CSE 1384 - Recursion
Lab 10

Objectives:

- Continue practicing past concepts
- Practice building recursive functions in C++

Assignment:

Create a program that will recursively build an equation from a given template. Essentially, imagine an equation is being built prompting the user to enter constants and variables. The letter ‘a’ prompts for a constant/numbers, whereas the letter ‘b’ is a prompt for a variable. Then the following equation templates:

- ab
- a + a / b

Could yield equations like:

- 32x
- 2 + 7 / y

Next, imagine that every instance of a parentheses is a sub-equation. Any set of parentheses needs the sub-equation built to contribute to the larger equation. This builds the general idea for your recursive function you’re going to be building.

A starter file has been given that will outline what the return type and parameters should be. It sets things up initially for you to build your program. This also provides a vector of equation templates your function needs to be able to handle. **Do not change what is currently in this program -- merely add to it.** The function is outlined as such:

- buildEq
    - Parameters: string equation, string tab
    - Return value: string

Here’s a sample for clarity’s sake before we jump into the function outline:

```
Starting equation: ((b - b)^2 + (b - b)^2)^1/2

Sub-equation: (b - b)^2 + (b - b)^2
        Sub-equation: b - b
                Enter in a variable: X2
                Enter in a variable: X1
        Sub-equation: b - b
                Enter in a variable: Y2
                Enter in a variable: Y1

Equation built: ((X2 - X1)^2 + (Y2 - Y1)^2)^1/2
```

Recursive calls should be fairly apparent in the display based on indentation (that's where your tab parameter comes into play!).

The purpose for the function is to build the actual equations. A rough outline of the function is as follows:

- Create an initial string, this will be used to build the equation to return
- Loop through the equation (treating the string as an array, as seen in the Cipher lab)
    - If the current character is an 'a' → ask for a number
        - Add to your string
    - If the current character is a 'b' → ask for a variable
        - Add to your string
    - If the current character is a '(' → this is a recursive case
        - First, you'll need to collect your sub-equation
            - Build a string and loop through to add to the string until you get to the appropriate end parentheses
            - *** *Please see hints for more information* ***
        - Display your built sub-equation!
        - Recursively call the buildEq function with the sub-equation and your tab + \t
    - If the character is anything else → add to your string
- Return the built equation string

Hints:

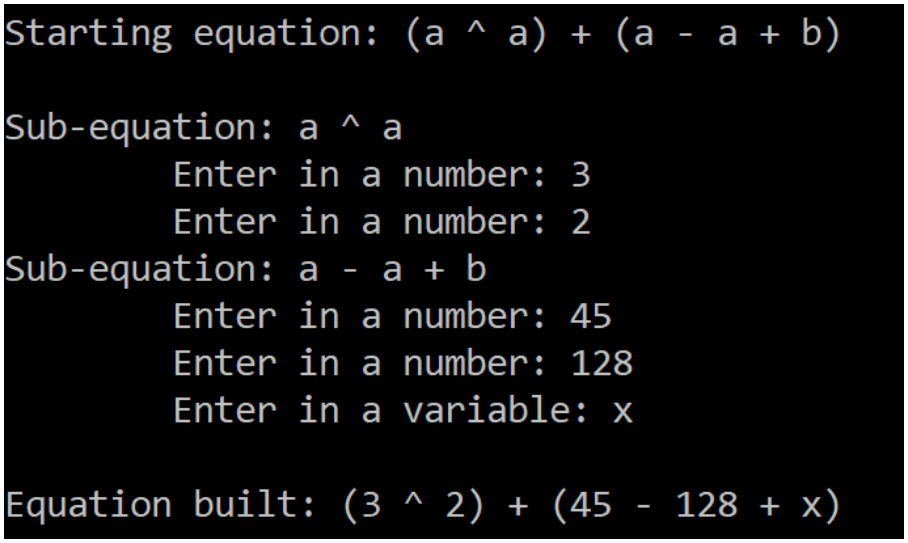
When building your sub-equation, you'll need to loop adding to a string until you reach the corresponding closing parentheses. I recommend keeping track of how many open parenthesis you come across and going from there.

*** Make sure you're not adding the parentheses of the sub-equation you're currently building for – this can lead to an infinite recursive loop. ***

Additionally, when you are building a sub-equation, you'll need to make your initial loop start at the **END** of the sub-equation. Otherwise, you can see some repeating. The size of the sub-equation can help with this!

Example Executions:

(Here's another sample of one of the equation templates in the vector provided to you)



Comment Block:

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
 Name: <your name>                          NetID: <your netID>
 Date: <current date>                       Due Date: <enter in due date>

 Description: <What is the program?>
*/
```

Deliverables:

- C++ code (.cpp file)
- A document (.pdf) with screenshots showing the program running
    - Some of the numbers must contain multi-digit numbers to show replacement isn't occurring

Point Breakdown:
(100 points total)
A submission that doesn't contain any code will receive a 0.

- 10pts - correctly implemented recursion
- 15pts - equations
    - 10pts - equation correctness
    - 5pts - extra characters are added appropriately to the equation
        - Spaces, operations, parentheses, etc
- 25pts - input / output
    - 5pts - correctly asks for numbers or variables depending on a/b
    - 5pts - accepts multi-character numbers/variables
    - 10pts - correctly adds indentation depending on recursion levels
    - 5pts - displays the sub-equation to the user once built
- 30pts - recursive operations
    - 5pts - does not include the parentheses of the sub-equation itself
    - 5pts - ends at the CORRECT ending parenthesis
    - 5pts - does not repeat any parts handled by recursive operations
    - 5pts - correctly adds returned value to equation
    - 10pts - can add an undetermined amount of nested operations via recursion
- 10pts - execution screenshots
    - MUST show all equation templates (6 in total)
    - Must show multi-character acceptance
- 10pts - programming style *
- (penalty) -15pts - changing the program given (changing the parameters, etc)

* Programming style includes good commenting, variable nomenclature, good whitespace, etc.