Design and Code Document - DCD
Your DCD for the problem you will be solving should be called IOS_2019_X_Hackathon_XXX, and should reside in the problem directory with the rest of your code. Your design should describe the design of your problem statament in enough detail that a knowledgable programmer could duplicate your work. This includes descriptions of the data structures you're going to use, all non-trivial algorithms and formulas, and a description of each function including its purpose, inputs, outputs, and assumptions it makes about the inputs or outputs. The DCD along with the source Code and respective a.out image should be zipped / compressed. The name of the file should be strictly IOS_2019_X_Hackathon_XXX.zip only. Note DCD compressed as .rar will not be considered for assessment. The marks will be based on DCD (7 Marks) and Viva (3 Marks).

Implementation should be strictly using C Language and Ubuntu OS Only

Coding Style
Good coding style enhances program readability, clarity, meaning, simplicity, etc. Good coding style is an art as well as a science, but there are a few simple rules you can follow that will greatly enhance the readability and clarity of your code. Modular design in the form of interface and implementation will be encouraged.

Comments:
Every file must contain a comment at the top that describes the code in that file. In any file that you create, this comment must start with the word "Author(s): " followed by your <srn,name> and the date that you created the file.
Every function must have a comment before it describing what it does, what parameters it takes, what it returns, and what assumptions it makes.

Every non-trivial block of code must have a comment saying what it does. For our purposes, non-trivial means 3-10 lines of code, depending upon the complexity. As a guideline, you should have about as many lines of comments as you have lines of code.
Every comment must be indented properly given its location in the file, on the preceding line and aligned with the code it describes.

Indentation
After every left brace, indent a fixed number of spaces (2-4).
With every right brace, unindent the same amount.
Everything else should be aligned accordingly.
Long lines (printfs, comments, etc.) that wrap around should be broken into parts so that they do not wrap.
Do not indent with tabs unless you know that your editor substitutes spaces for tabs

Spacing
Every comment (except the one at the top of the file) must be preceded by a blank line. There should be no blank line between a comment and the code it describes.
A blank line (and often a comment) should come before every chunk of code that does something different from the code before it.
A blank line should come between all declarations and all other code in a function.
A blank line should follow any right brace, unless the following line also has a right brace.

Braces

When declaring a function, the braces go in column 0. The left brace goes on the line after the function declaration, and the right brace goes after the body of the function.

Any other time, such as with an if, for, or while, the left brace goes on the same line as the if, for, or while, and the right brace aligns with the if, for, or while on its own line.

Identifiers

Identifiers (functions, variable names) should use only lower-case letters, with underscores between words.

Identifiers should be as meaningful as possible, without growing too long. Don't use more than two words in your identifiers unless absolutely necessary.

Single-letter identifiers may not be used except as loop counters, and then only when something more meaningful is undesirable for some very good reason.

And, of course, you may not use global variables unless no other solution will work. Any global variables must be justified and explained in your design document.