Programming Assignment 1: Fracture

COP 3223, Fall 2024, Juan Parra

Due: September 6th 2024 before 11:59 PM

Table of Contents

Objective

In this assignment, you will write a program that will house many internal functions. Instead of typing everything within main(), you will create functions with different responsibilities to start "fracturing" your code. Fracture, in this case, is not meant to be a negative term. Think of it like this, when there's a fracture in something, one must understand how it came to be, isolate said problem, and work on it without the bias of external factors.

All requirements below will assume a shape of a circle for all calculations.

For those eager to pursue Software engineering (or SWE for short), you will learn to construct programs that does not **overwhelm** itself. There's a rule of thumb that says 50 lines per function (no more than that) but there's an even better way of describing this:

- Avoid giving a function anxiety for all the stuff it has to do. Or in other words, DRY.

DRY stands for "Don't Repeat Yourself" and is meant to minimize redundancy by ensuring that any piece of knowledge or logic is represented in a single, unambiguous place within the system.

If you do end up pursuing a career in SWE, you must be able to process technical documentation and work with new environments or languages. This will be essential to your career and this assignment will provide you with an exercise in doing just that.

1 Motivation

Fracturing tends to be seen as a negative thing. As if, any sort of break or crack, is considered weak. I'd like to view it in a different light. Think of it like this, when there's a fracture, the only way to address it is through isolation and seclusion.

If unable to isolate the fracture, one can get carried away and forget what they were actually working on.

```
void print_an_addition_operation(int arg1, int arg2)
{
    printf("%d\n", arg1 + arg2);
}

void print_a_division_operation(int arg1, int arg2)
{
    printf("%f\n", arg1 / arg2);
}

int main(void)
{
    print_an_addition_operation(5, 5);
    print_a_division_operation(6, 3);
}
```

However, if you are unable to fracture a program and keep everything in one function, you face what I call, anxiety.

```
int main(void)
{
    int x = 5 + 5;
    float = 6 / 3;
    int z = y * x;

    printf("%d\n", x);
    printf("%d\n", y);
    printf("%d\n", z);

}
```

In the example above, you start seeing main get well alot to look at. Usually this is fine since its under the 50 line limit but if you look closely, you're more at risk to make mistakes. I even forgot to put **y after float**! Besides that, you're doing essentially **three** different functions!

Now there is a disclaimer I need to mention. **Not everything** has to be split up. It is very common as you progress in your career to have functions that does

multiplication, iteration, memory allocation, in one go. For the purpose of this assignment, we want to get used to breaking stuff up. However, as you progress, the responsibility of fracturing will rely on you and your best judgment. I will also say, be open to feedback. Sometimes one can get carried away with their code and overfill it or break up too unnecessary. It's all a learning experience and you will get better over time.

2 Requirements

You will submit a single source file, named **fracturing.c**, that contains all required function definitions, and any helper functions you deem necessary.

3 Adopting a Growth Mindset

Excerpt from Dr. Szumlanski:

A word of advice before we dive in to the details: When faced with an assignment like this, which has many different facets, some of which might appear foreign and/or challenging, it's important not to look at it as an instrument that is being used to measure how much you already know (whether that's knowledge of programming, operating systems, or even what some might call "natural intellectual capability"). Rather, it's important to view this assignment as a chance to learn something new, grow your skill set, and embrace a new challenge. It's also important to view intellectual capability as something that can grow and change over one's lifetime. Adopting that mindset will allow you to reap greater benefits from this assignment (and from all your college-level coursework) regardless of the grades you earn.

4 Function Requirements

In the source file you submit, **fracturing.c**, you must implement the following functions. You may implement helper functions to make these work, as well, although that may or may not be necessary for this assignment. Please be sure the spelling, capitalization, and return types of your functions match these prototypes exactly.

For a quick reminder, helper functions are simply user defined functions meant to separate logic into new functions. They do not have to follow the restrictions of the required functions. Example: have a function to print and a function to read user input

As for user input, you can ask once (so using two points throughout the program's lifetime) or re-ask (so 5 sets of points). It is up to you how you want to implement this (just make sure to not change the required function definitions). Each set of points represent the **diameter**.

int main(int argc, char **argv);

- **Arguments**: This function have should have the two defined arguments above.
- **Output**: This function should have no output.
- **Return**: A simple integer. Just 0. **This is to allow the autograder to work.**
- **Disclaimer**: This function should be used to call the other functions.

double calculateDistance();

- **Arguments**: This function have no arguments.
- **Output**: This function should output three lines of text.
  - Point #1 entered: x1 = -whatever was entered-; y1 = -whatever was entered-
  - Point #2 entered: x2 = -whatever was entered-; y2 = -whatever was entered-
  - The distance between the two points is -whatever is computed-
- **Return**: A double representing the distance.

double calculatePerimeter();

- **Arguments**: This function have no arguments.
- **Output**: This function should output two lines of text.
  - Point #1 entered: x1 = -whatever was entered-; y1 = -whatever was entered-
  - Point #2 entered: x2 = -whatever was entered-; y2 = -whatever was entered-
  - The perimeter of the city encompassed by your request is -whatever is computed-
- **Return**: A double indicating how difficult you found to do this function on a scale of 1.0 through 5.0 where 1 is easy and 5 is hard
- **Disclaimer**: Must use a function (like calculateDistance) within to avoid repeating code.

double calculateArea();

- **Arguments**: This function have no arguments.
- **Output**: This function should output three lines of text
  - Point #1 entered: x1 = -whatever was entered-; y1 = -whatever was entered-
  - Point #2 entered: x2 = -whatever was entered-; y2 = -whatever was entered-
  - The area of the city encompassed by your request is -whatever is computed-
- **Return**: A double indicating how difficult you found to do this function on a scale of 1.0 through 5.0 where 1 is easy and 5 is hard
- **Disclaimer**: Must use a function (like calculateDistance) within to avoid repeating code.

double calculateWidth();

- **Arguments**: This function have no arguments.
- **Output**: This function should output three lines of text
  - Point #1 entered: x1 = -whatever was entered-; y1 = -whatever was entered-
  - Point #2 entered: x2 = -whatever was entered-; y2 = -whatever was entered-
  - The width of the city encompassed by your request is -whatever is computed-
- **Return**: A double indicating how difficult you found to do this function on a scale of 1.0 through 5.0 where 1 is easy and 5 is hard

double calculateHeight();

- **Arguments**: This function have no arguments.
- **Output**: This function should output three lines of text
  - Point #1 entered: x1 = -whatever was entered-; y1 = -whatever was entered-
  - Point #2 entered: x2 = -whatever was entered-; y2 = -whatever was entered-
  - The height of the city encompassed by your request is -whatever is computed-
- **Return**: A double indicating how difficult you found to do this function on a scale of 1.0 through 5.0 where 1 is easy and 5 is hard

BONUS (Optional): double askForUserInput();

- **Arguments**: This function have no arguments.
- **Output**: This function should output nothing.
- **Return**: A double representing the user's typed in value... this will attribute to the DRY method as mentioned earlier and can be applied to the previous functions.

An example of main calling the functions would be:

```
    int main(void)
    {
        calculateDistance();
        calculatePerimeter();
        calculateArea();
        calculateWidth();
        calculateHeight();

        return 0;
    }
```

5 Special Restrictions

You must adhere to the following special restrictions when writing this assignment.

5.1 Use #define for PI

You must declare a preprocessor directive for PI that will hold the value: 3.14159

5.2 No Global Variables

You may not use any global variables in this assignment. (A global variable is a variable that is declared outside of a function – e.g., just below a #define statement – and can therefore be accessed by any and all functions in your source file. Example below:

```
int global_variable = 5;

int main(void)
{
    printf("%d\n", global_variable);

    return 0;
}
```

5.3 No File I/O

Your program cannot read or write to any files.

6 Compilation and Testing

6.1 IDEs

The key to getting multiple files to compile into a single program in Codespaces, Codeblocks, or any IDE is to create a project. Here are the step-by-step instructions for creating a project in CodeBlocks (if you choose to work locally), which involves importing the file you've created (even if it's just an empty file so far).

1. Start CodeBlocks.
2. Create a New Project (File -> New -> Project).
3. Choose "Empty Project" and click "Go."
4. In the Project Wizard that opens, click "Next."
5. Input a title for your project (e.g., "Ohce").
6. Choose a folder (e.g., Desktop) where CodeBlocks can create a subdirectory for the project.
7. Click "Finish."

Now you need to import your files. You have two options:

1. Drag your source and header files into CodeBlocks. Then right click the tab for each file and choose "Add file to active project."– or
2. Go to Project -> Add Files.... Browse to the directory with the source and header files you want to import. Select the files from the list (using CTRL-click to select multiple files). Click "Open." In the dialog box that pops up, click "OK."

You should now be good to go. Try to build and run the project (F9)

6.2 Linux/Mac Command Line

To compile your source file (.c file) at the command line:

```
gcc fracturing.c
```

By default, this will produce an executable file called a.out, which you can run with command line arguments like so:

```
./a.out
```

If you want to name the executable file something else, use

```
gcc fracturing.c -o fracturing
```

...and then run the program with command line arguments like so:

```
./fracturing
```

An even better way is combining both commands (by using the Ampersand symbol):

```
gcc fracturing.c -o fracturing && ./fracturing
```

7 General Tips for Working on Programming Assignments

Here's some general advice that might serve you well in this and future assignments in any course:

1. In general, it is ideal to code in short bursts and review your changes early before waiting too long. Example, you check after the completion of a short function or after a few lines in a complex function. The idea is to make sure you are progressively succeeding rather than progressively failing.
2. You're bound to encounter errors in your code at some point. Use printf() statements to verify that your code is producing the results you think it should be producing (rather than making assumptions that certain components are working as intended). You should get in the habit of not trusting your own code and using printf() to provide yourself with evidence that your code does what you think it does.
3. You will eventually want to read up on how to set break points and use a debugger. Some helpful Google queries might be: CodeBlocks debugger and gdb debugging tutorial.

8 Deliverables (Submitted via Webcourses, not Github Classroom)

Submit a single source file, named **fracturing.c**, via Webcourses. The source file should contain definitions for all the required functions (listed above), as well as any helper functions you need to make them work.

Ensure you submit the collaboration log. Other than those two, do not submit additional source files.

Be sure to include your name and UCFID as a comment at the top of your source file similar to how you did codespaces setup.

9 Grading

The *tentative* scoring breakdown for this programming assignment is:

- 10% calculateWidth is implemented correctly
- 10% calculateHeight is implemented correctly
- 20% calculateDistance is implemented correctly
- 20% calculateArea is implemented correctly
- 20% calculatePerimeter is implemented correctly
- 10% implementation details (manual inspection of your code)
- 10% adequate comments and whitespace (so styling); source includes student name and UCFID

Your grade will be based largely on your program's ability to compile and produce the exact output expected. Even minor changes (such as capitalization or punctuation errors) in your output will cause your program's output to be marked as incorrect, resulting in point deductions. The same is true of how you name your functions and their parameters. Please be sure to follow all requirements carefully and test your program thoroughly.

Additional points will be awarded for style (appropriate commenting and whitespace) and bonus functions in regards to requirements. For example, the graders might inspect to see if you compartmentalize the user input function to avoid repeating the same 4 lines per function...

As a general reminder, please do not wait till the last second to read this or work on the code. It's easy for life to get in the way and especially important to get ahead when given the resources. You got this, good luck!