WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

CptS 121 - Program Design and Development

Programming Assignment 2: A Modular Approach to the Equation Evaluator

Assigned: Wednesday, September 9, 2020
Due: Wednesday, September 16, 2020 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 2019 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 Community 2019
- 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 program you will build a modular equation evaluator (you may want to start from your solution to programming assignment 1). Once again, y
will write a C program that evaluates the equations provided below. The program must prompt the user for inputs to each equation and evaluate the
based on the inputs. All equations should be placed into a single .c file. This means you should ***NOT*** have 7 Visual Studio projects or 7 .c files. 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*, variable *a, shift_int*, *R1*, *R2*, and *R3* are floating-point values. The *plaintext_character* and *encoded_character* variables are characters, and the *a, shift_int, R1*, *R2*, and *R3* variables are integers. PI should be defined as a constant macro (#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. Total series resistance: series_resistance = R1 + R2 + R3, for 3 resistors. *R1*, *R2*, and *R3* are integers.
2. Sales tax: total_sales_tax = sales_tax_rate * item_cost (note: it's OK to show the result beyond the hundredths place, we don't know how to sh
   to the hundredths place yet)
3. Volume of a right rectangular pyramid: volume_pyramid = (l * w * h) / 3, where l and *w* are the length and width of the base, respectively, and *h* is the height of the pyramid.
4. Total parallel resistance: parallel_resistance = 1 / (1 / R1 + 1 / R2 + 1 / R3), for 3 resistors. *R1*, *R2*, and *R3* are integers.

5. Character encoding: encoded_character = (plaintext_character - 'a') + 'A' - shift_int; *shift_int* is an integer (note: what happens if plaintext_character is lowercase? What happens with various *shift_int* values? Please use the ASCII table to help you understand how to interpret the encoded character produced.)
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 = y / (3/17) - z + x / (a % 2) + PI (recall: *a* is an integer; the 3 and 17 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*!):

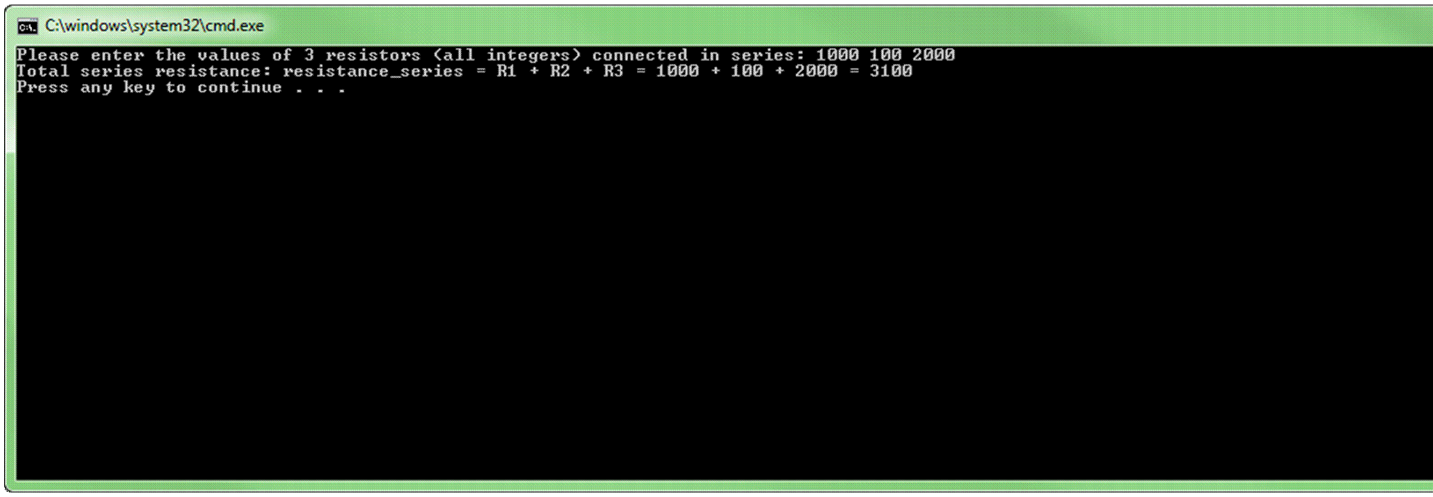
- int calculate_series_resistance (int r1, int r2, int r3)
- double calculate_total_sales_tax (double sales_tax_rate, double item_cost)
- double calculate_volume_pyramid (double l, double w, double h)
- A function for calculating the total parallel resistance (you must decide on a function name, return type, and parameter list!)
- A function for performing the character encoding (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 *calculate_series_resistance ( )* function should evaluate the equation defined as series_resistance = R1 + R2 + R3 and return the resultant resistance, etc. You must print the results to the **hundredths** place. Also, the functions sho 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 values of 3 resistors (all integers) connected in series: 1000 100 2000
Total series resistance: resistance_series = R1 + R2 + R3 = 1000 + 100 + 2000 = 3100
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 Blackboard Learn https://learn.wsu.edu/webapps/login/ submit your assignment. You will submit your assignment in the ***lab*** Blackboard space. Under the "Content" link navigate to the "Programming Assignment Submissions" folder and upload your solution to the appropriate “Assignment” space. You must upload your solution, through an attachment, as <your last name>_pa2.zip by the due date and time.
2. Your .zip file should contain your one header file (a .h file), two C source files (which must be .c files), and project workspace. Delete the debu folder before you zip the project folder.
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 requirement We will grade according to the following criteria:

- 5 pts for correct declaration of constant macro(s)
- 35 pts for proper prompts and handling of input (5 pts/equation)
- 42 pts for correct calculation of results based on given inputs (6 pts/equation)
- 14 pts for appropriate functional decomposition or top-down design (2 pts/function)
- 4 pts for adherence to proper programming style and comments established for the class