAD", "MOM", "CIVIC", "KAYAK", etc.. are palindromes.
3. You are only allowed to use character arrays to implement this part. The usage of any external string libraries, implementation using `strcmp`, pointers, and other library based functions is not acceptable.
4. It is strongly recommended to use Pointer arithmetic to iterate the array. However, I will not stop you from using the indexing method using square brackets if you prefer to do so.

Validation:

You are welcome to use your own test case for the numbers provided by the user. It is important to make sure the following test case is validated. The output from the strings program is provided below:

A sample screenshot for output



```
amohan@amohanmacpro Desktop % gcc palindrome.c -o palindrome.out
amohan@amohanmacpro Desktop % ./palindrome.out
Enter the word to check Palindrome: DAD
The word is a Palindrome
amohan@amohanmacpro Desktop %

amohan@amohanmacpro Desktop % gcc palindrome.c -o palindrome.out
amohan@amohanmacpro Desktop % ./palindrome.out
Enter the word to check Palindrome: Computer
The word is not a Palindrome
amohan@amohanmacpro Desktop %
```

Section 5 - 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 6 - 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 **swap.c** file.
2. updated completed version of **meal.c** file.
3. updated completed version of **file1.c** and **file2.c** file.
4. updated completed version of **palindrome.c** file.
5. 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.
6. 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.

