Lab 02 Specification – Exploring Programming in C
Due (via your git repo) no later than 2 PM, Friday, 10th September 2021.
50 points

Lab Goals

- Exploring the implementation of Binary Match Table.
- Develop a series of C program(s) involving character i/o, logical, and iterative programming tasks.

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 1 and 2**

Assignment Details

Now that we have discussed some basics of C Programming, we are ready to explore and do a few exercises to think logic and implement it in C programs. We will also get an opportunity to implement the logical solution of the binary match table, and prepare to implement other low end computing tasks in this course.

In this lab, students will practice developing a logical and programmatic solution to problems such as generating a **X pattern** and **File Operations** in C to retain the knowledge from the class discussions so far.

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 skill tests, 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-21-lab02 directory using terminal (Mac/Ubuntu) or Command Prompt/Docker quick start terminal (windows).

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

   docker build -t cs200lab02 .

   Please note, you are required to have the period in the command above.

   (c) Note: In the command above, cs200lab02 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-21-lab02` directory to run the build command. If you are not inside the `cmpsc200-fall-21-lab02` 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 ”cs200lab02” 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 cs200lab02
   ```

   Windows:

   ```
   docker run --rm -v "%cd%/src":/root -it cs200lab02
   ```

   (b) To run the above command, it is required to be inside the `cmpsc200-fall-21-lab02` directory. And, please note, you will log in to the container after entering the above command.

   (c) 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.

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

   gcc hello.c -o hello.out

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

   ./hello.out

   (f) **[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: Stars Program

This section is worth 15 points. The points breakdown is provided below:

- Task 1 = 5 points
- Task 2 = 10 points

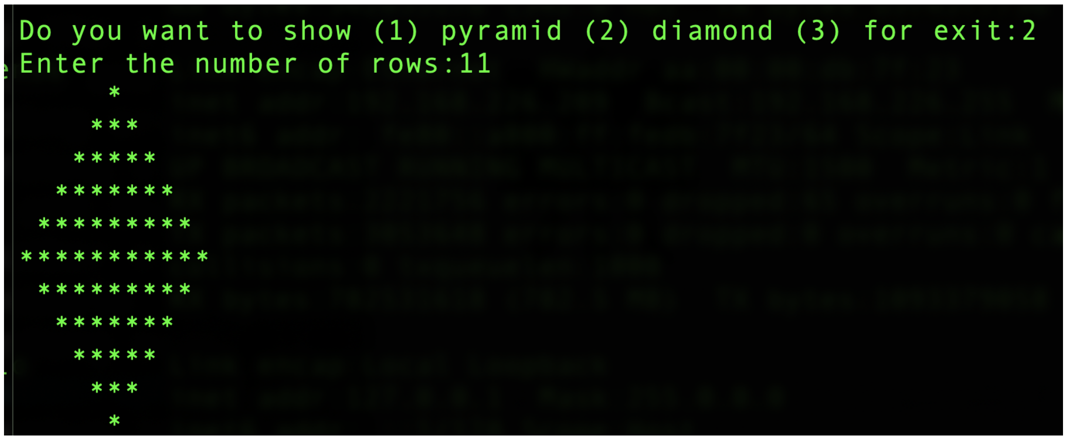
We implemented a nested for-loop program, to pretty print a series of asterisks, during our class discussions. Additionally, we did a thinking exercise on developing a pyramid pattern in C. In this section, we will practice an important logical & iterative problem of pattern printing, which is driven by an understanding of binary values 0 and 1. To practice this logical problem, it is required to complete the tasks listed below:

1. **Task 1:** Read through the pattern of the stars printing program provided in the starter-code repository with a file named **stars.c**. This code is a variation of the initial version of this program that we did in class, named `pretty.c`. The program has a main method, that prompts the user for an input repeatedly inside a while loop. Based on this input prompt, the program will do one of the following: <print a pyramid, print a diamond, exit>. The `pyramid` method in the starter code has the complete implementation for the pyramid printing part. The pyramid printing is based on displaying two characters, that is an asterisk and space. The logical challenge in solving such a problem is to understand the pattern in printing these characters. Here we used a variable named `mid` to find the midpoint for each row. It is worth noting that, this midpoint is the same index number for all rows. A nested for-loop is used to channelize the printing procedure, which is iterated through the rows and columns. Assume the number of rows = 5. Develop a table to list all the different i, j values, and the printed character for the corresponding (i, j) value combination. A sample incorrect table is provided below, for reference to the expected structure for creating the table. Please fill out the table correctly and upload your solution using a file named `stars-table`.

i	**j**	**char**
0	0	**space**
0	1	*
1	0	**space**
1	1	*
2	3	*
3	4	**space**
4	3	**space**

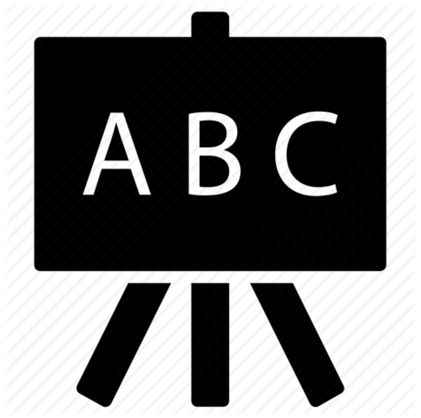
2. **Task 2:** Read through the `diamond` method in the starter code. This code is incomplete. Code review should indicate that the user is prompted to provide the number of rows. This user prompt is implemented using the combination of scanf and `getchar` builtin methods. A `getchar()` is often added next to a `scanf`, so that the program executes correctly when there more than one scanf inside the complete program. Here add the corresponding logic to identify the odd or even numbers. If the number provided is even, simply print a message to the user asking them to enter an odd number. If the number is odd, then setup the corresponding code as per the details provided in Task 3. This part should be implemented using an if-else condition and the

special operator that we discussed in class to separate odd and even numbers. **Task 3:** Inside the else condition, add the respective lines of code to figure out the midpoint value. From there, it is required to set up a nested for loop, similar to the pyramid section, but the internal logic is slightly different. Here, we are printing a diamond and not a pyramid. Add the respective to complete this section. To solve this part effectively, create a table similar to the `stars-table` developed in Task 1. Do this table for a few cases, such as the number of rows = 3, 5, 7, 9. Doing these tables will help you to think through the logic needed for solving this problem. A sample screenshot of the output of the program is provided below for reference:



```
Do you want to show (1) pyramid (2) diamond (3) for exit:2
Enter the number of rows:11
     *
    ***
   *****
  *******
 *********
***********
 *********
  *******
   *****
    ***
     *
```

Section 2: ASCII Casing Program.



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

- Task 4 = 10 points
- Task 5 = 10 points

We implemented an ASCII program, to process and print the ASCII code for each character typed in from the keyboard, during our class discussions. In this section, we will do an important realization of how characters are decoded by computers, which is then translated into binary values of 0 and 1. To practice this low-end computation, it is required to complete the tasks listed below:

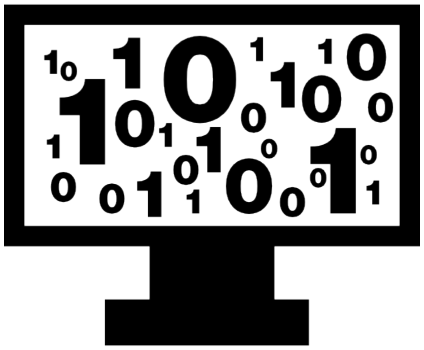
1. **Task 3:** Read through the character decoding program provided in the starter-code repository with a file named **casing.c**. This code is a variation of the initial version of this program that we did in class, named `ascii.c` and `position.c`. The program has a main method, that prompts the user for an input repeatedly inside a

while loop. Based on this input prompt, the program will do one of the following: <print a lower case text, print an upper case text, exit>. The `toUpper` method in the starter code has the complete implementation for the conversion of the provided text to upper case format. The program utilizes a while loop internally to capture the primitive one character at a time. The characters are then translated to the upper case format. Please look at your class notes, and slides to check your understanding of the ASCII mapping table discussed in class. The program will simply print the character as it is if it encounters a character that is not a lower case. If there is a lower case character, then the program does the appropriate conversion.

2. **Task 4:** The `toLower` method in the starter code is incomplete. Add the appropriate code in this method, to capture the primitive one character at a time using the while loop, similar to the `toUpper` method. Inside the while loop, add the appropriate code, to convert the user typed characters into its equivalent lower case format. Print the converted characters as output. If the character is not upper case format, then simply print the actual character. Reviewing the code in Task 3 should help a lot in completing this task.

3. **Task 5:** The `toCamel` method in the starter code is incomplete. Add the appropriate code in this method, to capture the primitive one character at a time using the while loop, similar to the `toUpper` method. Inside the while loop, add the appropriate code, to convert the user typed characters into its equivalent camel case format. In the camel case, it is required to remove any space in the list of characters and make sure that the first character of each word is capitalized and the other characters are lower case. For example: if the user type **coMpuTer orgaNiZation**, the output should be **Computer Organization**.

Section 3: Binary Match Table Program



- Task 7 = 15 points

We solved the binary match table, to convert the decimal into its equivalent binary through toggling 0's and 1's, during our class discussions. In this section, we will do an important realization of the procedure to solve this problem, programmatically. To practice this low-end computation, it is required to complete the tasks listed below:

1. **Task 6:** Read through the binary match table development program provided in the starter-code repository with a file named **binary.c**. This code is a programmatic trial to implement the binary match table that we did through paper and pen. The program has a main method, that prompts the user for an input repeatedly inside a while loop. Based on this input prompt, the program will do one of the following: build the binary table for the user-provided number of rows, or exit the program. The `buildTable` method in the starter code has the partial code to implement the binary match table. To implement such a program, there is three critical logic that is required to be implemented in the program, namely:

- Compute the number of columns in the binary match table based on the number of rows. For example, there are 3 columns for 5 rows, 3 columns for 7 rows, 4 columns for 8 rows, and so on. This could be computed using the formula with a combination of ceil and log mathematical operators, displayed below:

  $cols = \lceil(\log_2(rows))\rceil$

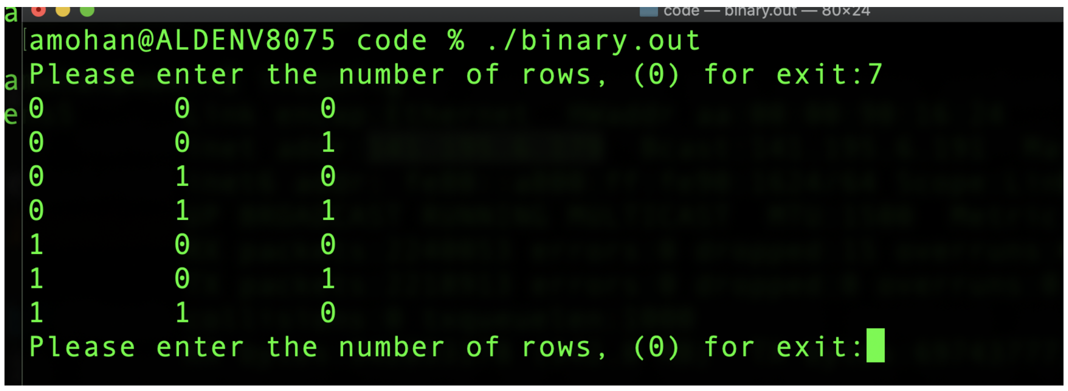
  It may be worth noting that a ceil operator would simply take a fraction as input, let us suppose (4.5) and output it as 5. A floor operator would output 4 for the same value.

  This part is implemented in the starter-code in line number 6. The math.h library is included in the program to utilize the pow, log, and ceil operators in the program. It is important that you think through this code, and realize the formalization that went in to implement this complicated and challenging part, just using one line of code. This implementation is a typical way of how computers think internally at the machine level.

- Set up the iteration using the nested for-loop to iterate through the rows and columns. The outer for-loop runs through the ascending order, whereas the inner for-loop runs through the descending order. The outer for-loop represents the number of rows, and the inner for-loop represents the number of columns. The number of columns is simply the number of bits in each row, for example if there are 4 columns, bit arrangement sequence in descending order is:b_3, b_2, b_1, b_0. To make things simple, this setup is already complete in the starter-code. It is important to reason this code for completing the next critical step.

- Set up the right code for printing the binary values based on the i and j values from the nested for-loop. This part is incomplete and required to be implemented.

2. **Task 7:** The `buildTable` method in the starter code is incomplete. There is one additional line that is required to be added to this method, which is in line number 9, to make the program fully functional. Although it may seem that the points are given for just implementing one line of code, this takes some work to figure out this line of code. A solid understanding from the code review in the previous task is instrumental to have this code implemented. The hint behind solving this part is to come up with a formula, using the combination of (modulo) and (pow) mathematic operators. To implement this part, it is required to include i and j values, in the formula, so that the printing is based on i and j values and the inclusion of modulo and pow operators. To reveal one step further, 2 to the power of one of the indices (i or j) is used in the formula. Why do we need to implement such a challenging formula? This practice is critically important to achieving one of the learning goals of the course, which is, understand the way machines compute internally. As a next step, as we move forward in the course, we would learn how computers compute the different mathematical operators internally. A sample screenshot of the output of the program is provided below for reference:



```
amohan@ALDENV8075 code % ./binary.out
Please enter the number of rows, (0) for exit:7
0       0       0
0       0       1
0       1       0
0       1       1
1       0       0
1       0       1
1       1       0
Please enter the number of rows, (0) for exit:
```

Submission Details

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

1. **stars-table** file
2. updated version of **stars.c** file
3. updated version of **casing.c** file
4. updated version of **binary.c** file
5. It is highly important, for you to meet the honor code standards provided by the college and to ensure that the submission is made before the deadline. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine unless otherwise cited.” in all your deliverables such as source code and PDF files.

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.