CptS 121 - Program Design and Development

Programming Assignment 2: A Modular Approach to the Equation Evaluator

Assigned: Friday, September 6, 2024
Due: Friday, September 13, 2024 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Analyze a basic set of requirements and apply top-down design principles for a problem
- Customize and define C functions
- Apply the 3 file format: 1 header file and 2 source files
- Document and comment a modular C program according to class standards
- Implement guard code in a header file

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Access Microsoft Visual Studio 2022 Integrated Development Environment (IDE)
- Analyze a basic set of requirements for a problem
- Declare variables
- Apply C data types and associated mathematical operators
- Comment a program according to class standards
- Logically order sequential C statements to solve small problems
- Compose a small C language program
- Compile a C program using Microsoft Visual Studio 2022
- Execute a program
- Create basic test cases for a program
- Summarize topics from Hanly & Koffman Chapter 3 including:
  - The 3 components to applying and defining a function include: function prototype, function definition, and function call
  - What is a structure chart
  - Top-down design principles
  - What is an actual argument and formal parameter
  - Differences between local and global variables and scope

III. Overview & Requirements:

For this C program you will build a modular equation evaluator (you may want to start from your solution to programming assignment 1). Once again, you will write a C program that evaluates the equations provided below. The program must prompt the user for inputs to each equation and evaluate them based on the inputs. All equations should be placed into a single .c file. All variables on the right hand sides of the equations must be inputted by the user. All variables, except for the *plaintext*_character, *encoded_character*, *offset*, and variable *a* are floating-point values. The *plaintext_character* and *encoded_character* variables are characters, and the *offset* and *a* variable are integers. PI, G must be defined as a constant macros (#defined constants). Error checking is NOT required for your program. You do NOT need to check for faulty user input or ***dividing by zero***.

1. Newton's Second Law of Motion: force = mass * acceleration
2. Volume of a cylinder: volume_cylinder = PI * radius^2 * height
3. Character encoding: encoded_character = offset + (plaintext_character - 'a') + 'A' (note: what happens if plaintext_character is lowercase?)
4. Gravity: force = G * mass1 * mass2 / distance^2, where G is the gravitational constant with value 6.67 * 10^{-11}
5. Fahrenheit to Celsius conversion: celsius = (fahrenheit – 32) / (9 / 5) (note: the 9 and 5 constants in the equation should be left as integers initially, but explicitly type-casted as floating-point values)
6. Distance between two points: distance = square root of $((x_1 - x_2)^2 + (y_1 - y_2)^2)$ (note: you will need to use sqrt ( ) out of <math.h>)
7. General equation: y = (89 / 27) - z * x + a / (a % 2) (recall: *a* is an integer; the 89 and 27 constants in the equation should be left as integers initially, but explicitly type-casted as floating-point values)

For this assignment you are required to define, at a minimum, the functions provided below (note: these are your required *prototypes*!):

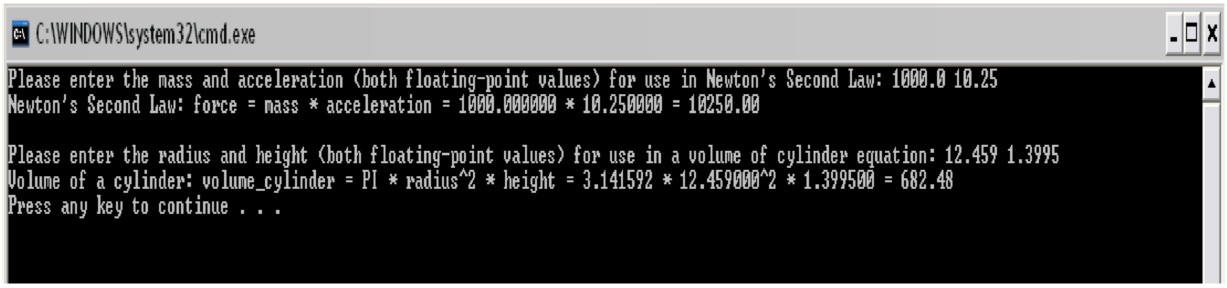
- double calculate_newtons_2nd_law (double mass, double acceleration)
- double calculate_volume_cylinder (double radius, double height)
- char perform_character_encoding (char plaintext_character, int offset)
- A function for calculating the force (you must decide on a function name, return type, and parameter list!)
- A function for converting Fahrenheit to Celsius (you must decide on a function name, return type, and parameter list!)
- A function for calculating the distance between two points (you must decide on a function name, return type, and parameter list!)
- A function for evaluating the general equation (you must decide on a function name, return type, and parameter list!)

A function must be defined for each of the above function signatures. The task that is performed by each function corresponds directly to the equations defined under the Equations section. For example, the *newtons_2nd_law ( )* function should evaluate the equation defined as force = mass * acceleration and return the resultant force, etc. You must print the results to the **hundredths** place. Also, the functions should not prompt the user for inputs. Prompts should be performed in main ( ) directly.

For this assignment you will need to define three different files. One file, called a header file (.h) needs to be defined which will store all #includes, #defines, and function prototypes. Name the header file for this assignment equations.h. The second file that needs to be defined is just a C source file. This file should be named equations.c and include all function definitions for the above functions. The last file that should be defined is the main.c source file. This file will contain the main ( ) function or driver for the program.

IV. Expected Results:

The following console window illustrates inputs and outputs that are appropriate for your program. Your program must display the results in a similar form as shown in the window. The window shows possible results, for the given input tests, for the first two equations. Note: all results must be displayed to the **hundredths** place.



```
Please enter the mass and acceleration (both floating-point values) for use in Newton's Second Law: 1000.0 10.25
Newton's Second Law: force = mass * acceleration = 1000.000000 * 10.250000 = 10250.00

Please enter the radius and height (both floating-point values) for use in a volume of cylinder equation: 12.459 1.3995
Volume of a cylinder: volume_cylinder = PI * radius^2 * height = 3.141592 * 12.459000^2 * 1.399500 = 682.48
Press any key to continue . . .
```

This console window shows only a partial view of the results, you will need to display the results for all of the equations!

V. Submitting Assignments:

1. Using Canvas https://canvas.wsu.edu/, please submit your solution to the correct “Programming Assignments” (PA) folder. Your solution should be zipped into a .zip file with the name `<your last name>_PA2.zip` and uploaded. To upload your solution, please navigate to your correct Canvas ***lab*** course space. Select the “Assignments” link in the main left menu bar. Navigate to the correct PA submission folder. Click the “Start Assignment” button. Click the “Upload File” button. Choose the appropriate .zip file with your solution. Finally, click the “Submit Assignment” button.
2. Your project should contain one header file (a .h file), two C source files (which must be .c files), and project workspace.
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

VI. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 3 pts for correct declaration of constant macro(s)
- 14 pts for proper prompts and handling of input (2 pts/equation)
- 35 pts for correct calculation of results based on given inputs (5 pts/equation)
- 21 pts for appropriate functional decomposition or top-down design (3 pts/function)

- 15 pts for applying 3-file format (i.e. 1 .h and 2 .c files) (5 pts/file)
- 12 pts for adherence to proper programming style and comments (must have a comment block at the top of each file, a comment block above each function definition, and some inline comments) established for the class