CS 2124 Data Structures

Programming Project 1 – Infix to Postfix

In this project, we will implement the stack data structure and use it to compute the answer to infix equations. We will do this by using a stack to convert the infix equation to a postfix equation, and then we will use a stack to evaluate the postfix equation.

**Files provided in the project:**

1. **stack.h**. This file does not need to be changed, but you should understand its contents. This file contains structure definitions and the prototypes for the following stack functions:
   1. **Stack newStack(int maximumStackSize).** Malloc a new StackImp, malloc an array of Elements of size maximumStackSize (store the address in stackElements pointer), set maxSize to be maximumStackSize, initialize count to be 0, and return a pointer to the StackImp.
   2. **void freeStack(Stack s).** Free the array inside the StackImp and then free the memory allocated for the stack s.
   3. **void push(Stack s, Element e).** Given a Stack and an Element, push the element onto the stack and increment the count. Print an error message if the stack was already full.
   4. **Element pop(Stack s).** Given a stack s, remove the top element from the stack, decrement count, and return its value.
   5. **int isEmpty(Stack s).** Given a stack, return true (1) if it is empty and return false (0) otherwise.
   6. **Element topElement(Stack s).** Given a stack, return the value of the element that is on top of the stack (without removing it).

**2)** **stack.c**. In this file, you should provide the function definitions for each of the stack functions listed above.

1. **p1Input.txt** This file contains some example infix equations for your code to solve. Each equation contains at least one of 4 operations: addition, subtraction, multiplication, and division with the corresponding symbols: +, -, \*, /*.* You can assume that each operand is a single-digit integer (0 – 9) and that division is integer division (so any fractional portion is dropped, e.g., 5 / 2 = 2). There may or may not be parentheses in the equation. Note that this is just a sample input file. When we grade your programs, we will use different equations (same format). Feel free to modify this file to test your program.
2. **p1Output.txt** The output you should be generating for p1Input.txt**.**

**5) abc123Project1.c** You should rename this file to use your own abc123 prior to submitting on Blackboard. This file contains the main function. Note that you do not need to modify main, only the two functions after main. You can modify main if you’d like to, but it is not necessary for this project. Inside of main, there is code that will open the p1Input.txt file, and it reads the file one line at a time until it reaches the end of the file. The line is stored as a null-terminated character array named *infixString*. We then pass this string to the first function that you are to implement:

**int convertToPostfix(char \*infixString, char \*postfixString)**

In this function, you are given pointers to two strings as input. The first one, infixString, is the infix equation that we just read from the file in main. The second one, postfixString, is an array where you should write the converted postfix string. The function returns an integer depending on if there are any errors that occurred in the formatting of the infix string. See function header in the file for more details. For full credit, your program should detect errors in the parentheses in the infix equation. **For extra credit**, your program should detect missing operators and missing operands.

When the function returns to main, we check to see if there was an error in the infix string. If there was an error, we print an error message and move onto the next infix string. If there was no error, then we pass your string in postfix representation to the second function that you are to implement:

**int evaluatePostfix(char \*postfixString)**

In this function, we take as input your string variable containing the equation in postfix form, and we evaluate it. The function returns an integer containing the computed value of the equation.

**6) Makefile**. Update the makefile to reflect your abc123. You can implement your code however you like, however before submitting ensure your project compiles on the fox servers using this makefile. If it does not compile using the makefile then you will get 0 points! You can compile using the command*make* and you can execute your program using *./project1*.

**Submitting**

Zip up your project folder and submit on the dropbox on Blackboard by the due date.